ulm university universität
uulm

Universität Ulm | 89069 Ulm | Deutschland
Fakultät für Ingenieurwissenschaften und Informatik
Institut für Neuroinformatik
Direktor: Prof. Dr. Günther Palm

Semi-Supervised Learning with Committees: Exploiting Unlabeled Data Using Ensemble Learning Algorithms

Dissertation zur Erlangung des Doktorgrades
Doktor der Naturwissenschaften (Dr. rer. nat.)
der Fakultät für Ingenieurwissenschaften und Informatik
der Universität Ulm

vorgelegt von
Mohamed Farouk Abdel Hady
aus Kairo, Ägypten

Ulm, Deutschland
2010

Amtierender Dekan der Fakultät für Ingenieurwissenschaften und Informatik:
Prof. Dr. Klaus Dietmayer

Vorsitzender des Promotionsausschusses: Prof. Dr. Uwe Schöning

Mitglieder des Promotionsausschusses:
Prof. Dr. Michael Weber
Prof. Dr. Heiko Neumann

Die Gutachter der Dissertation:
Prof. Dr. Günther Palm
Prof. Dr. Wolfgang Minker
Prof. Dr. Barbara Hammer

Tag der Promotion: 8. Februar 2011

ulm university universität
uulm

University of Ulm | 89069 Ulm | Germany
Faculty of Engineering and Computer Science
Institute of Neural Information Processing
Director: Prof. Dr. Günther Palm

Semi-Supervised Learning with Committees: Exploiting Unlabeled Data Using Ensemble Learning Algorithms

A thesis submitted to
Faculty of Engineering and Computer Science
at University of Ulm
in fulfillment of the requirements for the degree of
Doctor of Philosophy in Science (Dr. rer. nat.)

by
Mohamed Farouk Abdel Hady
from Cairo, Egypt

Ulm, Germany
2010

Dean of the Faculty of Engineering and Computer Science:
Prof. Dr. Klaus Dietmayer

Chairman of the doctoral committee: Prof. Dr. Uwe Schöning

Members of the doctoral committee:
Prof. Dr. Michael Weber
Prof. Dr. Heiko Neumann

Reviewers of the dissertation:
Prof. Dr. Günther Palm
Prof. Dr. Wolfgang Minker
Prof. Dr. Barbara Hammer

Day of Conferral of Doctorate: February 8, 2011

Zusammenfassung

Überwachtes maschinelles Lernen ist ein Teilgebiet der Künstlichen Intelligenz, das sich mit dem automatischen Lernen von Vorhersagemodellen aus gelabelten Daten befasst. Solche Lernansätze sind nützlich für viele interessante reale Anwendungen, insbesondere für Aufgaben bei der automatischen Klassifikation, dem Information-Retrieval oder dem Data Mining aus großen Datensammlungen von Texten, Bildern und Videos.

Im traditionellen überwachten Lernen, benutzt man gelabelte Daten um das Vorhersagemodell zu bestimmen. Allerdings ist die Annotation der Trainingsdaten mit Lehrersignalen für reale Anwendungen oft schwierig, kosten- und auch zeitintensiv, da ein menschlicher Experte mit Erfahrung und der notwendigen Ausbildung in der Anwendungsdomäne gebraucht wird. Dies gilt vor allem für Anwendungen mit einer großen Klassenzahl, besonders dann wenn starke Ähnlichkeiten zwischen den Klassen vorhanden sind.

Semi-überwachtes Lernen (SSL) löst diesen inhärenten Engpass, durch die Integration von ungelabelten Daten in den überwachten Lernprozess. Das Ziel ist es, die Klassifikationsleistung des Modells durch diese bisher nicht annotierten Datenpunkte zu steigern, bei gleichzeitiger Reduzierung des Labeling-Aufwandes durch menschliche Experten. Die Forschungen im Bereich des semi-überwachten Lernens lassen sich in vier Hauptrichtungen unterteilen: SSL mit Graphen, SSL mit generativen Modellen, Semi-überwachte Support-Vektor-Maschinen und SSL mit Ensembles. Semi-überwachtes Lernen und Ensemble-Lernen sind zwei wichtige Paradigmen des maschinellen Lernens, die sich fast parallel, aber mit unterschiedlichen Philosophien entwickelt haben. Semi-überwachtes Lernen versucht die Klassifikationsleistung durch die Nutzung ungelabelter Daten zu steigern, dagegen wird im Ensemble-Lernen versucht, das gleiche Ziel durch die Verwendung mehrerer Prädiktoren zu erreichen.

In dieser Dissertation fokussiere ich auf SSL mit Ensembles (SSL durch Disagreement) und vor allem auf "Co-Training" Algorithmen. "Co-Training" ist ein oft angewendeter SSL-Algorithmus der von Blum und Mitchell im Jahr 1998 in die

Literatur eingeführt wurde. Er setzt voraus, dass jede Instanz durch zwei oder mehrere Merkmalsmengen repräsentiert ist, die auch "Views" genannt werden. Jeder "View" muss hinreichend zum Lernen des Modells sein und alle "views" sollen unabhängig sein. In diesem Zusammenhang habe ich einige zentrale Problemstellungen bei der Kombination von Ensemble-Lernen und semi-überwachten Lernen identifiziert, die ich in der vorliegenden Dissertation bearbeitet habe. Hierbei diskutiert ich insbesondere Aufgabenstellungen mit großer Anzahl von Klassen und mit vielen Instanzen, die multimodal repräsentiert sind. Kann "Co-Training" angewendt werden, wenn keine natürliche Merkmalsaufspaltung vorliegt? Wie kann man mehrere Klassifikatoren für das "Co-Training" effektiv konstruktieren? Wie berechnet man einen Konfidenzwert zur Klassifikation bzw. Vorhersage? Für den Fall, das es Beziehungen und Ähnlichkeiten zwischen den Klassen gibt, können diese Beziehungen im SSL gelernt oder ausgenutzt werden? Wie kann die Dempster-Shafer-Kombinationsmethode zur Konfidenz-Bestimmung eingesetzt werden? Können hierarchische neuronale Netze als Klassifikatoren ungelabelter Daten verwendet werden? Kann durch aktives Lernen die Performanz semi-überwachter Lernverfahren verbessert werden? Kann SSL mit Ensembles auf Regressionsaufgaben übertragen werden?

Ich habe ferner Fragen im Bereich des Ensemble-Lernens diskutiert, die in einem engen Zusammenhang mit den von mir studierten SSL Verfahren stehen. Führen trainierbare Kombinierer gegenüber festen Kombinationsabbildungen in hierarchischen Ensembles yu verbesserten Klassifikationsraten? Lässt sich die Performanz hierarchischer Klassifikatoren durch Ensembles steigern? Lassen sich informationstheoretische Betrachtungen nutzen um die Größe eines Ensembles zu reduzieren? Die Diskussion dieser Fragestellungen zeigt unmittelbar den Nutzen der semi-überwachten Lernverfahren in komplexen realen maschinellen Lernverfahren.

Abstract

Supervised machine learning is a branch of artificial intelligence concerned with learning computer programs to automatically improve with experience through knowledge extraction from examples. It builds predictive models from labeled data. Such learning approaches are useful for many interesting real-world applications, but are particularly useful for tasks involving the automatic categorization, retrieval and extraction of knowledge from large collections of data such as text, images and videos.

In traditional supervised learning, one uses "labeled" data to build a model. However, labeling the training data for real-world applications is difficult, expensive, or time consuming, as it requires the effort of human annotators sometimes with specific domain experience and training. There are implicit costs associated with obtaining these labels from domain experts, such as limited time and financial resources. This is especially true for applications that involve learning with large number of class labels and sometimes with similarities among them.

Semi-supervised learning (SSL) addresses this inherent bottleneck by allowing the model to integrate part or all of the available unlabeled data in its supervised learning. The goal is to maximize the learning performance of the model through such newly-labeled examples while minimizing the work required of human annotators. Exploiting unlabeled data to help improve the learning performance has become a hot topic during the last decade and it is divided into four main directions: SSL with graphs, SSL with generative models, semi-supervised support vector machines and SSL by disagreement (SSL with committees). It is interesting to see that semi-supervised learning and ensemble learning are two important paradigms that were developed almost in parallel and with different philosophies. Semi-supervised learning tries to improve generalization performance by exploiting unlabeled data, while ensemble learning tries to achieve the same objective by using multiple predictors.

In this thesis, I concentrate on SSL by disagreement and especially on Co-Training style algorithms. Co-Training is a popular SSL algorithm introduced by

Blum and Mitchell in 1998. It requires that each instance is represented by two or more sets of features that are called views. Each view must be sufficient for learning and all views must be independent. I explore several important questions regarding how to exploit different ensemble learning algorithms in SSL for tasks involving large number of classes and instances that has either single or multiple representations. How can Co-Training algorithm be applied if there is not a natural feature splitting? How to construct multiple classifiers to be co-trained effectively? How to measure confidence in class label prediction? If there is relationships and similarities among classes, can these relationships be learned and exploited during SSL? How can the Dempster-Shafer evidence-theoretic combiner be appropriate for confidence measure? How can hierarchical neural network classifiers exploit unlabeled data to improve the accuracy of image classification? How can active learning improve the performance of semi-supervised learning with committees? How can SSL with committees be extended to regression tasks? I investigate other questions that are indirectly related to SSL. How can a trainable combiner be designed for hierarchical ensembles? Can an ensemble of class hierarchies outperform a single class hierarchy? How can information theory be used to prune ensembles? The answers to the questions illustrate the utility and promise of semi-supervised learning algorithms in complex real-world machine learning systems.

Acknowledgments

First of all, I would like to express my thanks to my advisor Prof. Dr. Günther Palm, the director of the Institute of Neural Information Processing, for accepting me as a doctoral student at his institute, for giving me the right advice at the right time and for carefully reviewing this thesis.

Especially, my deepest gratitude goes to my mentor Dr. Friedhelm Schwenker for supporting me with his valuable suggestions, fruitful discussions and constructive criticisms and for carefully reading this thesis. Despite his workload and tight schedule, he has cooperated with me in writing many papers.

Many thanks to the German Academic Exchange Service (DAAD) whose doctoral scholarship financed my thesis. I would like to thank all the DAAD co-workers both in Cairo and in Bonn for the excellent organization of the scholarship. They have done a lot of effort to prepare me, through German courses, seminars and consultation. After the arrival and during my residence in Germany, they support me in all aspects. Especially I would like to express my gratitude to Frau Margret Leopold as she is always reachable and solves any faced problem.

Also, I would like to express my thanks to the German Science Foundation (DFG) for supporting the publication of my papers and conferences attendance through the funding of both the project *"Learning with Fuzzy Teacher Signals in Neural Multiple Classifier Systems"* (under grant SCHW623/4-3) and the Transregional Collaborative Research Centre SFB/TRR 62 *"Companion-Technology for Cognitive Technical Systems"*.

Last, but not least, I am grateful to my family for all their patience, support and loving. My mother for her unlimited support and guidance throughout my life. She has never stopped believing in me and encouraging me to finish my studies. My son Ahmed and my daughter Heba who make the nights shorter but my days much brighter. My marvelous wife Marwa for her encouragement and never-ending love. To them is all my love and prayers.

Ulm, February 2011 Mohamed F. Abdel Hady

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

Machine learning is the study of how to learn computer programs to automatically improve with experience through knowledge extraction from examples. They improve by becoming better at making decisions, explaining observations, or predicting outcomes. For instance, computers have been learned to interpret human speech by learning from vocal recordings that have been annotated for words and sentences [37, 154]. They have been learned to diagnose diseases by analyzing profiles of healthy and unhealthy patients [119]. They have been learned to interpret handwritten digits and characters [136] and to analyze the contents of videos and images [212]. They have been learned to recognize hand gestures and facial expressions [190, 64, 169]. They have been learned to filter spam emails through analyzing thousands of legal and spam emails [96]. Generally, the learning algorithms [58] used for these tasks fall into two groups:

- *Unsupervised learning*: The learning algorithm is given a collection of unlabeled data. The goal is to organize aspects of the collection in some way. For instance, clustering data points, called examples, into natural groups based on a set of observable features.

- *Supervised learning*: The learning algorithm is given a collection of labeled instances, each denoted by the pair (x, y). The goal is to construct a model that can predict the output y for any new example x, based on a set of features that describe it. When y belongs a set of discrete values, the task is called *classification*, when it is a real number, the task is called *regression*.

This thesis is concerned with applications that can be approached as supervised learning problems.

1.1 Semi-Supervised Learning

Supervised learning algorithms require a large amount of labeled training data in order to construct models with high prediction performance, see Figure 5.1.

In many practical data mining applications such as computer-aided medical diagnosis [119], remote sensing image classification [175], speech recognition [95], email classification [96], or automated classification of text documents [139, 140], there is often an extremely inexpensive large pool of unlabeled data available. However, the data labeling process is often difficult, tedious, expensive, or time consuming, as it requires the efforts of human experts or special devices. Due to the difficulties in incorporating unlabeled data directly into conventional supervised learning algorithms such as support vector machines and neural networks and the lack of a clear understanding of the value of unlabeled data in the learning process, the study of semi-supervised learning attracted attention only after the middle of 1990s. As the demand for automatic exploitation of unlabeled data increases, semi-supervised learning has become a hot topic.

In the machine learning literature, there are mainly three paradigms for addressing the problem of combining labeled and unlabeled data to boost the performance: semi-supervised learning, transductive learning and active learning. *Semi-supervised learning* (*SSL*) refers to methods that attempt to take advantage of unlabeled data for supervised learning, see Figure 5.4, or to incorporate prior information such as class labels, pairwise constraints or cluster membership in the context of unsupervised learning. *Transductive learning* refers to methods which also attempt to exploit unlabeled examples but assuming that the unlabeled examples are exactly the test examples. *Active learning* [173] refers to methods which assume that the given learning algorithm has control on the selection of the input training data such that it can select the most important examples from a pool of unlabeled examples, then an oracle such as a human expert is asked for labeling these examples, where the aim is to minimize data utilization.

The recent research of the machine learning community on semi-supervised learning (*SSL*) concentrates into four directions: semi-supervised classification [30, 140, 96, 210, 215, 119], semi-supervised regression [214], semi-supervised clustering such as constrained and seeded k-means clustering [195, 181, 19] and semi-supervised dimensionality reduction [20, 218]. Interested readers in recent advances of *SSL* are directed to the literature survey of Zhu [219]. Many semi-supervised classification algorithms have been developed. They can be divided into five categories according to Zhu [219]: (1) *Self-Training* [139], (2) semi-supervised learning with generative models [131, 140, 175], (3) S3VMs (Semi-Supervised Support Vector Machines) [88, 39, 73, 115], (4) semi-supervised learning with graphs [23, 212, 220], and (5) semi-supervised learning with committees (semi-supervised by disagreement) [30, 140, 96, 210, 215, 119, 213].

The goal of this thesis is to demonstrate that supervised classification with committees using a small amount of labeled data and a large number of unlabeled examples create more accurate classifier ensembles. In general, unlabeled examples are much less expensive and easier to collect than labeled examples. This is particularly true for image, audio and video classification tasks involving online data sources, such as remote-sensing images [175], speech signals [95] and medical

diagnosis [119], where huge amounts of unlabeled content are readily available. Collecting this content can frequently be done automatically, so it is feasible to quickly gather a large set of unlabeled examples. If unlabeled data can be integrated into supervised learning then building pattern recognition systems will be significantly faster and less expensive than before.

1.2 Thesis Statement

This thesis asks and answers several research questions. This section summarizes and formulates these questions. The second part of the thesis will answer these questions where the answers are evaluated through several statistical experiments.

Q1: How can Co-Training be applied if there is not a natural feature splitting? *Co-Training* is a popular *semi-supervised learning* algorithm that requires each example to be represented by multiple sets of features (views) where these views are sufficient for learning and independent given the class. However, these requirements are hard to be satisfied in many real-world domains because there are not multiple representations available or it is computationally inefficient to extract more than one feature set for each example. *Co-Training* does not perform so well without an appropriate feature splitting [139]. I investigate single-view *Co-Training* style algorithms that do not require redundant and independent views (see Chapter 9).

Q2: How to construct multiple classifiers to be co-trained effectively? This question is a more general form of question Q1. Wang and Zhou [197] provided a theoretical analysis that emphasizes that the important factor for the success of disagreement-based single-view *Co-Training style* algorithms is the creation of a large diversity (disagreement) among the co-trained classifiers, regardless of the method used to create diversity, for instance through: sufficiently redundant and independent views as in standard *Co-Training* [30, 139], artificial feature splits in [62, 162], different supervised learning algorithms as in [71, 210], training set manipulation as in [24, 215] or feature set manipulation as in [119] or different parameters of the same supervised learning algorithm as in [214]. Note that Brown et al. presented in [36] an extensive survey of the various techniques used for creating diverse ensembles, and categorized them, forming a preliminary taxonomy of diversity creation methods. One can see that multi-view Co-Training is a special case of semi-supervised learning with committees. Therefore, the data mining community is interested in a more general *Co-Training style* framework that can exploit the diversity among the members of an ensemble for correctly predicting the unlabeled data in order to boost the generalization ability of the ensemble. I investigate how to create good classifiers for *Co-Training* such as

applying the random subspace method or Bagging (see Chapter 8, Chapter 9, Chapter 11 and Chapter 12).

Q3: How to measure confidence in label prediction? An important factor that affects the performance of any *Co-Training* style algorithm is how to measure the confidence in predicting the class label of an unlabeled example which determines its probability of being selected. An inaccurate confidence measure can lead to selecting and adding mislabeled examples to the labeled training set which leads to performance degradation during the SSL process. Often the *Co-Training* style algorithm depends on class probability estimates in order to measure confidence. I will study how to improve the class probability estimates of the co-trained classifiers such as hierarchical neural networks, decision trees, k-nearest neighbor classifiers, RBF neural network regressors and support vector machines (see Sections 8.2.1, 8.3.1, 9.2.2, 11.2.2 and 12.2.3, respectively).

Q4: How can the Dempster-Shafer evidence-theoretic combiner be used for confidence measure? This question is special case of question Q3 when hierarchical multi-class decomposition is used. There are many reasons for selecting this theory in the context of hierarchical multiple classifiers combination. It can discriminate between ignorance and uncertainty. Since it is able to assign evidence not only to atomic but also to subsets and intervals of hypotheses, it easily represents evidences at different levels of abstraction. It can combine evidences provided by different sources and can measure the conflict among them. I investigate to measure the prediction confidence of hierarchical ensembles based on the class probability estimates provided by the Dempster-Shafer evidence-theoretic combination method (see Section 8.2).

Q5: How can hierarchical neural network classifiers explpot unlabeled data to improve the classification accuracy? The hierarchical neural network classifiers as any supervised learning algorithm perform well when there is a sufficient amount of labeled data. Most of the *Co-Training* related work used just two classifiers with a natural feature splitting. I investigate *semi-supervised learning* algorithms to exploit the abundant unlabeled data to improve the generalization ability when the available labeled data is scarce. I demonstrate that *Co-Training* is a helpful and valid approach to use unlabeled data for image classification (see Chapter 8).

Q6: How can active learning improve the performance of semi-supervised learning with committees? Both *semi-supervised learning* and *active learning* tackle the same problem but from different directions. That is, they aim to improve the generalization error and at the same time minimize the cost of data annotation through exploiting the abundant unlabeled data. *Semi-supervised*

learning exploits the unlabeled examples where the underlying classifiers are *most confident* in the prediction of their class labels. They depend on a given confidence measure for sample selection. On the other hand, *active learning* exploits the *most informative* unlabeled examples where the underlying classifiers disagree on the prediction of their labels (contention points). I study how to combine the advantages of committee-based *semi-supervised learning* and *active learning* (see Chapter 10).

Q7: How can semi-supervised learning with committees be extended to regression tasks? Although the success of semi-supervised learning for classification, there is not much work on SSL for regression. For classification, it is a straightforward task because many classifiers can estimate class posterior probabilities such as Naive Bayes classifier or return real-valued outputs that can be transformed to class probability estimates such as neural networks and decision trees. Assuming that a classifier estimates the probability that an instance x_1 belongs to classes ω_1 and ω_2 is 0.9 and 0.1, respectively, while that for an instance x_2 is 0.6 and 0.4, respectively, then the classifier is more confident that x_1 belongs to classes ω_1 than x_2. Therefore, a *labeling confidence* can be assigned to each unlabeled example using its class probability distribution. For regression, the mechanism for estimating the confidence is a challenge because the number of possible predictions in regression is unknown. Krogh and Vedelsby [103] proposed to use variance as an effective selection criterion for active learning because a high variance between the estimates of the ensemble members leads to a high average error. Unfortunately, a low variance does not necessarily imply a low average error. That is, it can not be used as a selection criterion for SSL because agreement of committee members does not imply that the estimated output is close to the target output. I investigate how to extend the ideas of semi-supervised learning with committees to regression tasks (see Chapter 11).

Q8: How to design a trainable combiner that outperforms non-trainable ones for hierarchical ensembles? A key factor for the design of an effective ensemble is how to combine its member outputs to give the final decision. Although there are various methods to build the class hierarchy (first stage) and to solve the underlying binary-class problems (second stage), there is not much work to develop new combination methods that can best combine the intermediate results of the binary classifiers within the hierarchy (third stage). The simple aggregation rules used for flat multiple classifier systems such as *minimum*, *maximum*, *average*, *product* and *majority vote* can not be applied to *hierarchical decision profiles*. I introduce a novel fusion method for hierarchical neural network classifiers (see Chapter 13).

Q9: Can an ensemble of class hierarchies outperform a single class hierarchy? An ensemble can outperform its individual classifiers if these classifiers are diverse and accurate. Dietterich [55] suggested statistical, computational and representational reasons why it is possible to construct an ensemble of classifiers that is often more accurate than a single classifier. These three fundamental reasons represent the most important shortcomings of existing base learning algorithms. Hence, the aim of an ensemble method is to alleviate or eliminate these shortcomings. The use of multiple classifiers allows to exploit the complementary discriminating information that these classifiers may provide. Therefore, the objective of combining such a group of classifiers is approximate the best classifier by producing a more accurate classifier decision than a single classifier. I investigate the generation of a set of class hierarchies based on a set of representations. In order to construct diverse individual classifiers, I assume that the object to be classified is described by multiple feature sets (views). The aim is to construct different class hierarchies using different combinations of views to improve the accuracy of the multi-class learning. In addition, I arise the question: "can soft combination methods outperform majority vote when evidence-theoretic framework is used to retrieve the decision of each class hierarchy?" (see Chapter 14).

Q10: How can information theory be used to prune ensembles? Typically, ensemble learning methods comprise two phases: the construction of multiple individual classifiers and their combination. Recent work has considered an additional intermediate phase that deals with the reduction of the ensemble size prior to combination. This phase has several names in the literature such as ensemble pruning, selective ensemble, ensemble thinning and classifier selection, the last one of which is used within this chapter. Classifier selection is important for two reasons. The first reason is classification accuracy. An ensemble may consist not only of accurate classifiers, but also of classifiers with lower predictive accuracy. Pruning the poor-performing classifiers while maintaining a good diversity of the ensemble is typically considered as the main factor for an effective ensemble. The minimization of classification time complexity is crucial in certain applications, such as stream mining. Thus the second reason is equally important, efficiency. Having a very large number of classifiers in an ensemble adds a lot of computational overhead. For instance, decision trees may have large memory requirements and lazy learning methods have a considerable computational cost during classification phase. Recently an information-theoretic view was presented for feature selection [35]. It derives a space of possible selection criteria and show that several feature selection criteria in the literature are points within this continuous space. I investigate to export this information-theoretic view to solve the open issue of classifier selection. The objective is to improve the efficiency of semi-supervised learning with committees through select the most accurate and diverse classifiers that can further undergo *Co-Training* (see Chapter 15).

1.3 Outline of the Thesis

This thesis is organized into two parts. The first part describes the basics and the theoretical foundations of the new methods proposed in this thesis. This part is organized as follows:

- Chapter 2 presents the central building blocks of the learning paradigms proposed in the second part chapters of this thesis. It contains introductory sections for radial basis function neural networks, k-nearest neighbor classifiers, decision trees and support vector machines.

- Chapter 3 provides an overview of ensemble learning in general, as well as the particular ensemble methods using in the contributions part of this thesis.

- Chapter 4 explores the different techniques in the literature to decompose a multi-class problem into a set of binary problems. In particular, it details the hierarchical tree-structured approach that will be used later in this thesis.

- Chapter 5 presents an overview of semi-supervised learning. It provides a taxonomy of the existing semi-supervised learning paradigms in general and in particular a survey on the recently developed semi-supervised algorithms that are based on ensemble learning.

- Chapter 6 provides an overview of active learning and especially the committee-based active learning algorithm. It presents the existing informativeness measures used by different active learners.

- Chapter 7 presents the real-world object recognition tasks used in this thesis. It describes the used feature extraction procedures such as principle component analysis, color histogram and orientation histogram. In addition, the cross validation technique and significance tests especially t-test are presented as they are used for performance evaluation.

The second part describes the main contributions of this thesis. It is organized as follows:

- Chapter 8 proposes two novel frameworks for multiple-view semi-supervised learning. These settings can reduce the annotation effort required for tasks such as image classification when each image is represented by multiple sets of features that provide different information.

- Chapter 9 introduces a new single-view semi-supervised framework that requires an ensemble of diverse classifiers instead of redundant and independent views required by the traditional *Co-Training* algorithm. These

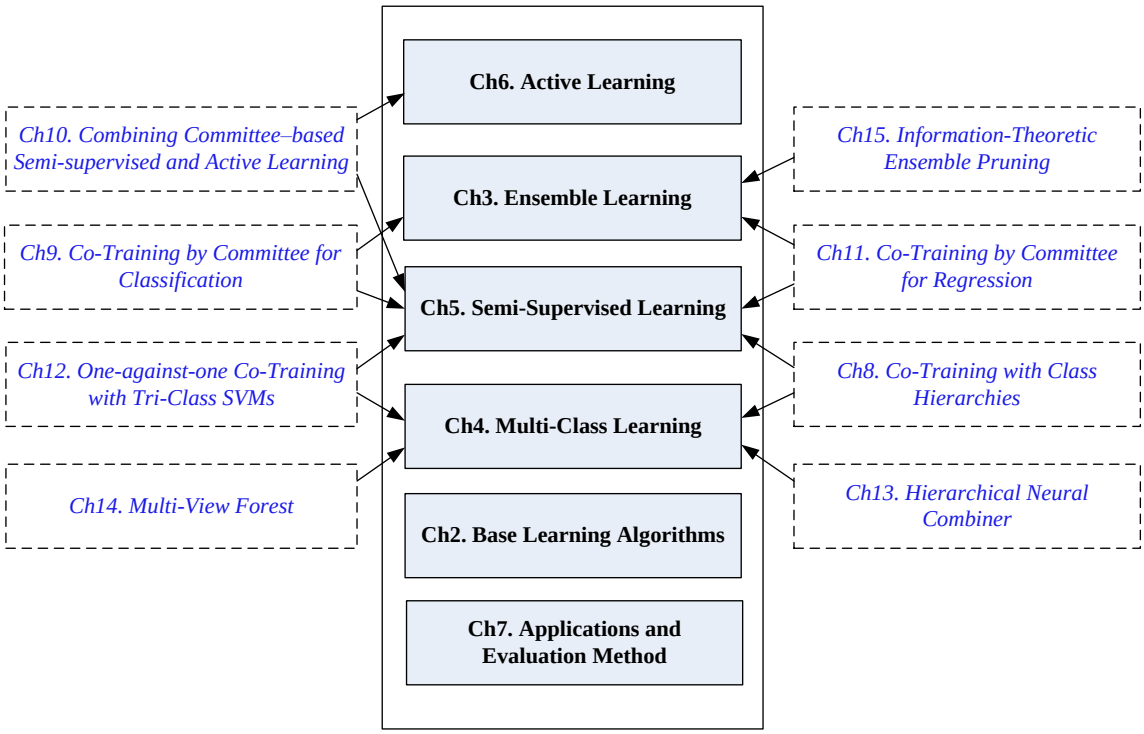


Figure 1.1: Graphical illustration of the organization of the thesis

settings can reduce the annotation cost required for tasks where each pattern is represented by only a single set of features.

- Chapter 10 proposes two possible combinations of committee-based active learning and the committee-based semi-supervised learning framework introduced in Chapter 9.

- Chapter 11 extends the idea of semi-supervised learning with committees from classification to regression tasks.

- Chapter 12 introduces a multi-view semi-supervised learning framework that is based a newly developed version of support vector machine (SVM). It includes an extended version of Sequential Minimal Optimization that is used for fast learning of SVM. In addition, it presents a probabilistic interpretation of SVM outputs.

The last three chapters of the second part describe some of my additional research in ensemble learning that are not directly related to semi-supervised learning, as follows:

- Chapter 13 introduces a new trainable fusion method for a tree-structured ensemble that integrates statistical information about its individual outputs, in the form of decision templates, into the training of an Radial Basis

Function (RBF) network. In addition, it presents a new similarity measure based on multivariate Gaussian function to match a decision profile with decision templates.

- Chapter 14 proposes a new ensemble method that constructs an ensemble of tree-structured classifiers using multi-view learning. The results indicate that the proposed forest can efficiently integrates multi-view data and outperforms the individual tree-structured classifiers.

- Chapter 15 provides an information-theoretic perspective on the issue of ensemble pruning and classifier selection.

Figure 1.1 illustrates the relationship between part I and part II. Finally, Chapter 16 summarizes the key contributions of this thesis, discusses open problems and future directions in semi-supervised learning, and offers some concluding remarks.

Part I

Basics

Chapter 2

Base Learning Algorithms

2.1 Radial Basis Function Neural Networks

The radial basis function (RBF) networks are artificial neural networks that use radial basis functions as activation functions. They were introduced into the neural network literature by Broomhead and Lowe in [34]. They are used in function approximation, classification, time series prediction, and control. The theoretical basis of the RBF approach lies in the field of interpolation of multivariate functions. The goal of interpolating a set of tuples $\{(x_\mu, y_\mu) : x_\mu \in \mathbb{R}^D, y_\mu \in \mathbb{R}, \mu = 1, \ldots, M\}$ is to find a function $f : \mathbb{R}^D \to \mathbb{R}$ with $f(x_\mu) = y_\mu$ for all $\mu = 1, \ldots, M$. The function f is a linear combination of M radial basis functions, each associated with a data point x_μ and weighted by an appropriate real-valued coefficient w_j:

$$f(x) = \sum_{\mu=1}^{M} w_\mu \phi(\|x - x_\mu\|_p) \tag{2.1}$$

and

$$\|x - x_\mu\|_p = \left(\sum_{i=1}^{D} |x_i - x_{\mu i}|^p \right)^{1/p}, \text{ for } x, x_\mu \in \mathbb{R}^D \tag{2.2}$$

where $\|.\|_p$ denotes the Minkowski distance between two D-dimensional feature vectors x and x_μ, as defined in Eq. (2.2) where $p \in [1, \infty)$ is the distance order. In general, the smaller the order, the more robust the resulting distance metric to data variations. Then the interpolation problem is equivalent to the following system of linear equations

$$\Phi w = y \tag{2.3}$$

where $w = (w_1, \ldots, w_M)^T$, $y = (y_1, \ldots, y_M)^T$ and Φ is a square matrix of order M defined by

$$\Phi = [\phi(\|x_\mu - x_j\|_p)]_{\mu,j=1}^{M} \tag{2.4}$$

If matrix Φ is invertible, the solution w of the interpolation problem can be explicitly calculated and has the form:

$$w = \Phi^{-1} y \tag{2.5}$$

Examples of radial basis functions ϕ often used in applications are:

1. Gaussian function

$$\phi(r) = e^{\frac{-r^2}{2\sigma^2}} \text{ for some } \sigma > 0, \text{ and } r \geq 0 \tag{2.6}$$

2. Inverse Multiquadric

$$\phi(r) = \frac{1}{(r^2 + \sigma^2)^{1/2}} \text{ for some } \sigma > 0, \text{ and } r \geq 0 \tag{2.7}$$

3. Thin plate spline

$$\phi(r) = r^2 ln(r) \text{ for some } r \geq 0 \tag{2.8}$$

The most popular and widely used RBF is the Gaussian function, defined in Eq. (2.6), using the L_2-norm which is known as Euclidean distance and given as.

$$\phi_j(x) = \phi(\|x - c_j\|_2) = exp(\frac{-\|x - c_j\|_2^2}{2\sigma_j^2}), \text{ for } j = 1, \ldots, M \tag{2.9}$$

where $c_j \in \mathbb{R}^D$ is called the center or prototype of the j^{th} RBF and $\sigma_j \in \mathbb{R}$ is called the width or the scaling parameter determines how steeply $\phi_j(x)$ decreases with growing the distance between x and the center c_j.

The target solution of the interpolation problem is typically a function passes through every data point (x_μ, y_μ). But in the presence of noise, the solution of the problem is a function oscillating between the given data points. An additional problem with the RBF approach for interpolation is that the number of basis functions is equal to the number of data points. As a result, calculating the inverse of the $M \times M$ matrix Φ becomes computationally expensive.

Broomhead and Lowe [34] proposed to reduce the number of basis functions $J << M$ and to place the basis functions at centers c_j instead of the training examples x_μ in order to reduce the computational complexity. Thus the decision function can be written as,

$$f(x) = \sum_{j=1}^{J} w_j \phi(\|x - c_j\|_p). \tag{2.10}$$

This technique produces a solution by approximating the data points instead of interpolating them.

In addition, in [34] an interpretation of the RBF approach as an artificial neural network is given. An RBF network typically has three layers as shown in Figure 2.1: an input layer contains D input neurons to feed an input feature vector into the network; a hidden layer of J non-linear RBF neurons as activation functions; and a layer of K output neurons, calculating a linear combination of the basis functions. Under some conditions on the basis function ϕ, RBF networks are universal approximators. This means that an RBF network with enough hidden neurons can approximate any continuous function with arbitrary accuracy [142]. This implies that RBF networks with adjustable prototypes can also be used for classification tasks [149]. For classification, the RBF network has to perform a mapping from a continuous input space $\mathbb{R}^D$ into a finite set of classes $\Omega = \{\omega_1, \ldots, \omega_K\}$, where K is the number of classes. In the training phase, the parameters of the network are determined from a finite labeled training set

$$L = \{(x_\mu, y_\mu) : x_\mu \in \mathbb{R}^D, y_\mu \in \Omega, \mu = 1, \ldots, M\} \tag{2.11}$$

where each feature vector x_μ is associated with a single class label y_μ. In the classification phase, the network is applied to an unlabeled example to predicted its class label. The output layer of an RBF network has an output unit for each class in Ω, and using the *1-of-K* encoding scheme, the class label of each training example $y_\mu \in \Omega$ is encoded into a K-dimensional binary vector $t_\mu \in \{0, 1\}^K$ through the relation $t_{\mu k} = 1$ iff $y_\mu = \omega_k$. Note that, other encoding schemes can be used but they are not common in pattern recognition applications. An RBF network with J basis functions is performing a mapping $f : \mathbb{R}^D \to \mathbb{R}^K$. That is,

$$f_k(x) = \sum_{j=1}^{J} w_{jk}\phi_j(x) + w_{0k}, \text{ for } k = 1, \ldots, K \tag{2.12}$$

where the w_{0k} denotes the bias term, which may be absorbed into the summation by including an extra basis function whose activation is set equal to 1 on the whole feature space ($\phi_0(x) = 1$). Typically, in pattern recognition the individual network outputs $f_k(x)$ are interpreted as class membership estimates. Therefore, an example x is assigned the class label ω_{k^*} whose output unit has the maximum activation:

$$k^* = arg \max_{1 \le k \le K} f_k(x) \tag{2.13}$$

Typically, an RBF neural network differs from the RBF approach for interpolation in some ways:

1. The number of basis functions J is much less than the number of training examples M ($J << M$), and the basis function centers $c_j \in \mathbb{R}^D$ are representative examples that are not necessary belonging to the training set.

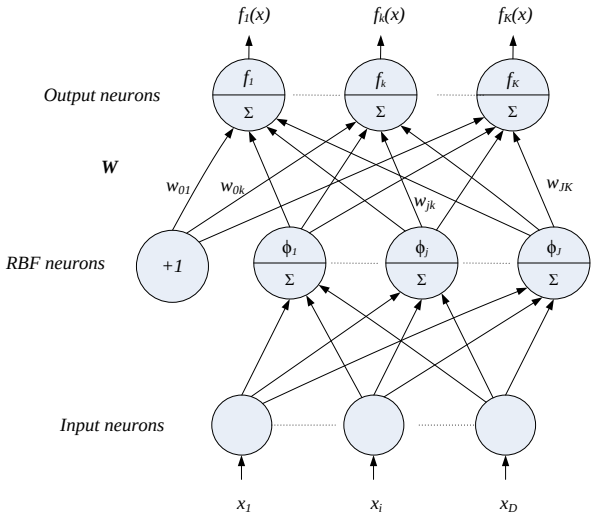


Figure 2.1: An illustration of a radial basis function neural network with D input neurons, J RBF neurons in the hidden layer and K output neurons

2. Instead of a global scaling parameter $\sigma_j \in \mathbb{R}$ for all basis functions, each basis function has its own scaling parameter that is given through scalars, vectors, or matrices $\sigma_j \in \mathbb{R}$, $\in \mathbb{R}^D$, or $\in \mathbb{R}^{D \times D}$.

For a multilayer perceptron (MLP) network, all parameters are usually adapted simultaneously by an optimization procedure. This training procedure is supervised, since it minimizes an error function measuring the difference between the network output and the correct output values. In contrast, there are three schemes have been developed to adjust the RBF centers and scaling parameters and the output layer weights.

2.1.1 One-Phase Learning Scheme

Here, the centers c_j are randomly sampled from the set of training examples L (or all training examples are used as centers). Typically, all the scaling parameters σ_j are set to a predefined real number σ. Thus, only the output layer weights w_{jk} are adjusted through some kind of supervised optimization in order to minimize a given error function (see Subsection 2.1.6).

2.1.2 Two-Phase Learning Scheme

Here, the two layers of the RBF network are trained separately, as follows: the RBF centers c_j and scaling parameters σ_j are determined, then subsequently the output layer is adjusted.

1. First, determine the RBF centers $c_j \in \mathbb{R}^D$ and the scaling parameters σ_j through a supervised or an unsupervised clustering algorithms such as k-means clustering, learning vector quantization (LVQ) or classification trees (see Subsection 2.1.4 and Subsection 2.1.5).

2. Then, adjust the output layer weights $w_{jk} \in \mathbb{R}$ for $j = 1, \ldots, J$ and $k = 1, \ldots, K$, using gradient descent error optimization or pseudo-inverse solution (see Subsection 2.1.6).

2.1.3 Three-Phase Learning Scheme

After the initialization of the RBF network using the two-phase learning scheme, a third training phase for RBF networks [166] is performed in the style of error back propagation learning in MLPs, where all types of parameters are adapted simultaneously. This learning scheme utilizes non-linear optimization and is computationally expensive but yields improved classification results compared to the two-stage learning scheme. If the error function of the network is a differentiable function as the sum-of-squares error,

$$E = \frac{1}{2} \sum_{\mu=1}^{M} \sum_{k=1}^{K} (t_{\mu k} - f_k(x_\mu))^2, \tag{2.14}$$

which is the difference between target output $t_{\mu k}$ and the network output $f_k(x_\mu)$. For a network with differentiable activation functions, a necessary condition for a minimal error is that its derivatives with respect to the kernel location c_j, kernel width Σ_j, and output weights w_{jk} vanish. In case of Gaussian function where Σ_j is a diagonal matrix defined by a vector $\sigma_j \in \mathbb{R}^D$, the learning rules are,

$$w_{jk} = w_{jk} - \eta \sum_{\mu=1}^{M} \phi_j(x_\mu)(t_{\mu k} - f_k(x_\mu)), \tag{2.15}$$

$$c_{ji} = c_{ji} - \eta \sum_{\mu=1}^{M} \phi_j(x_\mu) \frac{x_{\mu i} - c_{ji}}{\sigma_{ji}^2} \sum_{k=1}^{K} w_{jk}(t_{\mu k} - f_k(x_\mu)), \tag{2.16}$$

$$\sigma_{ji} = \sigma_{ji} - \eta \sum_{\mu=1}^{M} \phi_j(x_\mu) \frac{(x_{\mu i} - c_{ji})^2}{\sigma_{ji}^3} \sum_{k=1}^{K} w_{jk}(t_{\mu k} - f_k(x_\mu)) \tag{2.17}$$

for $i = 1, \ldots, D$ and $j = 1, \ldots, J$. Choosing the right learning rate η is a critical issue in neural network training. If its value is too low, convergence to a minimum is slow. On the other hand, if it is selected too high, successive steps in parameter space overshoot the minimum of the error surface. This problem can be avoided by a proper stepwise tuning.

2.1.4 Determine RBF Centers

Clustering and vector quantization techniques are typically used when the data points have to be divided into natural groups and no class labels are available. Here, the aim is to determine a small but representative set of centers or prototypes from a larger data set in order to minimize some quantization error. For classification tasks where the class labels of the training examples are known, supervised vector quantization algorithms, such as Kohonen's learning vector quantization (LVQ) algorithm, can also be used to determine the prototypes.

2.1.4.1 k-means Clustering

The k-means clustering [123, 20] is an unsupervised competitive learning algorithm that partitions a given feature space into k disjoint regions $\Re_1, \ldots, \Re_k$ where each region j is defined as,

$$\Re_j = \{x \in \mathbb{R}^D : j = arg \min_{1 \leq i \leq k} \|x - c_i\|_2\} \tag{2.18}$$

Such a partition of the input space is called a Voronoi tessellation, see Figure 2.2,

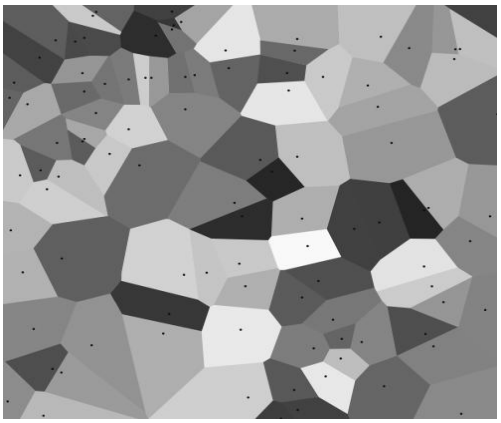


Figure 2.2: An illustration of Voronoi regions in a two-dimensional feature space

where prototype vector c_j is the representative for region $\Re_j$. The loss function defined by

$$E(c_1, \ldots, c_k) = \sum_{j=1}^{k} \sum_{x_\mu \in \mathbb{C}_j} \|x_\mu - c_j\|_2^2 \tag{2.19}$$

is minimal, if each prototype c_j is the center of gravity of data group or cluster $\mathbb{C}_j = \Re_j \cap \{x_1, \ldots, x_M\}$.

Given an initial set of prototypes c_j, $j = 1, \ldots, k$, most often by randomly selecting k examples from the training data set L. For each training example $x_\mu \in L$, first the Euclidean distance between x_μ and all the existing prototypes is

calculated, $d_j = \|x_\mu - c_j\|$. Competition is realized by searching for the nearest prototype for x_μ, $d_{j*} = \min_{1 \le j \le k} d_j$. If the *incremental k-means clustering* is adopted, the winning prototype c_{j*} is then directly updated using the learning rule

$$c_{j*} = c_{j*} + \frac{1}{|\mathbb{C}_{j*}| + 1}(x_\mu - c_{j*}) \tag{2.20}$$

which moves the prototype c_{j*} in the direction of the example x_μ. If the *batch mode k-means clustering* is used, add x_μ into $\mathbb{C}_{j*}$ and after the presentation of all training examples, all the prototypes c_j are adapted through the learning rule

$$c_j = \frac{1}{|\mathbb{C}_j|} \sum_{x_\mu \in \mathbb{C}_j} x_\mu \tag{2.21}$$

This iterative process can be stopped if the sets of data points within each cluster $\mathbb{C}_j$ in two consecutive epochs are not changed.

Supervised k-means clustering. Since the training examples are labeled, for each class $\omega_k \in \Omega$, the above procedure is repeated on the training examples L_k belonging to this class, $L_k = \{x_\mu : (x_\mu, y_\mu) \in L, y_\mu = \omega_k\}$. As a result, n_k clusters are formed for class ω_k where n_k is proportional to the number of examples in L_k, that is,

$$n_k = \alpha \times |L_k|, \text{ where } \alpha \in (0, 1) \tag{2.22}$$

After performing k-means clustering for K times, the prototypes $c_1, \ldots, c_J$ can be used as the initial RBF centers where $J = \sum_{k=1}^{K} n_k$ denotes the total number of prototypes in the hidden layer of the RBF network.

2.1.4.2 Learning Vector Quantization (LVQ)

Learning Vector Quantization (LVQ) [99] is a supervised competitive neural network learning algorithm. The LVQ method can be thought of as a supervised version of the original Self Organizing Maps (SOM) [98]. LVQ network, as shown in Figure 2.3, consists of two layers: an input layer that feeds the examples into the network and a hidden layer with J neurons that are called prototypes or code vectors. The prototype vectors $c_1, \ldots, c_J$ divide the input space into J disjoint regions called Voronoi cells. In the training phase, a training set $L = \{(x_\mu, y_\mu) : \mu = 1, 2, ..., M\}$ is iteratively presented to the network. A set of J randomly selected examples from L is used as an initialization of the prototypes c_j, $j = 1, \ldots, J$. At each iteration, the location of prototype c_j is updated according to its distances to the presented training examples x_μ where a prototype c_j moves in the direction of x_μ if they belong to the same class and moves in the opposite direction otherwise. There are several learning schemes which differ in terms of the definition of the learning rate and the number of prototypes

adapted at each learning epoch. Some of the learning strategies are described below.

In the classification phase, a given example x is assigned to the class label of its nearest prototype c_{j^*} where the nearest prototype is defined as,

$$j^* = arg \min_{1 \leq j \leq J} \|x - c_j\| . \tag{2.23}$$

Note that this is the decision rule adopted by the 1-nearest neighbor classifier (see Section 2.2). Some of the variants of the LVQ learning algorithm are described below while both the initialization and the classification phases are the same for all variants.

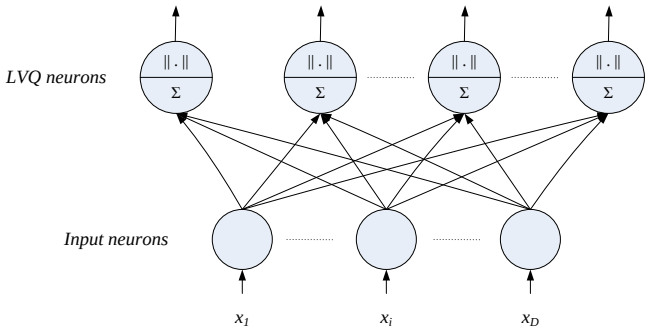


Figure 2.3: An illustration of an LVQ network

LVQ1: This learning algorithm is the basic version of the LVQ algorithm. It is also called winner-takes-all scheme, because only the nearest prototype, called winning neuron, to the presented training example is adapted while the other prototypes remain unchanged. At each iteration t and for each example x_μ, first the distance between x_μ and all prototypes c_j are calculated

$$d_j = \|x_\mu - c_j\| . \tag{2.24}$$

Then the index of the winning prototype c_{j^*} is defined as

$$j^* = arg \min_{1 \leq j \leq J} d_j. \tag{2.25}$$

Only the winning prototype c_{j^*} is updated as defined in the learning rule,

$$c_j(t+1) = \begin{cases} c_j(t) + \eta(t)(x_\mu - c_j(t)) & \text{if } class(c_j) = class(x_\mu), j = j^* \\ c_j(t) - \eta(t)(x_\mu - c_j(t)) & \text{if } class(c_j) \neq class(x_\mu), j = j^* \\ c_j(t) & j \neq j^* \end{cases} \tag{2.26}$$

If the winning prototype and the training example x_μ belong to the same class, the winner neuron is shifted towards x_μ, otherwise it is pushed away. The global learning rate $\eta(t)$, $0 \leq \eta(t) \leq 1$, can either be constant or decrease with time t.

Optimized learning rate LVQ1(OLVQ1): It is a modified version of LVQ1 where each prototype $c_j(t)$ has its own learning rate $\eta_j(t)$ in the learning rule instead of the global $\eta(t)$. This modification accelerates the convergence. This local learning rate is defined as

$$\eta_j(t) = min(\frac{\eta_j(t-1)}{s(t)\eta_j(t-1)+1}, \eta_{max}) \tag{2.27}$$

where $\eta_j(0)$ is considered as the initial learning rate [98] and $s(t) = 1$ if c_j and x belong to the same class and $s(t) = -1$ otherwise. Since $\eta_j(t)$ can also increase, for each $\eta_j(t)$ an upper bound $\eta_{max} \in (0, 1)$ is defined.

LVQ2.1: This variant of LVQ takes into consideration the two nearest prototypes c_{j^*} and $c_{j^{**}}$ to the presented training example in contrast to LVQ1 and OLVQ1 where the location of only the nearest prototype is adapted, where c_{j^*} is defined as in Eq.(2.25) and $c_{j^{**}}$ is defined as follows.

$$j^{**} = arg \min_{1 \leq j \leq k, j \neq j^*} d_j \tag{2.28}$$

The following three requirements must be fulfilled:

1. The two nearest prototypes c_{j^*} and $c_{j^{**}}$ belong to different classes.
2. The presented example x_μ belongs to the same class of c_{j^*} or $c_{j^{**}}$.
3. x_μ falls into a window of relative width w. The window is a zone of values defined around the midplane of d_{j^*} and $d_{j^{**}}$. The presented example x_μ falls into this window if

$$\min(\frac{d_{j^*}}{d_{j^{**}}}, \frac{d_{j^{**}}}{d_{j^*}}) > \frac{1-w}{1+w} \tag{2.29}$$

For $class(c_{j^*}) = class(x)$ and $class(c_{j^{**}}) \neq class(x_\mu)$, the learning rules are

$$c_{j^*}(t+1) = c_{j^*}(t) + \eta(t)(x_\mu - c_{j^*}(t)) \tag{2.30}$$
$$c_{j^{**}}(t+1) = c_{j^{**}}(t) - \eta(t)(x_\mu - c_{j^{**}}(t)) \tag{2.31}$$

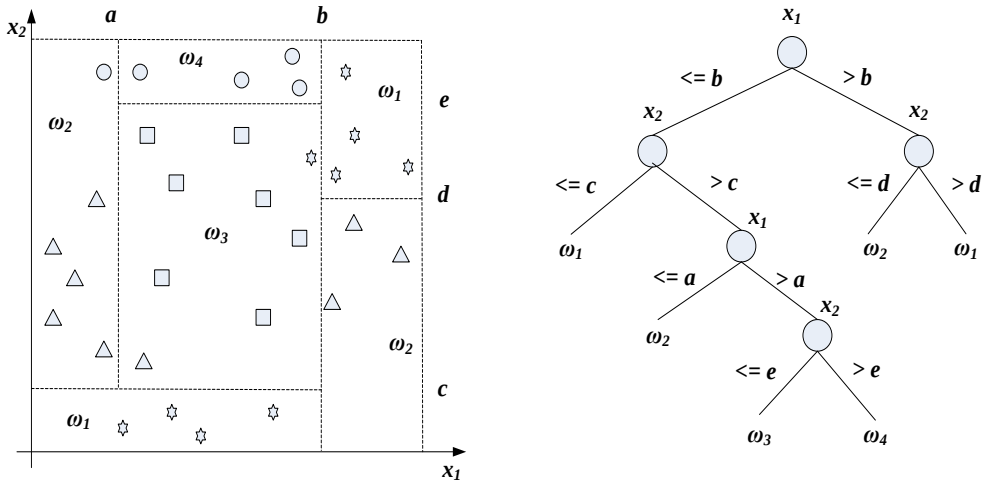
LVQ3: It is a modified version of LVQ2.1 where the locations of the two nearest prototypes c_{j^*} and $c_{j^{**}}$ are adapted. If both prototypes belong to the same class as the presented example x_μ, $class(c_{j^*}) = class(c_{j^{**}}) = class(x_\mu)$, then the learning rule is defined as follows

$$c_{j^*}(t+1) = c_{j^*}(t) + \epsilon\eta(t)(x - c_{j^*}(t)) \tag{2.32}$$
$$c_{j^{**}}(t+1) = c_{j^{**}}(t) + \epsilon\eta(t)(x - c_{j^{**}}(t)) \tag{2.33}$$

where $\epsilon \in [0, 1)$ is a scaling factor depending on the width of the window w. Note that if $class(c_{j^*}) \neq class(c_{j^{**}})$, LVQ3 algorithm is equivalent to LVQ2.1. After LVQ training phase, the resulting prototypes $c_1, \ldots, c_k$ can be used as the initial RBF centers [167]. Further details about the LVQ learning algorithms can be found in [98, 99].

2.1.4.3 Initialization with Decision Trees

Decision trees (or classification trees) [152] partition the input space $\mathbb{R}^D$ into disjoint axes-parallel hyper-rectangular regions $\Re_j$, see Section 2.3 for more details on classification trees. Two methods to determine the RBF centers based on k-means clustering and learning vector quantization (LVQ) are discussed in Section 2.1.4.1 and Section 2.1.4.2. In contrast to these methods, classification tree algorithm not only determine the RBF centers but also can adjust the RBF widths.



(a) feature space (b) binary decision tree

Figure 2.4: A binary decision tree of depth 4 constructed with two features, denoted by x_1 and x_2, is given (right panel). The data points belong to four different classes (denoted by ω_1, ω_2, ω_3 and ω_4) in a two dimensional feature space. The corresponding partitioning into hyper-rectangles parallel to the axes of the feature space is shown (left panel).

Kubat [104] was the first who suggested to initialize an RBF network with a decision tree then Schwenker and Dietrich addressed this topic in [165]. The set of J disjoint hyper-rectangular regions $\Re_j \subset \mathbb{R}^D$ produced by a decision tree can be transformed into a set of centers c_j and scaling parameters σ_j to initialize an RBF network. Hence, the number of leafs in the decision tree is the number of hidden RBF neurons in the network (this is an advantage because determining

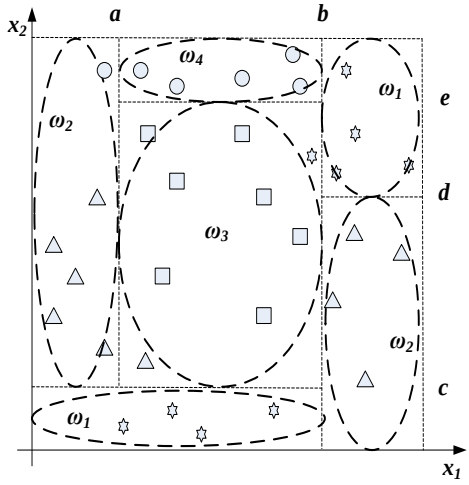


Figure 2.5: The regions in the feature space defined through the leaves of the decision tree. Centers of the RBFs are located in the middle of each hyper-rectangle and the contour of the RBFs are hyper-ellipsoids

the number of hidden nodes is a well-known problem in RBF networks design). In Figure 2.4, a decision tree and the set of regions defined through its leaves are shown. Each leaf of the decision tree defines a rectangular region in the feature space $\mathbb{R}^D$, here $D = 2$. For binary trees, each node is determined by a splitting condition consisting of a single feature x_i, $i \in \{1, \ldots, D\}$ and a boundary value $b_i \in \mathbb{R}$. Note that the data points of a single class are located in different regions of the input space, and thus one class can be represented by more than one leaf of the decision tree. For instance, class ω_1 is represented by two leaves. Each region $\Re_j$, represented by a leaf, is completely defined by a path through the tree starting at the root and terminating in a leaf.

For each region $\Re_j$, represented by a leaf of the decision tree, with

$$\Re_j = [a_{j1}, b_{j1}] \times \cdots \times [a_{jD}, b_{jD}] \tag{2.34}$$

Therefore, an RBF center $c_j = (c_{j1}, \ldots, c_{jD})$ can be defined through

$$c_{ji} = (a_{ji} + b_{ji})/2, \text{ for all } i = 1, \ldots, D \tag{2.35}$$

2.1.5 Determine RBF Widths

The setting of the radial basis function widths is an important factor that influences the classification (or approximation) performance of an RBF network [25]. It has a direct influence on the degree of smoothness of the function approximated by the network. If the kernel width $\sigma \in R$ is too large, the estimated probability

density is over-smoothed and the nature of the underlying true density may be lost. On the other hand, if σ is too small there may be an over-training of the given training data set. The smaller the widths are the less smoother are the realized functions. In general the Gaussian basis function ϕ_j is defined as

$$\phi_j(x) = e^{-\frac{1}{2}(x-c_j)^T \Sigma_j^{-1}(x-c_j)}, \text{ for all } j = 1, \ldots, J \tag{2.36}$$

where each Σ_j is a positive definite $D \times D$ matrix. Depending on the structure of the matrices Σ_j, four types of hyper-ellipsoids appear.

1. Σ_j are positive definite matrices. This implies that the axes of the hyper-ellipsoids are not necessary parallel to the axes of the feature space. (matrix-valued)

2. Σ_j are diagonal matrices: Here, the contour of a basis function ϕ_j is not radially symmetric. That is, the axes of the hyper-ellipsoids are parallel to the axes of the input space, but with different length, see Figure 2.5. In this case Σ_j is completely defined by a D-dimensional vector $\sigma_j \in \mathbb{R}^D$. (vector-valued)

3. $\Sigma_j = \sigma_j^2 Id$ where $\sigma_j^2 > 0$: Here the basis functions are radially symmetric, but are scaled with different widths. (real-valued)

4. $\Sigma_j = \sigma^2 Id$ where $\sigma^2 > 0$: In this case all basis functions ϕ_j have a radial symmetric contour all with constant width. This is the setting of RBF in the context of interpolation. (real-valued)

where Id is the identity matrix. The following is a list of different schemes used for the setting of the real-valued and vector-valued RBF widths in an RBF network. In all cases, a parameter $\alpha > 0$ has to be set heuristically.

1. All σ_j are set to the same value σ, which is proportional to the average of the p minimal distances between all pairs of prototypes.

2. The average of the distances between c_j and the p nearest prototypes of c_j is used to set the kernel width σ_j.

3. The kernel width σ_j is set proportional to the distance between c_j and the nearest prototype with a different class label.

$$\sigma_j = \alpha \min\{\|c_j - c_i\| : class(c_j) \neq class(c_i), i = 1, \ldots, J\} \tag{2.37}$$

4. The width $\sigma_j \in \mathbb{R}^D$ is set to the standard deviation of each feature calculated using the training examples belonging to cluster $\mathbb{C}_j$:

$$\sigma_{ji} = \alpha \sqrt{\frac{1}{|\mathbb{C}_j|} \sum_{x_\mu \in \mathbb{C}_j} (x_{\mu i} - c_{ji})^2} \tag{2.38}$$

5. In the case of using decision tree, the j^{th} RBF width is a diagonal matrix Σ_j, which is determined by a vector $\sigma_j \in \mathbb{R}^D$ as follows

$$\Sigma_j = diag(\sigma_{j1}^2, \ldots, \sigma_{jd}^2) \tag{2.39}$$

$$\sigma_{ji} = \frac{\alpha}{2}(b_{ji} - a_{ji}), \text{ for all } i = 1, \ldots, D \tag{2.40}$$

In general, the centers c_j and the scaling matrices Σ_j representing the location and the shape of radial basis functions can be determined using other techniques such as genetic algorithms (GAs) [200, 79], or expectation-maximization (EM) [156].

2.1.6 Calculate the Output Layer Weights

After that the RBF centers c_j and the scaling parameters, given by the matrices Σ_j, have been determined, the weights of the output layer can consequently be calculated. It is assumed that the hidden layer of the RBF network has k basis functions. Let $L = \{(x_\mu, y_\mu) : x_\mu \in \mathbb{R}^D, y_\mu \in \Omega, \mu = 1, \ldots, M\}$ be the training set, $\Phi_{\mu j} = \phi_j(x_\mu)$ the output of the j^{th} RBF with the μ^{th} feature vector as input and using the *1-of-K* encoding scheme, the class label of each training example $y_\mu \in \Omega$ is encoded into an K-dimensional binary vector $t_\mu \in \{0, 1\}^K$ through the relation $t_{\mu k} = 1$ iff $y_\mu = \omega_k$. Given the two matrices $\Phi = (\Phi_{\mu j})$ and $T = (t_{\mu k})$, the matrix of the output layer weights W is the result minimizing of the error function:

$$E(W) = \|\Phi W - T\|^2. \tag{2.41}$$

2.1.6.1 Error Back Propagation

The solution can be found by gradient descent optimization of the error function defined in Eq. (2.41). Each training example in the training set is presented to the input layer of the network and the predicted outputs are calculated. The difference between each predicted output and the corresponding target output is calculated. This error is then propagated back through the network and the weights on the arcs of the networks are adjusted so that if the training example is presented to the network again, then the error would be less. The learning algorithm typically iterates through the training set L many times where each iteration is called an epoch in the neural network literature. This leads to the delta learning rule for the output layer weights

$$w_{jk} = w_{jk} - \eta \sum_{\mu=1}^{M} \phi_j(x_\mu)(t_{\mu k} - f_k(x_\mu)), \tag{2.42}$$

or its incremental version

$$w_{jk} = w_{jk} - \eta \phi_j(x_\mu)(t_{\mu k} - f_k(x_\mu)), \tag{2.43}$$

where $\eta > 0$ is the learning rate. After this step of calculating the output layer weights, all parameters of the RBF network have been determined. This learning scheme is efficient and provides good classification results. The main shortcoming of gradient descent error back propagation algorithm is that it is slow.

2.1.6.2 Pseudo-Inverse Solution

A least squares solution can be found directly to the system of linear equations $W = \Phi^+ T$ where Φ^+ is the pseudo-inverse of the activation matrix Φ which can be defined as

$$\Phi^+ = \lim_{\alpha \to 0} (\Phi^T \Phi + \alpha Id)^{-1} \Phi^T \tag{2.44}$$

where Id is the identity matrix. If the pseudo inverse of $(\Phi^T \Phi)$ is already known, then simply $\Phi^+ = (\Phi^T \Phi)^{-1} \Phi^T$. This direct computation is faster than the gradient descent optimization and yields good classification results.

2.2 k-Nearest Neighbors Algorithms

The nearest neighbor algorithm has been widely used for decades to construct an effective classification model [68, 48]. It is based on a distance function that measures the difference or similarity between instances in the feature space. It is called lazy learning algorithm because it requires neither extensive training nor the adjustment of parameters. The classifier is trained by storing all the M training examples into memory. Let $\Omega = \{\omega_1, \ldots, \omega_C\}$ be the set of classes.

2.2.1 k-Nearest Neighbors Classifier

The calculation of nearest neighbors involves two steps: calculating and sorting the distances. Given an unseen example x, the distances between x and the stored training examples $x_\mu, \mu = 1, \ldots, M$, are calculated

$$d_\mu = \|x - x_\mu\|_p = \left(\sum_{i=1}^{D} |x_i - x_{\mu i}|^p \right)^{1/p} \tag{2.45}$$

where $\|.\|_p$ denotes the Minkowski distance between two D-dimensional feature vectors x and x_μ, as defined in Eq. (2.45) where $p \in [1, \infty)$ is the distance order. Then the distances are sorted in ascending order $d_{v(1)} \leq \cdots \leq d_{v(M)}$. Finally, the set of the k closest neighbors, $N_k(x)$, is defined as

$$N_k(x) = \{x_{v(1)}, \ldots, x_{v(k)}\} \tag{2.46}$$

The nearest neighbors that belong to class ω_c is defined as

$$N_k^c(x) = \{x_\mu \in N_k(x) | y_\mu = \omega_c\}, \text{ for } c = 1, \ldots, C \tag{2.47}$$

Majority voting is conducted to assign the most frequent class $\hat{y}$ to x.

$$\hat{y} = arg \max_{\omega_c \in \Omega} |N_k^c(x)| \tag{2.48}$$

2.2.2 Fuzzy k-Nearest Neighbors Classifier

The k-nearest neighbors classifier can be considered as a fuzzy classifier. It can provide an estimate for the degree of membership of an example x to each class in Ω where the distances between x and its k closest neighbors are incorporated in the class memberships calculation [194]. The fuzzy class membership degree f is represented by a mapping $f : \mathbb{R}^D \rightarrow [0,1]^C$ as

$$f_c(x) = \frac{\sum_{x_j \in N_k^c(x)} \phi_j(x)}{\sum_{c'=1}^{C} \sum_{x_j \in N_k^{c'}(x)} \phi_j(x)} \tag{2.49}$$

and $\phi_j(x)$ can take one of the following forms:

- The inverse of the distance

$$\phi_j(x) = \frac{1}{\|x - x_j\|_p + \epsilon} \tag{2.50}$$

  where $\epsilon > 0$ is a constant added to avoid zero denominator.

- The decreasing exponential function

$$\phi_j(x) = exp(-\frac{\|x - x_j\|_p}{2\sigma^2}) \tag{2.51}$$

  when $p = 2$ then it is the Gaussian function widely used in RBF networks and $\sigma > 0$ is the width parameter

An unseen example x is assigned to the class $\hat{y}$ with the highest membership

$$\hat{y} = arg \max_{1 \leq c \leq C} f_c(x) \tag{2.52}$$

2.2.3 Nearest Prototype Classifier

The nearest-neighbor algorithm has large storage requirements because it requires that all the training examples be loaded in the memory in the classification phase. In addition, the classification time is computationally expensive because it requires calculating and sorting the distances between the example to be classified and all the training examples. In order to significantly reduce the computational load, a clustering algorithm such as k-means clustering defined in Section 2.1.4.1 or LVQ defined in Section 2.1.4.2 can be applied in the training phase of the

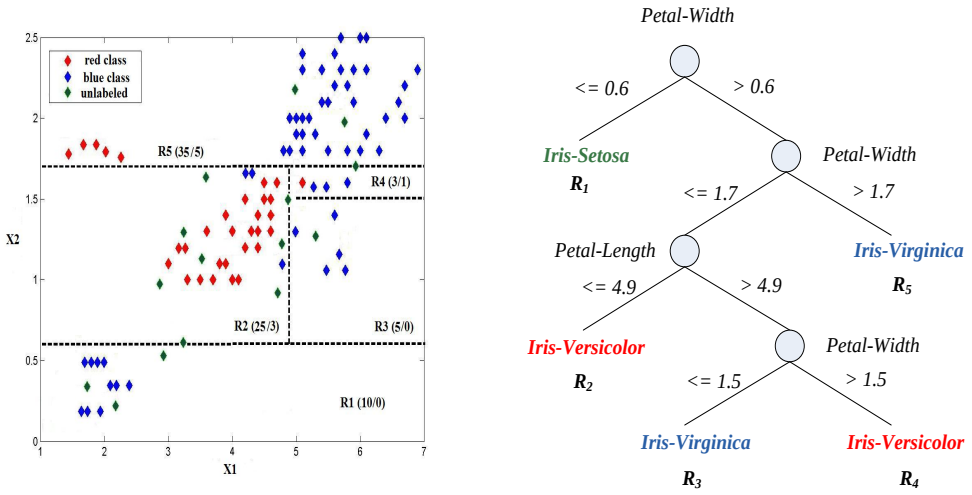
nearest-neighbor classifier to determine a smaller set of $J << M$ prototypes from the M training examples. Reducing the complexity is important for handling the large datasets that are available in many real-world applications in medicine, biology, finance, etc. Finding a smaller set of prototypes can also get grid of noisy training examples and therefore can improve the classification accuracy. For the *Nearest Prototype Classifier* [68, 178, 108], only the small set $J << M$ prototypes is stored in the memory at the classification phase and the distances are calculated only between the example to be classified and the J prototypes. Many authors also consider the distances between x and the k nearest prototype vectors to calculate the class membership estimates.

2.3 Decision Trees

A decision tree is a tree structure classifier that consists of internal nodes, leaf nodes, and arcs. Decision trees (or classification trees) [151, 152] partition the input space $\mathbb{R}^D$ into disjoint axes-parallel hyper-rectangular regions $\Re_j$. The binary decision tree is the most popular type where each node has either zero or two children. Each node in a decision tree represents a certain region $\Re$ of $\mathbb{R}^D$. If the node is a terminal node, called a leaf, all data points within this region $\Re$ are assigned to the same class. If a node has two children then the two regions represented by the children nodes, denoted by $\Re_{left}$ and $\Re_{right}$ form a partitioning of $\Re$, i.e. $\Re_{left} \cup \Re_{right} = \Re$ and $\Re_{left} \cap \Re_{right} = \emptyset$ (see Figure 2.6). Decision trees are popular especially in ensemble learning because of their computationally inexpensive training time (compared to neural network classifiers) and classification time (compared to k-nearest neighbors classifiers).

C4.5 [152] is an improvement of the ID3 [151] algorithm that takes into account missing values, continuous features, and tree pruning. It builds decision trees in a top-down fashion and prunes them. At each node, all the features are evaluated based on the current training data and using some evaluation function. The feature with the highest score is selected, and the training set is split into two partitions based on a certain value of the selected feature. The process is repeated recursively for each data partition. The splitting procedure stops when one of the following conditions is fulfilled: (1) all the training examples within the current node belong to the same class, (2) the number of the training examples is less than a user defined threshold, or (3) the evaluation criterion indicates that there is no splitting can lead to further improvement.

Another type of decision trees, called *oblique decision trees*, use hyperplanes that are not necessary parallel to any axis of the feature space. A generalization is to use hyperplanes in a transformed space, where each new feature is an arbitrary function of a selected subset of the original features. The speed of execution depends on the transformation function and the complexity of the hyperplanes.



(a) reduced feature space (b) binary decision tree

Figure 2.6: A binary decision tree of depth 4 constructed for the iris data set with only two features of the four given features, denoted by *petal-length* and *petal-width* (right panel). The data points belong to three different classes (denoted by *iris-setosa*, *iris-virginica*, and *iris-versicolor*). Each node is labeled with the selected feature and a boundary value. The corresponding hyper-rectangles parallel to the axes is shown (left panel), where boundary values and class labels are shown. The minimum and maximum values of each feature within the training set are additional boundary values. Hence, all regions are bounded.

2.3.1 Evaluation Criteria

Most of the evaluation functions used for feature selection, depend on minimizing the impurity of the data. That is, it selects features that construct regions in the feature space where the examples from one class is significantly greater than the examples belonging to other classes, optimally having all the examples from the same class. CART [33] uses the Gini index,

$$Gini\_index(Y) = 1 - \sum_{\omega_k \in \Omega} p(Y = \omega_k)^2 = 1 - \sum_{\omega_k \in \Omega} (\frac{n_k}{n})^2 \tag{2.53}$$

while ID3, C4.5 and C5 depend on the entropy of data (information gain and gain ratio) in feature selection. Suppose that $\Omega = \{\omega_1, \ldots, \omega_K\}$ is the set of classes in a given data set L of n examples. The entropy of a random variable Y, denoted by $H(Y)$, that represents the amount of information needed to classify L is defined as,

$$H(Y) = -\sum_{\omega_k \in \Omega} p(Y = \omega_k) \log_2 p(Y = \omega_k) = -\sum_{\omega_k \in \Omega} \frac{n_k}{n} \log_2 \frac{n_k}{n} \tag{2.54}$$

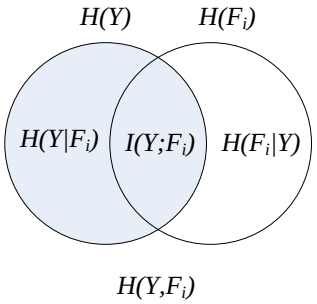


Figure 2.7: Individual $(H(Y), H(F_i))$, joint $(H(Y, F_i))$, and conditional entropies for the random variables that represent the class label Y and the input feature F_i with mutual information $I(Y; F_i)$.

where n_k is the number of examples belonging to class ω_k and the base of the logarithm has a common value 2 since the information is encoded in bits. Suppose that the examples in L are represented by D features. The aim of the induction algorithm is to select the most relevant feature of the D features to partition L. Let F_i be the i^{th} feature that has m possible values $\{a_1, \ldots, a_m\}$. Then the conditional entropy of Y given F_i, denoted by $H(Y|F_i)$, defines the amount of information remaining in Y if the value F_i is known and is given as,

$$\begin{aligned} H(Y|F_i) &= -\sum_{j=1}^{m} p(F_i = a_j) \sum_{\omega_k \in \Omega} p(Y = \omega_k | F_i = a_j) \log_2 p(Y = \omega_k | F_i = a_j) \\ &= -\sum_{j=1}^{m} \frac{s_j}{n} \sum_{\omega_k \in \Omega} \frac{p(Y = \omega_k, F_i = a_j)}{p(F_i = a_j)} \log_2 \frac{p(Y = \omega_k, F_i = a_j)}{p(F_i = a_j)} \\ &= -\sum_{j=1}^{m} \frac{s_j}{n} \sum_{\omega_k \in \Omega} \frac{s_{kj}}{s_j} \log_2 \frac{s_{kj}}{s_j} \end{aligned} \tag{2.55}$$

where s_j is the number of examples having the value a_j for feature F_i and s_{kj} is the number of examples belonging to class ω_k and having the value a_j for feature F_i. The mutual information between Y and F_i, that measures the difference in the impurity before the splitting and after the splitting using feature F_i, see Figure 2.7, is defined as

$$Gain(F_i) = I(Y; F_i) = H(Y) - H(Y|F_i) \tag{2.56}$$

One shortcoming of Gain is that it gives higher preference to the features with larger number of values. Quinlan proposed to use *Gain Ratio* instead of *Gain* in order to compensate this shortage.

$$GainRatio(F_i) = \frac{Gain(F_i)}{splitInfo(F_i)} = \frac{I(Y; F_i)}{H(F_i)} \tag{2.57}$$

where $H(F_i)$ is the entropy of F_i and represents the amount of information provided by feature F_i. The splitting information of some features may be small and lead to unstable gain ratio. To avoid this, the most relevant feature, F_{i^*}, is selected as follows

$$i^* = arg \max_{1 \leq i \leq D} GainRatio(F_i) \tag{2.58}$$

subject to the constraint that

$$Gain(F_{i^*}) > avgGain \text{ and } avgGain = \frac{1}{D} \sum_{i=1}^{D} Gain(F_i). \tag{2.59}$$

Then the training data L is split based on this feature.

2.3.2 Pruning

After the tree is constructed, a pruning step is performed in order to avoid overfitting. There are three common approaches for pruning classification trees. The first approach is based on a separate validation set where the tree is pruned to minimize the validation error. Both reduced error pruning (REP) used in C4.5 and cost-complexity pruning (CCP) used in CART depend on this approach. The second approach is based on information-theoretic functions to seek the tree with minimal complexity, such as the minimum description length (MDL) criterion [157]. The third approach depend on a probabilistic estimate of the error that is based on the frequency of examples in each node, such as pessimistic error pruning (PEP) [152] used in C4.5. Unlike REP, in PEP the same data set is used for both growing and pruning the tree.

2.3.3 Classification Phase

To classify an unseen example, the decision tree will evaluate the test specified by the root node and follow the branch corresponding to the evaluation result. For instance, an example with *petal-width* ≤ 0.6 will follow the left branch in the tree from Figure 2.6 while it follows the right branch otherwise. Then it will follow the left branch if *petal-width* ≤ 1.7. Unless the example reaches a leaf node, it will traverse the tree through the branches according to the evaluation output of each node. When it reaches a leaf node, the example will be assigned the class label that has the maximum number of training examples associated with this leaf. For instance, any example reaches region $R1$ will be assigned to class *iris-setosa* because this region is occupied by 50 training examples from class *iris-setosa* and 0 training examples from the other classes.

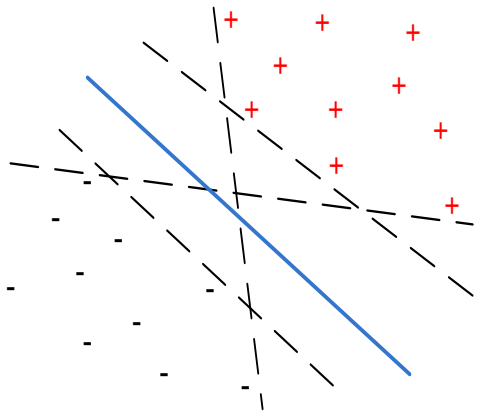


Figure 2.8: The optimal separating hyperplane (solid line) and four non-optimal separating hyperplanes (dashed lines)

2.4 Support Vector Machines

Support Vector Machines (SVM) are learning algorithms based on the statistical learning theory [193]. They were originally used for classifying linearly separable data. The simplest binary SVM constructs an optimal hyperplane that defines a decision boundary to separate a set of positive examples from a set of negative examples, which can work well for unseen examples. There are many possible separating hyperplanes but there is only one which maximizes the margin [193, 13]. The margin is the minimal distance between the separating hyperplane and the nearest training example to both classes, see Figure 2.8.

2.4.1 Hard-Margin Support Vector Machines

Let $L = \{(x_\mu, y_\mu) : x_\mu \in \mathbb{R}^D, y_\mu \in \{-1, 1\}, \mu = 1, \ldots, M\}$ be the set of training examples belonging to class ω_h or class ω_k where the associated label is $y_\mu = -1$ for ω_h and $y_\mu = 1$ for ω_k. If the training data are linearly separable, the decision function can be written as:

$$f(x) = \langle w, x \rangle - b, \tag{2.60}$$

where w is a D-dimensional vector orthogonal to the hyperplane, $b \in \mathbb{R}$ is a bias term, and the following constraints hold for $\mu = 1, \ldots, M$, (see Figure 2.9),

$$f(x_\mu) \begin{cases} > 0 & \text{for } y_\mu = 1, \\ < 0 & \text{for } y_\mu = -1 \end{cases} \tag{2.61}$$

Thus, to control separability the following constraints are used instead of Eq.(2.61).

$$f(x_\mu) \begin{cases} \geq 1 & \text{for } y_\mu = 1, \\ \leq -1 & \text{for } y_\mu = -1 \end{cases} \tag{2.62}$$

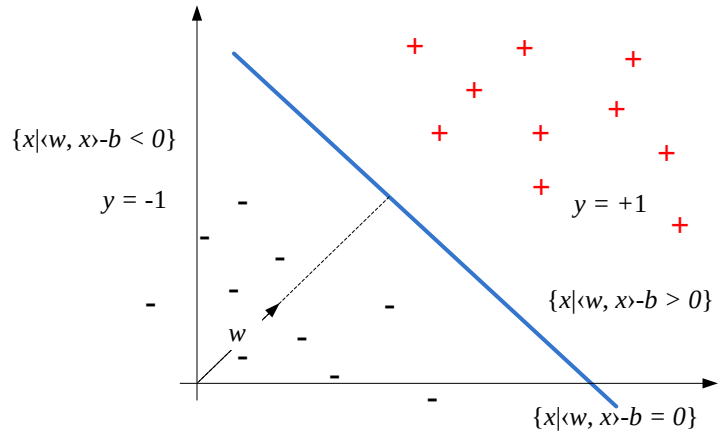


Figure 2.9: The separating hyperplane written in terms of orthogonal weight w and bias b

Note that 1 and -1 can be any constant $a(> 0)$ and -a, respectively. But by dividing both sides of the inequalities by a, Eq. (2.62) is obtained which is equivalent to

$$y_\mu(\langle w, x_\mu\rangle - b) \geq 1, \text{ for } \mu = 1\ldots, M \tag{2.63}$$

The hyperplane

$$f(x) = \langle w, x\rangle - b = c \text{ for } -1 \leq c \leq 1 \tag{2.64}$$

forms a separating hyperplane that separates the training examples in L. The separating hyperplane $f(x) = 0$ lies in the middle between the two hyperplanes $f(x) = -1$ and $f(x) = 1$. The distance between the hyperplane and the nearest training example is called the *margin*. Figure 2.8 demonstrates five decision functions that satisfy the constraints in Eq. (2.63). There are an infinite number of decision functions that satisfy Eq. (2.63). The generalization ability of the separating hyperplane depends on its location, and the hyperplane with the maximum margin is called *the optimal separating hyperplane* (see Figure 2.10). The Euclidean distance from any training example x_μ to the separating hyperplane $f(x) = 0$ must satisfy

$$\frac{y_\mu(\langle w, x_\mu\rangle - b)}{\|w\|} \geq \frac{1}{\|w\|} \tag{2.65}$$

Therefore, in order to maximize the margin, the norm of w must be minimized. That is, the optimal separating hyperplane can be obtained by solving the following problem:

$$\min_{w,b} \Psi_P(w, b) = \frac{1}{2}\|w\|^2, \tag{2.66}$$

with respect to w and b subject to the constraints:

$$y_\mu(\langle w, x_\mu\rangle - b) \geq 1, \text{ for } \mu = 1\ldots, M \tag{2.67}$$

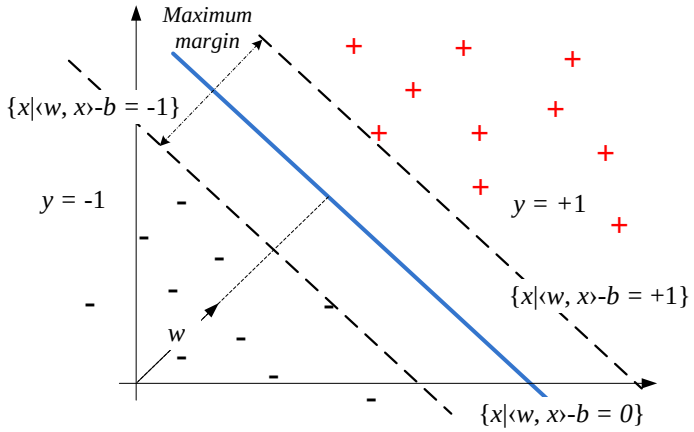


Figure 2.10: The optimal separating hyperplane for a linearly separable data set. The larger the margin, the better is the generalization ability of the classifier

This quadratic problem is a convex optimization problem [13] and is called *the primal formulation*. The training examples that satisfy the equalities in Eq. (2.67) are called *support vectors*. Since the unknown variables of the convex optimization problem are w and b, the number of variables to be obtained is the number of input features plus one, $D + 1$. When the number of input features is small, we can solve Eq. (2.66) and Eq. (2.67) by the quadratic programming techniques. But, as will be discussed later, because we map the input space into a high-dimensional feature space, in some cases, with infinite dimensions, the problem defined in Eq. (2.66) and Eq. (2.67) is converted into its equivalent dual problem, using standard Lagrangian techniques, whose number of variables is the number of training examples.

First, the constrained problem defined in Eq. (2.66) and Eq. (2.67) is converted into the unconstrained problem

$$L_P(w, b) = \frac{1}{2} \|w\|^2 - \sum_{\mu=1}^{M} \alpha_\mu [y_\mu (\langle w, x_\mu \rangle - b) - 1] \tag{2.68}$$

where $\alpha = (\alpha_1, \ldots, \alpha_M)^T$ and α_μ are the nonnegative Lagrange multipliers. The

optimal solution satisfies the following Karush-Kuhn-Tucker (KKT) conditions

$$\frac{\partial L_P}{\partial w} = 0 \Rightarrow w = \sum_{\mu=1}^{M} \alpha_\mu y_\mu x_\mu; \tag{2.69}$$

$$\frac{\partial L_P}{\partial b} = 0 \Rightarrow \sum_{\mu=1}^{M} \alpha_\mu y_\mu = 0; \tag{2.70}$$

$$\alpha_\mu[y_\mu(\langle w, x_\mu\rangle - b) - 1] = 0 \text{ for } \mu = 1, \ldots M, \tag{2.71}$$

$$\alpha_\mu \geq 0 \text{ for } \mu = 1, \ldots M. \tag{2.72}$$

The relations between the inequality constraints and their associated Lagrange multipliers given by Eq. (2.71) are called *KKT complementarity conditions*. Substituting Eq. (2.69) and Eq. (2.70) into Eq. (2.68), we obtain the dual formulation of the problem:

$$\max_{\alpha} \Psi_D(\alpha) = \sum_{\mu=1}^{M} \alpha_\mu - \frac{1}{2} \sum_{\mu,j=1}^{M} \alpha_\mu \alpha_j y_\mu y_j \langle x_\mu, x_j\rangle \tag{2.73}$$

subject to the constraints

$$\sum_{\mu=1}^{M} \alpha_\mu y_\mu = 0 \quad \text{and} \quad \alpha_\mu \geq 0 \text{ for } \mu = 1, \ldots M. \tag{2.74}$$

If the classification problem is linearly separable, the global optimal solution $\alpha_\mu(\mu = 1, \ldots, M)$ exists [13]. This is one of the advantages of support vector machines over neural networks, which have numerous local minimal solutions. For quadratic programming, the values of the primal and dual objective functions, Ψ_P and Ψ_D, coincide at the optimal solutions if they exist, which is called the zero duality gap. Then from Eq. (2.69) the decision function is defined as

$$f(x) = \sum_{\mu \in S} \alpha_\mu y_\mu \langle x_\mu, x\rangle - b \tag{2.75}$$

where S is the set of support vector indices, and from the *KKT complementarity conditions* defined in Eq. (2.71), if $\alpha_j \neq 0$, then

$$b = \langle w, x_j\rangle - y_j \tag{2.76}$$

For more precise calculation, take the average over all support vectors (x_j, y_j) as follows:

$$b = \frac{1}{|S|} \sum_{j \in S} [\langle w, x_j\rangle - y_j]. \tag{2.77}$$

Therefore, an unknown example x is classified into:

$$\begin{cases} \text{Class 1 (assigned to } \omega_h) & \text{if } f(x) > 0, \\ \text{Class 2 (assigned to } \omega_k) & \text{if } f(x) < 0. \end{cases} \tag{2.78}$$

If $f(x) = 0$, x is on the boundary and thus is unclassifiable. When training examples are linearly separable, the region $\{x|-1 < f(x) < 1\}$ is a generalization region.

2.4.2 Soft-Margin Support Vector Machines

In hard-margin SVMs, it is assumed that the training examples are linearly separable. When the data are not linearly separable, there is no feasible solution, and the hard-margin SVM is unsolvable [13]. Cortes and Vapnik [44] extend SVMs to the case of inseparability, the nonnegative slack variables $\epsilon_\mu (\geq 0)$ are introduced into the inequality constraint:

$$y_\mu(\langle w, x_\mu \rangle - b) \geq 1 - \epsilon_\mu, \text{ for } \mu = 1 \ldots, M \tag{2.79}$$

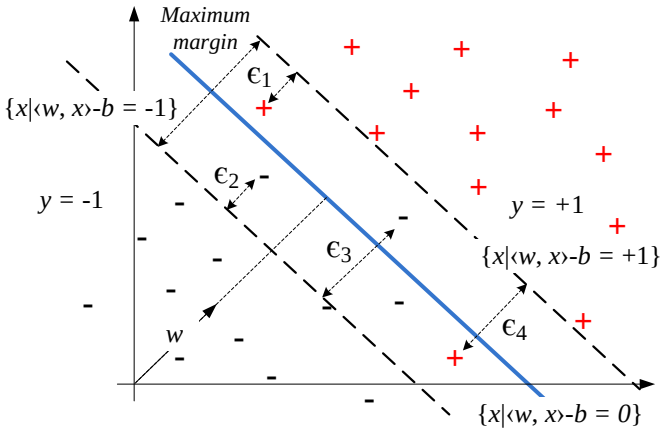


Figure 2.11: The optimal separating hyperplane for a nonlinearly separable data set in a two-dimensional space

By the slack variables ϵ_μ, feasible solutions always exist. For the training examples x_μ, if $0 < \epsilon_\mu < 1$ (such as ϵ_1 and ϵ_2 in Figure 2.11), the examples do not have the maximum margin but are still correctly classified. But if $\epsilon_\mu \geq 1$ (such as ϵ_3 and ϵ_4 in Figure 2.11) the examples are misclassified by the optimal hyperplane. To obtain the optimal hyperplane that minimize the number of training example that do not have the maximum margin, we solve the following problem:

$$\min_{w,b} \Psi_P(w, b) = \frac{1}{2} \|w\|^2 + C \sum_{\mu=1}^{M} \epsilon_\mu, \tag{2.80}$$

with respect to w and b subject to the constraints:

$$y_\mu(\langle w, x_\mu\rangle - b) \geq 1 - \epsilon_\mu, \quad \epsilon_\mu \geq 0 \quad \text{for } \mu = 1 \ldots, M \tag{2.81}$$

where $\epsilon = (\epsilon_1, \ldots, \epsilon_M)^T$ and C is a regularization term that controls the trade-off between maximizing the margin and training error minimization.

Similar to the linearly separable case, using standard Lagrangian techniques, the following unconstrained problem is obtained

$$L_P(w, b) = \frac{1}{2}\|w\|^2 + C\sum_{\mu=1}^{M}\epsilon_\mu - \sum_{\mu=1}^{M}\alpha_\mu[y_\mu(\langle w, x_\mu\rangle - b) - 1 + \epsilon_\mu] - \sum_{\mu=1}^{M}\beta_\mu\epsilon_\mu \tag{2.82}$$

where $\alpha = (\alpha_1, \ldots, \alpha_M)^T$ and $\beta = (\beta_1, \ldots, \beta_M)^T$ represent the nonnegative Lagrange multipliers. The optimal solution satisfies the following KKT conditions

$$\frac{\partial L_P}{\partial w} = 0 \Rightarrow w = \sum_{\mu=1}^{M}\alpha_\mu y_\mu x_\mu; \tag{2.83}$$

$$\frac{\partial L_P}{\partial b} = 0 \Rightarrow \sum_{\mu=1}^{M}\alpha_\mu y_\mu = 0; \tag{2.84}$$

$$\frac{\partial L_P}{\partial \epsilon_\mu} = 0 \Rightarrow \alpha_\mu + \beta_\mu = C \text{ for } \mu = 1, \ldots M \tag{2.85}$$

$$\alpha_\mu[y_\mu(\langle w, x_\mu\rangle - b) - 1 + \epsilon_\mu] = 0 \text{ for } \mu = 1, \ldots M, \tag{2.86}$$

$$\beta_\mu\epsilon_\mu = 0 \text{ for } \mu = 1, \ldots M, \tag{2.87}$$

$$\alpha_\mu \geq 0, \;\; \beta_\mu \geq 0, \;\; \epsilon_\mu \geq 0 \text{ for } \mu = 1, \ldots M. \tag{2.88}$$

Thus substituting Eq. (2.83) to Eq. (2.87) into Eq. (2.82), we obtain the following dual problem.

$$\max_\alpha \Psi_D(\alpha) = \sum_{\mu=1}^{M}\alpha_\mu - \frac{1}{2}\sum_{\mu,j=1}^{M}\alpha_\mu\alpha_j y_\mu y_j\langle x_\mu, x_j\rangle \tag{2.89}$$

subject to the constraints

$$\sum_{\mu=1}^{M}\alpha_\mu y_\mu = 0 \quad \text{and} \;\; C \geq \alpha_\mu \geq 0 \text{ for } \mu = 1, \ldots M. \tag{2.90}$$

The only difference between soft-margin SVMs and hard-margin SVMs is that α_μ cannot exceed C.

From Eq. (2.85) and the *KKT complementarity conditions* defined in Eq. (2.86) and Eq. (2.87), there are three cases for α_μ:

- If $\alpha_\mu = 0$, then $\epsilon_\mu = 0$ and $y_\mu(\langle w, x_\mu \rangle - b) \geq 1$. Thus x_μ is correctly classified and it is not a support vector.
- If $0 < \alpha_\mu < C$, then $\epsilon_\mu = 0$ and $y_\mu(\langle w, x_\mu \rangle - b) = 1$. Thus, x_μ is a support vector. The support vectors with $0 < \alpha_\mu < C$ are called unbounded support vectors.
- If $\alpha_\mu = C$, then $y_\mu(\langle w, x_\mu \rangle - b) - 1 + \epsilon_\mu = 0$ and $\epsilon_\mu \geq 0$. Thus x_μ is a support vector. The support vectors with $\alpha_\mu = C$ are called bounded support vectors. If $0 \leq \epsilon_\mu < 1$, x_μ is correctly classified, and if $\epsilon_\mu \geq 1$, x_μ is misclassified.

The decision function is the same as that of the hard-margin SVM and is given by

$$f(x) = \sum_{\mu \in S} \alpha_\mu y_\mu \langle x_\mu, x \rangle - b \tag{2.91}$$

where S is the set of support vector indices. For $0 < \alpha_j < C$,

$$b = \langle w, x_j \rangle - y_j \tag{2.92}$$

For more precise calculation, take the average over all unbounded support vectors

$$b = \frac{1}{|U|} \sum_{j \in U} [\langle w, x_\mu \rangle - y_j]. \tag{2.93}$$

where U is the set of unbounded support vector indices. Therefore, an unknown example x is classified into:

$$\begin{cases} \text{Class 1 (assigned to } \omega_h) & \text{if } f(x) > 0, \\ \text{Class 2 (assigned to } \omega_k) & \text{if } f(x) < 0. \end{cases} \tag{2.94}$$

If $f(x) = 0$, x is on the boundary and thus is unclassifiable. When there are no bounded support vectors, the region $\{x| - 1 < f(x) < 1\}$ is a generalization region.

2.4.3 Nonlinear Mapping to a High-Dimensional Space

2.4.3.1 Kernel Trick

The objective of finding the optimal hyperplane is to maximize the generalization ability of the SVM classifier. But if the training examples are not linearly separable, the obtained classifier may not have high generalization ability although the hyperplanes are determined optimally. Thus to enhance linear separability, the original input space is mapped into a high-dimensional dot-product space, that is called the *feature space* in order to distinguish it from the input space [40], where

it is possible to construct an optimal separating hyperplane with better generalization ability. This transformation is justified by Cover's Theorem in [164]. It is a nonlinear function $\phi(x) = (\phi_1(x), \ldots, \phi_L(x))^T$ that maps the D-dimensional input vector x into the L-dimensional feature space. Thus, the linear decision function in the feature space is given by

$$f(x) = \langle w, \phi(x) \rangle - b, \tag{2.95}$$

where w is the L-dimensional weight vector and b is the bias term. The formulation of the new optimization problem can be done by replacing all occurrences of x with $\phi(x)$. This leads to the following dual problem.

$$\max_{\alpha} \Psi_D(\alpha) = \sum_{\mu=1}^{M} \alpha_\mu - \frac{1}{2} \sum_{\mu,j=1}^{M} \alpha_\mu \alpha_j y_\mu y_j \langle \phi(x_\mu), \phi(x_j) \rangle \tag{2.96}$$

To solve this optimization problem, there is a need to explicitly map the training examples into the higher-dimensional space and then to compute the dot products $\langle \phi(x_\mu), \phi(x_j) \rangle$, this is computationally expensive.

According to the Hilbert-Schmidt theory, if a symmetric function $\mathbb{K}(x_i, x_j)$ satisfies

$$\sum_{i,j=1}^{M} h_i h_j \mathbb{K}(x_i, x_j) \geq 0 \tag{2.97}$$

for any $M \in \mathbb{N}, x_i \in \mathbb{R}^D, h_i \in \mathbb{R}$, there exists a mapping function ϕ that maps x into the dot-product feature space that satisfies

$$\mathbb{K}(x_i, x_j) = \langle \phi(x_i), \phi(x_j) \rangle. \tag{2.98}$$

Substituting Eq. (2.98) into Eq. (2.97),

$$\left(\sum_{i=1}^{M} h_i \phi(x_i) \right) \left(\sum_{j=1}^{M} h_j \phi(x_j) \right) \geq 0 \tag{2.99}$$

The condition in Eq. (2.97) or Eq. (2.99) is called Mercer condition, and the function that satisfies it is called the positive semidefinite kernel or the Mercer kernel.

The advantage of using kernels is that we need not treat the high-dimensional feature space explicitly. This technique is called *kernel trick*. That is, we use kernel function $K(x_i, x_j)$ in training and classification instead of $\phi(x_i)$ and $\phi(x_j)$.

The dual problem in the feature space is defined as follows:

$$\max_{\alpha} \Psi_D(\alpha) = \sum_{\mu=1}^{M} \alpha_\mu - \frac{1}{2} \sum_{\mu,j=1}^{M} \alpha_\mu \alpha_j y_\mu y_j \mathbb{K}(x_\mu, x_j) \tag{2.100}$$

subject to the constraints

$$\sum_{\mu=1}^{M} \alpha_\mu y_\mu = 0 \quad \text{and} \quad C \geq \alpha_\mu \geq 0 \text{ for } \mu = 1, \ldots M. \tag{2.101}$$

The KKT complementarity conditions are given by

$$\alpha_\mu \left(y_\mu \left(\sum_{j=1}^{M} y_\mu \alpha_\mu \mathbb{K}(x_\mu, x_j) - b \right) - 1 + \epsilon_\mu \right) = 0 \text{ for } \mu = 1, \ldots M, \tag{2.102}$$

$$(C - \alpha_\mu)\epsilon_\mu = 0 \text{ for } \mu = 1, \ldots M, \tag{2.103}$$

$$\alpha_\mu \geq 0, \quad \epsilon_\mu \geq 0 \text{ for } \mu = 1, \ldots M. \tag{2.104}$$

The decision function is the same as that of the hard-margin SVM and is given by

$$f(x) = \sum_{\mu \in S} \alpha_\mu y_\mu \mathbb{K}(x_\mu, x) - b \tag{2.105}$$

where b is defined as by an unbounded support vector x_j

$$b = \sum_{\mu \in S} \alpha_\mu y_\mu \mathbb{K}(x_\mu, x_j) - y_j \tag{2.106}$$

For more stable calculation, take the average over the unbounded support vectors

$$b = \frac{1}{|U|} \sum_{j \in U} [\sum_{\mu \in S} \alpha_\mu y_\mu \mathbb{K}(x_\mu, x_j) - y_j]. \tag{2.107}$$

where U is the set of unbounded support vector indices. Therefore, an unknown example x is classified into:

$$\begin{cases} \text{Class 1 (assigned to } \omega_h) & \text{if } f(x) > 0, \\ \text{Class 2 (assigned to } \omega_k) & \text{if } f(x) < 0. \end{cases} \tag{2.108}$$

If $f(x) = 0$, x is on the boundary and thus is unclassifiable.

2.4.3.2 Kernels

- Linear Kernels: If the training examples are linearly separable in the input space, there is no need to map the input space to a higher-dimensional space. In this case, linear kernel is used

$$\mathbb{K}(x_\mu, x_j) = \langle x_\mu, x_j \rangle \tag{2.109}$$

- Polynomial Kernels: A polynomial kernel with degree $d \in \mathbb{N}$ is

$$\mathbb{K}(x_\mu, x_j) = (\langle x_\mu, x_j \rangle + 1)^d \tag{2.110}$$

When $d = 1$, the kernel is the linear kernel plus 1. If $d = 2$ and $D = 2$, the polynomial kernel becomes,

$$\begin{aligned} \mathbb{K}(x_\mu, x_j) &= x_{\mu 1}^2 x_{j1}^2 + x_{\mu 2}^2 x_{j2}^2 + 2x_{\mu 1} x_{j1} x_{\mu 2} x_{j2} + 2x_{\mu 1} x_{j1} + 2x_{\mu 2} x_{j2} + 1 \\ &= \langle g(x_\mu), g(x_j) \rangle \end{aligned} \tag{2.111}$$

where $g(x_\mu) = (x_{\mu 1}^2, x_{\mu 2}^2, \sqrt{2} x_{\mu 1} x_{\mu 2}, \sqrt{2} x_{\mu 1}, \sqrt{2} x_{\mu 2}, 1)$. In general, polynomial kernels satisfy Mercers condition.

- Radial Basis Function Kernels: The radial basis function (RBF) kernel is given by

$$\mathbb{K}(x_\mu, x_j) = \exp(-\gamma \left\| x_\mu - x_j \right\|^2), \tag{2.112}$$

where γ is a positive parameter that controls the kernel width.

Chapter 3

Ensemble Learning

3.1 Introduction

Ensemble learning is an effective machine learning paradigm that is used successfully in almost all kinds of applications such as text categorization, optical character recognition, face recognition, computer-aided medical diagnosis. An ensemble consists of a set of individual predictors (such as neural networks or decision trees) whose predictions are combined when classifying a given example (see Figure 3.1). Multiple classifiers combination to achieve higher accuracy is an important research area in a number of communities such as machine learning, pattern recognition, and artificial neural networks, and appears under different notions in the literature, e.g. multiple classifier systems (MCS), ensemble learning, classifier fusion, classifier ensembles, divide-and-conquer classifiers, mixture of experts [107]. The series of annual International Workshops on Multiple Classifier Systems (MCS), held since 2000, plays a vital role in organizing the development in the field of ensemble learning. Dietterich [55] suggested statistical, computational and representational reasons why it is possible to construct an ensemble of classifiers that is often more accurate than a single classifier. These three fundamental reasons represent the most important shortcomings of existing base learning algorithms (see Chapter 2). Hence, the aim of an ensemble method is to alleviate or eliminate these shortcomings.

- The *statistical problem*: A base learning algorithm *BaseLearner* can be considered as searching the hypothesis space $\mathbb{F}$, space of all possible classifiers, to identify the best single classifier f. When the available amount of training data is too small compared to the size of the classifier space, *BaseLearner* can not identify f. Although the data is insufficient, *BaseLearner* can still find many different classifiers in $\mathbb{F}$ that give good accuracy on the training data. By constructing an ensemble H of these accurate classifiers, one can find a good approximation of f.

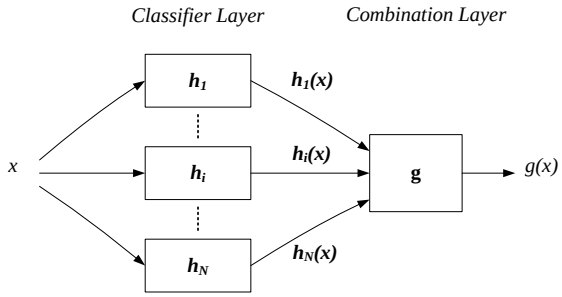


Figure 3.1: Two layer architecture of an ensemble where the aggregation of the decisions of the individual classifiers at the first layer is achieved by an additional combination layer.

- The *computational problem*: Many learning algorithm perform local search to optimize the classifier parameters and therefore get stuck in local optima. For instance, MLP neural network [25] depends on gradient descent to minimize an error function over training data. Although the *statistical problem* doe not exist (that is, the training data is enough), it may still be computationally not easy to find the best single classifier. By constructing ensemble of classifiers, where each classifier performs local search starting from different starting points, may provide a better approximation of f than any of the individual classifiers.

- The *representational problem*: In many machine learning applications, the best classifier f is not exist in the search space $\mathbb{F}$. By combining a set of classifiers drawn of $\mathbb{F}$, it may be possible to expand the space.

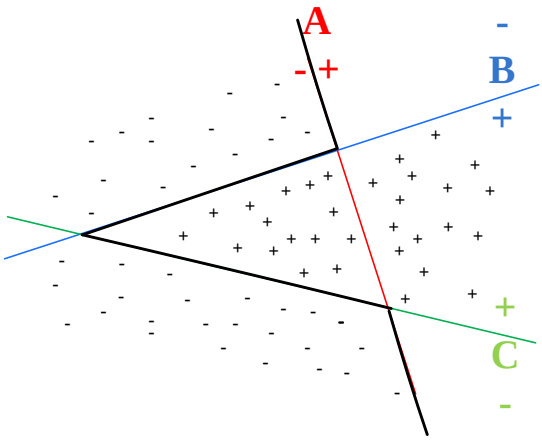


Figure 3.2: An ensemble of three linear classifiers

Figure 3.2 illustrates graphically the *representational problem* through three linear classifiers where the objective is to discriminate between positive (+) and negative examples (-). It is clear that the training data are not linearly separable.

Thus, none of the three straight lines can alone completely separate the positive and negative examples. For instance, classifier B can not correctly classify all the negative examples in the bottom half of the figure. However, the ensemble of three lines, where each line provides one vote, correctly classifies all the examples because for each example, at least two of the three linear classifiers correctly classify it, so the majority is always correct. This is the result of constructing an ensemble of three very different linear classifiers, this effective ensemble is a piecewise linear classifier (the bold line in Figure 3.2). This example clearly motivates the need to have ensemble members whose errors are not highly correlated.

3.2 Diversity

A necessary and sufficient condition for an ensemble of classifiers to outperform the accuracy of its individual members is to constitute accurate and diverse classifiers [78]. A classifier is accurate if it has error rate better than random guessing on unseen data. Diversity among classifiers means that they have independent (uncorrelated) errors, that is, they have different misclassified examples (for instance, the linear classifiers in Figure 3.2). As the errors of ensemble members increase, they become more identical. That is, there is a trade-off between average error and diversity of ensemble members.

3.2.1 How to Measure Diversity?

3.2.1.1 For Regression

The *error-ambiguity decomposition* [103] and *bias-variance-covariance decomposition* [191] qualitatively define the regression error diversity for linearly weighted ensembles by connecting it back to the mean squared error. The *ambiguity decomposition* [103] holds for convex combination functions and it breaks down the ensemble error E into two terms. The first term $\bar{E}$ is the weighted average error of the individuals, $\sum_i w_i(h_i(x) - y)^2$. The second term $\bar{A}$ is the ambiguity term, $\sum_i w_i(h_i(x) - H(x))^2$, indicates the variability in the ensemble member outputs for a given example x. That is, $E = \bar{E} - \bar{A}$. Since this term is subtractive from the first term and positive, it is guaranteed that ensemble error is less than or equal to the average individual errors. The larger the ambiguity term, the larger the ensemble error reduction provided that the first term is kept fixed. But, as the variability among the individuals increases, so does the value of the average individual errors. This shows that increasing the diversity alone is not enough, there should be a balance between diversity (the ambiguity term) and individual accuracy (the average error term), in order to minimize the ensemble error E. The importance of the *error-ambiguity decomposition* is that it shows that given any set of predictors, the error of the convex-combined ensemble will be less than or equal to the average error of the individuals. If $\bar{A} = 0$ for a given set

of examples, then $E = \bar{E}$ and this means that there is no performance gain from combining identical predictors.

The *bias-variance-covariance decomposition* [191] for an ensemble is examined to understand the regression error diversity. It holds for convex combination functions and it breaks down the ensemble generalization error in the form of mean squared error (MSE) into three terms: bias, variance and covariance.

$$Err(H) = E\{(H - y)^2\} = \overline{bias}^2 + \frac{1}{N}\overline{var} + (1 - \frac{1}{N})\overline{covar} \tag{3.1}$$

where H is the ensemble output for a given example x and is defined as the average of the N classifier outputs,

$$H(x) = \frac{1}{N}\sum_{i=1}^{N} h_i(x), \tag{3.2}$$

the average bias of the ensemble members is

$$\overline{bias} = \frac{1}{N}\sum_{i}(E\{h_i\} - y), \tag{3.3}$$

the average variance of the ensemble members is

$$\overline{var} = \frac{1}{N}\sum_{i} E\{(E\{h_i\} - h_i)^2\} \tag{3.4}$$

and the average covariance of the ensemble members is

$$\overline{covar} = \frac{1}{N(N-1)}\sum_{i}\sum_{j\neq i} E\{(h_i - E\{h_i\})(h_j - E\{h_j\})\} \tag{3.5}$$

Note that there should be a balance between bias and variance because attempts to reduce the bias component will cause an increase in variance, and vice versa. Also the covariance term indicates that the mean square error of an ensemble estimator depends on the amount of error correlation between individual estimators. We would ideally like to decrease the covariance, without causing any increase in the bias or variance terms. Unlike the ambiguity decomposition that depends on a given training set, the Bias-Variance-Covariance decomposition takes into account the distribution over possible training sets. This is an advantage because what we are interested in, is the expected error on unseen data points given these distributions.

3.2.1.2 For Classification

It is not easy to provide a solid quantification of classification error diversity although there is a much clearer framework for explaining the role of regression error

diversity because classification ensemble methods usually depend on non-linear combination methods like majority voting. We would like to have an expression that similarly decomposes the classification error rate into the error rates of the individuals and a term that quantifies their 'diversity'. Although this is beyond the present state of the art, a number of empirical studies have tried to derive heuristic expressions that may approximate this unknown error diversity term.

Margineantu and Dietterich [124] proposed to use the kappa-statistic κ as a pairwise agreement measure. It is defined as follows: Given two classifiers h_1 and h_2, K classes and m examples, we can define a coincidence matrix C where element C_{ij} represents the number of examples that are assigned by the first classifier to class ω_i and by the second classifier to class ω_j. Then, the agreement measure κ is defined as follows:

$$\kappa = \frac{\theta_1 - \theta_2}{1 - \theta_2} \tag{3.6}$$

where

$$\theta_1 = \frac{\sum_{i=1}^{K} C_{ii}}{m} \quad \text{and} \quad \theta_2 = \sum_{i=1}^{K} \left(\sum_{j=1}^{K} \frac{C_{ij}}{m} \sum_{j=1}^{K} \frac{C_{ji}}{m} \right) \tag{3.7}$$

If h_1 and h_2 are identical, only the main diagonal of C will contain non-zero elements (θ_1=1) and $\kappa = 1$. If h_1 and h_2 are totally different, their agreement (θ_1) will be the same as the agreement by chance (θ_2) and $\kappa = 0$. If h_1 and h_2 are negatively dependent, then $\kappa < 0$ and when one classifier is wrong, the other has more than random chance of being correct. The *Kappa-Error diagram* is introduced in [124] as a way for visualizing the relationship between a given ensemble diversity and accuracy. The x-axis of the Kappa-error diagram represents the $\kappa_{i,j}$ between two ensemble members h_i and h_j while the y-axis represents the average error $E_{i,j} = \frac{E_i + E_j}{2}$. Therefore, an ensemble of N members has a cloud of $\frac{N(N-1)}{2}$ points in the diagram. The *Kappa-Error diagram* can be calculated using the training set or a separate validation set. Since small values of κ indicate better diversity and small values of $E_{i,j}$ indicate better accuracy, the most desirable cloud will lie in the bottom left corner. This is useful for visual evaluation of the relative positions of clouds for different ensembles. Comparing clouds of points for AdaBoost versus Bagging, they verified that AdaBoost produces more diverse ensembles of classifiers than Bagging.

Kuncheva and Whitaker[110] divided the diversity measures into two types: pairwise and non-pairwise. Pairwise measures calculate the average of a particular similarity metric between all possible pairs of classifiers in the ensemble. The difference between a diversity measure and another is the underlying similarity metric. On the other hand, the non-pairwise diversity measures either use the idea of entropy or calculate a correlation of each ensemble member with the averaged output. They studied ten statistics that can be used to measure diversity among binary classifiers: four averaged pairwise measures (the Q statistic, the

correlation, the disagreement and the double fault) and six non-pairwise measures (the entropy of the votes, the difficulty index, the Kohavi-Wolpert variance, the interrater agreement, the generalized diversity and the coincident failure diversity). They found that most of these measures are highly correlated. However, to the best of our knowledge, there is not a conclusive study showing which measure of diversity is the best to use for constructing and evaluating ensembles.

Brown [35] had investigated the issue of ensemble diversity from an information theoretic perspective. The main finding was an expansion of the ensemble mutual information into accuracy term, several diversity and conditional diversity terms.

$$I(X_{1:n};Y) = \sum_{i=1}^{n} I(X_i;Y) - \sum_{\substack{X \subseteq S \\ |X|=2..n}} I(\{X\}) + \sum_{\substack{X \subseteq S \\ |X|=2..n}} I(\{X\}|Y) \qquad (3.8)$$

where $X_{1:n}$, X_i, Y are random variables represent the ensemble output, the i^{th} classifier output and the target output, respectively. In addition, $I(\{X\})$ and $I(\{X\}|Y)$ represent the multi-variate mutual information and class-conditional mutual information among $|X|$ classifiers, respectively. He stated that this expansion reflects the true complexity of the accuracy-diversity issue. Error diversity is not simply a pairwise measure between classifiers, such as the Q-statistics or the Double-Fault measures. In fact, diversity exists on several levels of interaction between the classifiers, having high and low order terms.

3.2.2 How to Create Diversity?

Kuncheva [106] provided a graphical illustration of four different approaches to build ensembles of diverse classifiers as shown in Figure 3.3. What differentiates between ensemble methods is the way used to promote the diversity between the member classifiers of an ensemble. Approach (a), which is based on *training set manipulation*, refers to ensemble methods that combine classifiers trained on different training sets, i.e. *bagging* [31] and *boosting* [65]. Approach (b), which is based on *feature set manipulation*, includes techniques that combine classifiers trained on different feature subsets, such as *Random Forests* [32] and *Random Subspace method* [82]. Approach (c) includes heterogeneous ensemble methods that combine classifiers trained using different learning algorithms such as decision tree, neural network and k-nearest neighbor classifiers [203]. On the other hand, homogeneous methods that combine classifiers trained using the same learning algorithm. Approach (d) assumes that the diverse classifiers are given and the task is to select the best combination method for the given learning task. Many ensemble methods have been developed that use other heuristics to promote diversity: manipulation of the output labels such as Error Correcting Output Coding (ECOC) [57] (see Chapter 4) and randomness injection such as training a set of neural networks with different random initializations of weights,

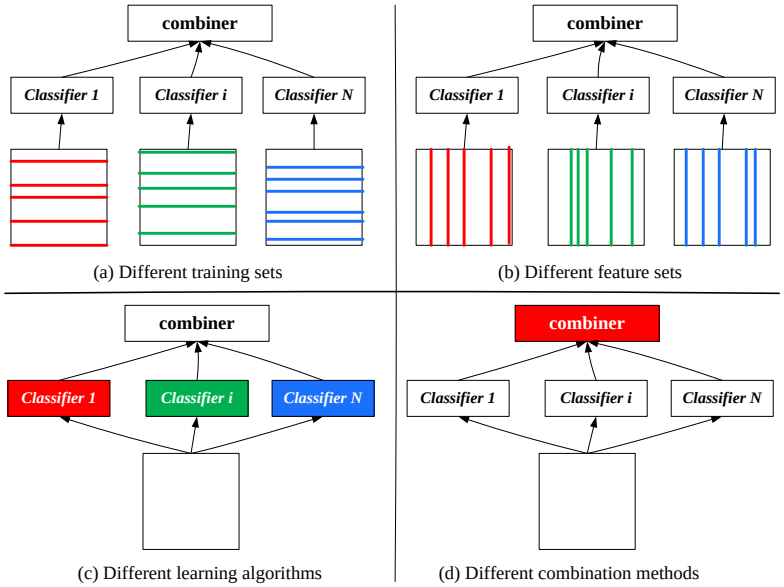


Figure 3.3: Four approaches to create an ensemble of diverse classifiers

or different architectures [100] or training a set of C4.5 decision trees through introducing some randomness into the feature selection for splitting tree nodes [56].

3.3 Taxonomies of Combination Methods

3.3.1 Selection and Fusion

There are two general combination paradigms: classifier selection (complementary ensemble) and classifier fusion (competitive ensemble). The former methods assume that each classifier is an expert in some local region of the input space. For a given example $x \in \mathbb{R}^D$, there is exactly one classifier responsible for the region of x and it is given the highest priority to label x [203]. The latter assumes that all classifiers are trained over the whole input space, and they are taking into consideration to classify x. Selection and fusion can be merged together such as in[87, 15], instead of selecting one expert, the decisions of more than one local expert are considered where each classifier is weighted by the level of expertise it has on x.

3.3.2 Hard, Ranking and Soft Combiners

Let $H = \{h_1, \ldots, h_N\}$ be a set of classifiers and $\Omega = \{\omega_1, \ldots, \omega_K\}$ be a set of class labels. Combining classifiers means to predict a class label to a given example x

based on the N classifier outputs $h_1(x), \ldots, h_N(x)$. Base classifiers can be divided into three types according to their outputs:

- *Crisp(Hard) classifier.* Each classifier h_i gets a feature vector $x \in \mathbb{R}^{D_i}$ as input and assigns it to a *crisp class label* from Ω.

$$h_i : \mathbb{R}^{D_i} \to \Omega \text{ or } h_i : \mathbb{R}^{D_i} \times \Omega \to \{0, 1\} \tag{3.9}$$

  where $\sum_{k=1}^{K} h_i(x, \omega_k) = 1$.

- *Ranking classifier.* Each classifier h_i is given by

$$h_i : \mathbb{R}^{D_i} \to 2^{\Omega} \tag{3.10}$$

  where the classifier returns a subset of Ω sorted in ascending order according to a rank function r such that: for each $\omega_a \in h_i(x)$ and $\omega_b \notin h_i(x), r(\omega_a) < r(\omega_b)$. Sometimes, it is called *multi-label classifier* because it assign a set of class labels to a given example x instead of a single label. This type is beyond the scope of this thesis.

- *Soft classifier.* Each classifier h_i gives as output a *soft class label*, that is

$$h_i : \mathbb{R}^{D_i} \to [0, 1]^K \text{ or } h_i : \mathbb{R}^{D_i} \times \Omega \to [0, 1] \tag{3.11}$$

  where the classifier output $h_i(x, \omega_k)$ can be viewed as the belief, evidence, certainty, probability or possibility of the hypothesis that x belongs to class ω_k and thus can be divided into:

  - *Possibilistic classifier.* if $\sum_{k=1}^{K} h_i(x, \omega_k) > 0$
  - *Probabilistic or Fuzzy classifier.* if $\sum_{k=1}^{K} h_i(x, \omega_k) = 1$, then $h_i(x, \omega_k)$ is a class posterior probability estimate, that is $h_i(x, \omega_k) = P(\omega_k|x)$

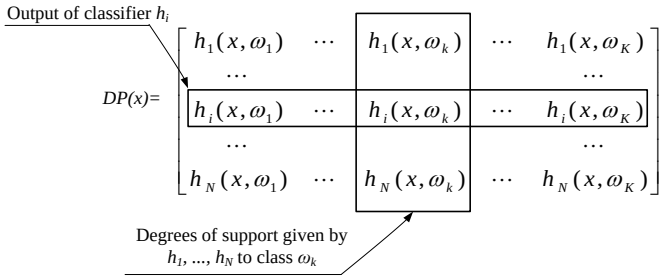
  Note that the crisp classifier is as a special case of the soft classifier, because the class label $\hat{y}$ assigned to a given example x is the class with the maximum membership degree

$$\hat{y} = arg \max_{1 \le k \le K} h_i(x, \omega_k) \tag{3.12}$$

An ensemble H is constructed by combining the outputs of the N classifiers as

$$H(x) = g(h_1(x), \ldots, h_N(x)) \text{ or } H(x, \omega_k) = g(h_1(x, \omega_k), \ldots, h_N(x, \omega_k)) \tag{3.13}$$

where g is the combination function that is called *hard combiner* if it uses the crisp class labels provided by the crisp classifiers of the ensemble. It is called *soft combiner* if it combines the *soft class label* of the individual soft classifiers. To facilitate the combination of the N classifier outputs, they can be stored in a $N \times K$ matrix, that is called *decision profile* and defined as follows,



(3.14)

3.3.3 Class-Conscious and Class-Indifferent Combiners

Based on the way of using the decision profile to find the overall support for each class ω_k, the combination methods are divided by Kuncheva [107] into class-conscious and class-indifferent, see Table 6.1. The class-conscious methods use only the k^{th} column of $DP(x)$ such as average, minimum, maximum and product rules. These type of methods use the context of the profile but lose part of the information because they use only column per class. The class-indifferent methods ignore the context of the decision profile and use all of $DP(x)$ as features in a new feature space, which is called the *intermediate feature space*. Any classifier can be used with the intermediate features as inputs and the class label as the output.

Table 3.1: Taxonomy of Combination Methods

combiner	Static (nontrainable)	Dynamic (trainable)
crisp	Majority Vote [112] Weighted Majority Vote [65]	Behaviour knowledge space [85] Naive Bayes [204] Wernecke Method[199]
ranking	Borda count [80]	Generalized borda count [143]
soft	Class Conscious Average Minimum Maximum Product Ordered weighted averaging [111]	Class Conscious Weighted Average Fuzzy Integral [41] Probabilistic Product Pseudoinverse matrix Class Indifferent Neural Networks [84],[3] Stacked Generalization [202] Dempster-Shafer method [158],[60] Decision Templates [109]

3.3.4 Trainable and Nontrainable Combiners

The combination methods can be divided into static (nontrainable) and dynamic (trainable) [59], see Table 6.1. The former refers to the combination rules that do not need training after the individual classifiers have been trained. For instance, Majority Voting, average or minimum. The latter are fusion functions that require additional training after the training of the individual classifiers whether it depends on an additional training set or uses the same training set. For instance, Behavior-Knowledge Space, Decision Templates [109], Naive-Bayes and Neural combiners. Experimental studies [109] show that adaptive fusion methods especially Decision Templates outperform fixed combination rules. A third class of combiners where the combiner are trained during the classifiers training, for instance, the *weighted majority vote* used in AdaBoost [65].

3.4 Ensemble Learning Algorithms

In the following sections, we present the ensemble methods that are used in this thesis. For an excellent survey on ensemble methods see [36].

3.4.1 Manipulation of Training Set

3.4.1.1 Bagging

Given a training set L of size m, standard *Bagging* [31] creates N base classifiers $h_i : i = 1, \ldots, N$ (See Algorithm 1). Each classifier is constructed using the base learning algorithm *BaseLearn* on a bootstrap sample of size m created by random resampling with replacement from the original training set. Breiman [31] mentioned that each bootstrap sample contains approximately 63% of the original training set, where each example can appear multiple times. He also indicated that, given N training examples, the probability that the i^{th} training example is selected zero or more times is approximately Poisson distributed with $\lambda = 1$. In the prediction phase, the class label assigned to given example x is the class with the maximum probability $P(\omega_k|x)$. In case of soft classifiers, $P(\omega_k|x)$ is the average of the N class probability distributions produced by the ensemble members $P_i(x)$. In case of crisp classifiers, $P(\omega_k|x)$ is the number of votes given to class ω_k divided by N (*majority vote*).

This technique works well for unstable base learning algorithms, where a small change in the input training set can lead to a major change in the output hypothesis. The learning algorithms of decision trees and neural networks are well-known as unstable algorithms but the linear classifiers, *k-nearest neighbor* (*kNN*) and Naive-Bayes learning algorithms are generally stable especially when the training set size is large. Breiman [31] and Davidson [49] showed that Bagging does not work well when applied to stable learners. The reason can be that the aim of

Algorithm 1 *Bagging* Algorithm

Require: Original training set L, Base learning algorithm ($BaseLearn$), number of bagging iterations (N)

Training Phase

1: **for** $i = 1$ to N do
2: $S_i = BootstrapSample(L)$
3: $h_i = BaseLearn(S_i)$
4: **end for**
5: **return** ensemble $\{h_1, \ldots, h_N\}$

Prediction Phase

6: **return** $H(x) = arg\max_{1 \leq k \leq K} P(\omega_k|x)$
for hard classifiers, $P(\omega_k|x) = \frac{1}{N} \sum_{h_i(x)=\omega_k} 1$, for $k = 1, \ldots, K$
for soft classifiers, $P(\omega_k|x) = \frac{1}{N} \sum_{i=1}^{N} P_i(\omega_k|x)$, for $k = 1, \ldots, K$

Bagging is to reduce the variance of the underlying base learner and the variance of stable learners is already low so it is hard to decrease it more. Zhou et al. [217] adapted Bagging to *k-nearest neighbor* classifiers through injecting randomness to distance metrics. That is, to construct the ensemble members, both the training set and the distance metric employed for determining the neighbors are perturbed. The empirical study reported in this paper shows that the proposed algorithm, *BagInRand*, can construct ensembles that effectively improve the accuracy over a single *kNN* classifier. Oza and Tumer [141] and Skalak [178] showed that an ensemble of *kNN* classifiers where each member trained using a small set of prototypes selected from the whole training set outperforms a single *kNN* classifier using all the training examples.

3.4.1.2 Boosting

Boosting is a family of ensemble learning algorithms that are very effective in improving performance compared to the ensemble members. *AdaBoost.M1* described in [65] is the most popular algorithm (See Algorithm 2). It introduces the diversity through generating a sequence of base classifiers h_i using weighted training sets (weighted by D_1, ..., D_N) such that the weights of training examples misclassified by classifier h_{i-1} are increased and the weights of correctly classified examples are decreased in order to enforce the next classifier h_i to focus on the *hard-to-classify* examples at the expense of the correctly classified examples (bias correction). That is, for each iteration i, the aim of *AdaBoost* is to construct a classifier h_i that improve the training error of classifier h_{i-1}. Consequently, *AdaBoost* stops if the training error of the classifier is zero or worse than random guessing. In the prediction phase, the class label assigned to given example x is the class with the maximum probability $P(\omega_k|x)$. In case of soft classifiers, $P(\omega_k|x)$ is the weighted average of the N class probability distributions produced

Algorithm 2 *AdaBoost* Algorithm

Require: Original training set ($L = \{(x_j, y_j)\}_{j=1}^{m}$), Base learning algorithm ($BaseLearn$), number of boosting iterations (N)

Training Phase

1: Initialize $D_1(j) = 1/m \quad \forall j \in \{1, \ldots, m\}$
2: **for** $i = 1$ to N **do**
3: $h_i = BaseLearn(L, D_i)$
4: Calculate the training error of h_i: $\epsilon_i = \sum_{j=1}^{m} D_i(j) \times I(h_i(x_j) \neq y_j)$
5: **if** $\epsilon_i = 0$ or $\epsilon_i \geq 1/2$ **then**
6: Set $N = i - 1$ and abort loop
7: **end if**
8: Set the weight of h_i: $w_i = \log(\beta_i)$ where $\beta_i = \frac{1-\epsilon_i}{\epsilon_i}$
9: Update weights of training examples:

$$D_{i+1}(j) = D_i(j) \times \begin{cases} \beta_i & \text{if } h_i(x_j) \neq y_j; \\ 1 & \text{otherwise.} \end{cases}$$

10: Normalize, $D_{i+1}(j) = D_{i+1}(j)/Z_i$ where $Z_i = \sum_{j'=1}^{m} D_{i+1}(j')$
11: **end for**
12: **return** ensemble $\{h_1, \ldots, h_N\}$

Prediction Phase

13: **return** $H(x) = arg\max_{1 \leq k \leq K} P(\omega_k|x)$
for hard classifiers, $P(\omega_k|x) = \frac{1}{\sum_{i=1}^{N} w_i} \sum_{h_i(x)=\omega_k} w_i$, for $k = 1, \ldots, K$
for soft classifiers, $P(\omega_k|x) = \frac{1}{\sum_{i=1}^{N} w_i} \sum_{i=1}^{N} w_i P_i(\omega_k|x)$, for $k = 1, \ldots, K$

by the ensemble members $P_i(x)$. In case of soft classifiers, $P(\omega_k|x)$ is the weighted votes given to class ω_k divided by N (*weighted majority vote*). Note that each member is assigned a weight w_i based on its training error.

Applying *AdaBoost* to decision trees is successful and is considered one of the best off-the-shelf classification methods. Despite its popularity, *AdaBoost* has two drawbacks [56]: it performs poorly given a small training set and also when there is a lot of training examples with incorrect class label (*mislabeling noise*).

3.4.2 Manipulation of Feature Set

3.4.2.1 Random Subspace Method (*RSM*)

Random Subspace Method (*RSM*) is an ensemble learning algorithm proposed by Ho [82]. The diversity is promoted through feature set manipulation instead of training set manipulation. That is, if the given data set is represented by D features, then d features are randomly selected resulting in a d-dimensional

random subspace of the original D-dimensional feature space. Then for each random subspace a classifier is constructed. The prediction of the committee for a given sample x is the average of the N class probability distributions provided by the ensemble members $P_i(x)$ then the most likely class of x is the class with the maximum probability. (in our study, $d = \frac{D}{2}$)

Algorithm 3 *RandomSubspaceMethod*

Require: L= $\{(x_j, y_j)\}_{j=1}^{m}$ where $x_j = (x_{j1}, \ldots x_{jD})$ - Original training set
$BaseLearn$ - Base learning algorithm
N - number of iterations
d - number of randomly selected features ($d < D$)

1: **for** $i = 1$ to N **do**
2: $S_i = RandomFeatureSelection(L, d)$
3: $h_i = BaseLearn(S_i)$
4: **end for**
5: The final hypothesis (*simple averaging*):
6: $H(x) = arg\max_{1 \leq k \leq K} P(x)$ where $P(x) = \frac{1}{N} \sum_{i=1}^{N} P_i(x)$
7: **return** ensemble H

3.4.2.2 Random Forest

Breiman [32] introduced an extended version of Bagging, that is called *Random Forest*. It combines Bagging with random feature selection for decision trees. In this method, each member of the ensemble is trained on a bootstrap sample from the original training set as in Bagging. Decision trees are then grown by selecting the feature to split on at each node from randomly selected feature subset F instead of the full feature set. Breiman [32] set the size of F to $\lfloor \log_2(D + 1) \rfloor$, where D is the total number of features. In order to maintain diversity the output random trees are not pruned. *Random Forest* has better diversity than *Bagging* because it depends on two sources of diversity: training set manipulation and feature set manipulation. Dietterich [54] recommends *Random Forest* as the method of choice for decision trees, as it compares favorably to *AdaBoost* and works well even with noise in the training set. The main shortcoming of *Random Forest* is that it can only be applied to decision trees.

3.4.3 Manipulation of the Output Targets

Unlike all the previous ensemble learning algorithms that learn ensemble members to discriminate among the same set of classes Ω. This family of ensemble learning algorithms creates the diversity through constructing the different ensemble members using different target classes. Chapter 4 discusses this family in more details.

Chapter 4

Multi-Class Learning

4.1 Introduction

Many real-world pattern recognition problems involve a large number of classes where the learning task is to assign single class label from a set of K labels to each input example. This learning paradigm usually called *Multi-Class Learning* and is divided into two directions. The first direction is to directly apply existing base learning algorithms (see Chapter 2) provided that they can be easily generalized to handle these multiple classes such as the neural networks and decision trees. The second direction is to decompose the multi-class problem into a set of binary-class problems and then to apply binary class learning algorithms to solve each problem separately. The importance of the second direction appears more when the underlying base learning algorithm can not handle multiple classes such as the perceptron algorithm [160] and support vector machine algorithm [193]. The experiments conducted by Dietterich [57] had shown that an ensemble of binary-class decision trees outperforms a single multi-class decision tree and the same conclusion for neural networks. The family of approaches that adopts the second direction is referred to as *output space decomposition* or *multi-class decomposition* techniques and it includes: one-against-others, one-against-one (pairwise) [101, 105], error-correcting output coding [57] and tree-structured (hierarchical) approaches [60]. In the following sections each of these approaches is briefly explained except for the tree-structured approach that is explained in more details because it will be used in the experiments. To facilitate the explanation, we assume that $L = \{(x_\mu, y_\mu)|x_\mu \in \mathbb{R}^D, y_\mu \in \Omega, \mu = 1, \ldots, m\}$ is the given set of training examples and $\Omega = \{\omega_1, \ldots, \omega_K\}$ is the predefined set of class labels.

4.2 One-Against-Others Approach

4.2.1 Training Phase

The single multi-class data set is decomposed into a set of K binary-class data sets (L_k), for each $k = 1, \ldots, K$, that is one binary problem for each class ω_k. For each class ω_k, a binary classifier is trained to discriminate between it and the other classes using the examples of L that belong to ω_k as positive examples (labeled 1) and all the examples of the other classes as negative examples (labeled -1), see Table 4.1(a). The rows represent classes and columns represent classifiers. For instance, for the digits recognition task, 10 binary classifiers are constructed.

Tumer and Ghosh [188] reduce the correlation among classifiers in an ensemble by training them with different feature subsets. They train K classifiers, one corresponding to each class in a K-class problem. For each class ω_k, a subset of features that have a low correlation to that class is eliminated. The degree of correlation among classifiers is controlled by the amount of eliminated features. This method, called *input decimation*, has been further explored by Tumer and Oza [141]. Experimental results on three data sets showed the advantage of input decimation over using combiners based on dimensionality reductions relying on Principle Component Analysis (PCA).

The main drawback of this decomposition approach is that it leads to imbalanced training sets [164] because the number of negative examples is $K-1$ times greater the the number of positive examples if the number of examples is equal for all K classes. Classifiers generally perform poorly on imbalanced datasets. For instance, if a support vector machine is used as binary classifier, the decision boundary will be biased to the negative examples. Another drawback is the *False Positives*. In the classification phase, it is expected that exactly one of the K classifiers replies with positive answer but for large K often more than one classifier reply positively which is a tie that must be broken by additional criteria.

4.2.2 Classification Phase

For each class ω_k, if h_k is a hard classifier $h_k : \mathbb{R}^D \rightarrow \{0, 1\}$,that is,

$$h_k(x) = \begin{cases} 1 & \text{if } x \text{ is assigned to } \omega_k \ , \\ 0 & \text{if } x \text{ is not assigned to } \omega_k. \end{cases} \tag{4.1}$$

Then a given example x is assigned a crisp class label ω_i if h_i respond positively, that is $h_i(x) = 1$. The main drawback of this combination rule is that sometimes more than one binary classifier can give a positive answer. In this case, tie can be broken arbitrary. This problem can be avoided if h_k is a soft classifier, $h_k : \mathbb{R}^D \times \{\omega_k, \neg\omega_k\} \rightarrow [0, 1]$, then the final soft class label assigned to x is the

Table 4.1: Decomposition of a 5-class problem into 5 binary classification problems using the *One-Against-Others Approach*.

(a) Training Phase

-	h_1	h_2	h_3	h_4	h_5
ω_1	1	-1	-1	-1	-1
ω_2	-1	1	-1	-1	-1
ω_3	-1	-1	1	-1	-1
ω_4	-1	-1	-1	1	-1
ω_5	-1	-1	-1	-1	1

(b) Classification Phase

	ω_1	ω_2	ω_3	ω_4	ω_5
ω_1	$h_1(x, \omega_1)$	$h_2(x, \neg\omega_2)$	$h_3(x, \neg\omega_3)$	$h_4(x, \neg\omega_4)$	$h_5(x, \neg\omega_5)$
ω_2	$h_1(x, \neg\omega_1)$	$h_2(x, \omega_2)$	$h_3(x, \neg\omega_3)$	$h_4(x, \neg\omega_4)$	$h_5(x, \neg\omega_5)$
ω_3	$h_1(x, \neg\omega_1)$	$h_2(x, \neg\omega_2)$	$h_3(x, \omega_3)$	$h_4(x, \neg\omega_4)$	$h_5(x, \neg\omega_5)$
ω_4	$h_1(x, \neg\omega_1)$	$h_2(x, \neg\omega_2)$	$h_3(x, \neg\omega_3)$	$h_4(x, \omega_4)$	$h_5(x, \neg\omega_5)$
ω_5	$h_1(x, \neg\omega_1)$	$h_2(x, \neg\omega_2)$	$h_3(x, \neg\omega_3)$	$h_4(x, \neg\omega_4)$	$h_5(x, \omega_5)$

average of the K classifier outputs, see Table 4.1(b), as follows:

$$H(x, \omega_i) = \frac{1}{K}\left(h_i(x, \omega_i) + \sum_{k=1, k\neq i}^{K} h_k(x, \neg\omega_k)\right) \tag{4.2}$$

Thus, the predicted class label $\hat{y}$ for a given example x is,

$$\hat{y} = arg \max_{1\leq k\leq K} H(x, \omega_k) \tag{4.3}$$

Both training and classification time complexity are linear with respect to the number of classes.

4.3 One-Against-One (Pairwise) Approach

4.3.1 Training Phase

This multi-class decomposition schema transforms the multi-class data set into $K(K-1)/2$ binary-class data sets ($L_{i,j}$), one for each pair of classes (ω_i, ω_j), for each $i, j = 1, \ldots, K$. A binary classifier $h_{i,j}$ is trained to discriminate between the two classes using the examples in L that belong to class ω_i as positive examples (labeled 1), those belonging to ω_j as negative examples (labeled -1) and the other examples are not used (labeled 0), see Table 4.2(a). The rows represent classes and columns represent classifiers. For instance, the digits recognition task is solved by constructing 45 binary classifiers.

Table 4.2: Decomposition of a 5-class problem into 10 binary classification problems using the *One-Against-One Approach*

(a) Training Phase

	$h_{1,2}$	$h_{1,3}$	$h_{1,4}$	$h_{1,5}$	$h_{2,3}$	$h_{2,4}$	$h_{2,5}$	$h_{3,4}$	$h_{3,5}$	$h_{4,5}$
ω_1	1	1	1	1	0	0	0	0	0	0
ω_2	-1	0	0	0	1	1	1	0	0	0
ω_3	0	-1	0	0	-1	0	0	1	1	0
ω_4	0	0	-1	0	0	-1	0	-1	0	1
ω_5	0	0	0	-1	0	0	-1	0	-1	-1

(b) Classification Phase

	ω_1	ω_2	ω_3	ω_4	ω_5
ω_1	-	$h_{1,2}(x,\omega_1)$	$h_{1,3}(x,\omega_1)$	$h_{1,4}(x,\omega_1)$	$h_{1,5}(x,\omega_1)$
ω_2	$h_{1,2}(x,\omega_2)$	-	$h_{2,3}(x,\omega_2)$	$h_{2,4}(x,\omega_2)$	$h_{2,5}(x,\omega_2)$
ω_3	$h_{1,3}(x,\omega_3)$	$h_{2,3}(x,\omega_3)$	-	$h_{3,4}(x,\omega_3)$	$h_{3,5}(x,\omega_3)$
ω_4	$h_{1,4}(x,\omega_4)$	$h_{2,4}(x,\omega_4)$	$h_{3,4}(x,\omega_4)$	-	$h_{4,5}(x,\omega_4)$
ω_5	$h_{1,5}(x,\omega_5)$	$h_{2,5}(x,\omega_5)$	$h_{3,5}(x,\omega_5)$	$h_{4,5}(x,\omega_5)$	-

4.3.2 Classification Phase

For each pair (ω_i, ω_j) such that $i < j$, if $h_{i,j}$ is a hard classifier, $h_{i,j} : \mathbb{R}^D \rightarrow \{0, 1\}$, that is,

$$h_{i,j}(x) = \begin{cases} 1 & \text{if } x \text{ is assigned to } \omega_i, \\ 0 & \text{if } x \text{ is assigned to } \omega_j. \end{cases} \tag{4.4}$$

Then a hard combiner such as majority vote is used to predict the class label of a given example x where all binary classifiers are applied. The support given for each class ω_i is the number of votes for this class, that is,

$$H(x, \omega_i) = \frac{1}{K-1} \left(\sum_{\substack{k=1, \\ h_{k,i}(x)=1}}^{i-1} 1 + \sum_{\substack{k=i+1, \\ h_{i,k}(x)=1}}^{K} 1 \right) \tag{4.5}$$

If $h_{i,j}$ is a soft classifier (see Section 3.3.2), $h_{i,j} : \mathbb{R}^D \times \{\omega_i, \omega_j\} \rightarrow [0, 1]$, then the final soft class label assigned to x is the average of the $K(K-1)/2$ classifier outputs, see Table 4.2(b), as follows:

$$H(x, \omega_i) = \frac{1}{K-1} \left(\sum_{k=1}^{i-1} h_{k,i}(x, \omega_i) + \sum_{k=i+1}^{K} h_{i,k}(x, \omega_i) \right) \tag{4.6}$$

Thus, the predicted class label $\hat{y}$ for a given example x is,

$$\hat{y} = arg \max_{1 \leq k \leq K} H(x, \omega_k) \tag{4.7}$$

The main drawback is that both training and classification time complexity are quadratic with respect to the number of classes. Which is computationally expensive especially when K is large.

4.4 Error-Correcting Output Codes (ECOC)

4.4.1 Training Phase

This multi-class decomposition schema was introduced by Dietterich and Bakiri [57]. In this technique a K-class data set is broken down into a set of N binary-class data sets. The basic idea is to create a codeword for each class and to arrange these K codewords as rows of a matrix M, that is called *code matrix*, where $M \in \{-1, 1\}^{K \times N}$ and N is the code length. This is a special case of the code matrix proposed in [14] where $M \in \{-1, 0, 1\}^{K \times N}$. In this general case, some entries M_{ij} in the matrix can be zero indicating that the corresponding class ω_i is not taken into account by the corresponding binary classifier h_j. A binary classifier h_i is trained to discriminate between the set of positive classes (labeled 1) and the set of negative classes (labeled -1) that omits the training examples of the other classes (labeled 0), see Table 4.3. From the perspective of machine learning, the matrix M represent a set of N binary classification tasks, one for each column. The way used to create this *code matrix* is called the *encoding strategies*. The *One-Against-Others* and *One-Against-One* are the most popular encoding strategies. Other heuristics are sparse random codes [57] and dense random codes [14]. In the dense random codes, each entry in the code matrix is selected uniformly at random from the set $\{-1, 1\}$. Allwein et al. proposed to set the code length $N = 10 \log_2(K)$. The dense matrix is created by choosing the matrix that has the largest minimum Hamming decoding distance among each pair of codewords in the matrix - the matrix with trivial and complementary codes is discarded. The second random approach - sparse random codes - takes its values from the pool $\{-1, 0, 1\}$. Each entry of the code matrix is 0 with probability 1/2 and -1 or 1 with probability 1/4. The length of the sparse codeword is set to $15 \log_2(K)$. Again, the matrix with the largest minimum Hamming decoding distance is selected considering that no trivial or complementary codes are present. All these encoding strategies are defined independently of the data set and satisfy two conditions:

- **Row separation**. Each codeword should be well-separated in terms of Hamming distance from each of the other codewords.
- **Column separation**. Each column h_i should be uncorrelated with all the other columns h_j, $j \neq i$. This condition is fulfilled if the Hamming distance between a column and the rest is maximized. This condition is necessary to avoid adding identical classifiers into the ensemble.

4.4.2 Classification Phase

The codeword is formed by applying the N binary classifiers h_i to a given example x and concatenating the results into a vector $h(x) = (h_1(x), \ldots, h_N(x))$. The

Table 4.3: Decomposition of a 5-class problem into 7 binary classification problems using the *Error-Correcting Approach*

	h_1	h_2	h_3	h_4	h_5	h_6	h_7
ω_1	1	0	1	1	0	1	1
ω_2	0	1	0	0	1	0	1
ω_3	0	0	1	1	1	1	0
ω_4	1	1	0	0	1	0	1
ω_5	0	1	1	0	0	0	0

way of combining the N classifier outputs to assign one of the K labels to x is called the *decoding strategy*. The simplest decoding strategy is to measure the closeness between the N codewords and the prediction vector. The closeness is measured by finding the Hamming distance between the codewords and the vector $h(x)$. Hamming distance d_H is the number of positions where the two compared codewords differ. The class with the nearest codeword to $h(x)$ is assigned to x,

$$\hat{y} = arg \min_{1 \leq k \leq K} d_H(M(\omega_k), h(x)) \tag{4.8}$$

This approach of predicting the class label is known as the *Hamming Decoding*. The main drawback of ECOC schema is that the encoding strategies used to design the code matrix do not take into account the dependencies among classes. This might lead to binary problems that are unnatural and hard to solve especially if the number of classes is large. Another drawback is the number of binary classifiers N is a parameters of the algorithm. It is not easy to find the minimum number of classifiers to achieve high classification performance. For instance, ECOC constructs a large number of classifiers for the random encoding strategies, dense and sparse, in order to achieve good accuracy.

4.5 Decision Directed Acyclic Graphs (DDAG)

This multi-class decomposition technique was introduced by Platt et al. [148]. DAG is a graph whose edges have direction and no cycles. For a K class problem, DAG include $K(K-1)/2$ internal nodes where each node contains a binary classifier to discriminate between a pair of classes. The nodes within the graph start with a root node at the top and spanned into two nodes at the second layer, 3 nodes at the third layer and so on until the final layer is reached that consists of K leaves representing the class labels, see Figure 4.1. The total depth of the graph is K-1.

4.5.1 Training Phase

The training phase is the same as that of the *One-against-One* technique (see Section 4.3).

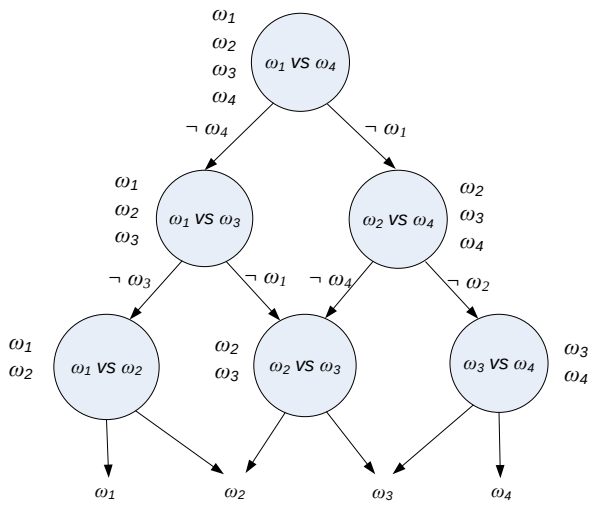


Figure 4.1: The structure of the Decision Directed Acyclic Graph for a 4-class problem

4.5.2 Classification Phase

The objective of DDAG is to reduce the classification time compared to *One-against-One* technique. To predict the class label of a given example x, starting at the root node then moves to the left or right node based on the binary decision of the classifier at this node. This process keeps on traversing the graph until it reaches a leaf, which the predicted class label. Thus, it requires only K-1 decision nodes to be evaluated in order to predict the unknown class. This classification approach is more efficient than the pairwise technique which requires the evaluation of all the $K(K-1)/2$ classifiers. The main drawback of DDAG is that it depends on the order of binary classes within the graph. This leads to difference in accuracy between different sequences. Ussivakul and Kijsirikul [192] proposed the Adaptive Directed Acyclic Graph (ADAG) method which is the modification of the DDAG. This method reduces the dependency of the sequence of nodes on the structure. In addition, the number of tests required to predict the correct class is reduced to $\log_2(K)$ times or less, considerably less than the number of tests required by DDAG which is linear with K. Their approach yields higher accuracy and reliability of classification, especially when the number of classes is relatively large.

4.6 Tree-Structured (Hierarchical) Approach

The aim of the tree-structured approach is to provide an encoding strategy for the design of the ECOC matrix that takes into account the relationships and similarities among classes and to achieve high classification accuracy using the minimum

number of classifiers. The task of the tree-structured approach is to decompose a given K-class problem into a set of simpler tree-structured K-1 binary problems and to train classifiers to solve the binary problems at the internal nodes within the tree through a base learning algorithm ($BaseLearn$). In the classification phase, the approach uses a given combination method ($TreeCombiner$) to combine the intermediate results of the internal node classifiers in order to produce the final decision of the ensemble for a given unseen instance x. The approach works as follows: First, the set of K classes (Ω) is split into two disjoint subsets, known as meta-classes or super-classes. Then these meta-classes are again split recursively until each meta-class contains one of the original classes. The resultant binary tree has K leaf nodes, one for each original class and K-1 internal nodes, each associated with two meta-classes and a binary classifier. (See Algorithm 4)

Algorithm 4 Tree Ensemble Learning Algorithm

Require: L - set of m labeled training examples
$\Omega = \{\omega_1, \ldots, \omega_K\}$- set of classes
$BaseLearn$ - base learning algorithm
$TreeCombiner$ - hierarchical combination method
Training Phase
1: $\Omega_1 \leftarrow \Omega$
2: Generate Class Hierarchy as follows:

1. $C \leftarrow \{(c_k, \omega_k)\}_{k=1}^{K} \leftarrow GetClassCentroids(L)$
2. $hierarchy \leftarrow BuildNode(\Omega_1, C)$

3: **for** each internal node j at $hierarchy$ **do**
4: Filter and relabel the training set L as follows:
$L_j = \{(x,t)|(x,y) \in L$ and $t = 0$ if $y \in \Omega_{2j}$ and $t = 1$ if $y \in \Omega_{2j+1}\}$
5: Train binary classifier, $h_j \leftarrow BaseLearn(L_j)$
6: **end for**
Prediction Phase
7: **return** $TreeCombiner(x, hierarchy)$ for a given instance x

4.6.1 Training Phase

4.6.1.1 Generate Class Hierarchy

There are various ways to build the tree structure, e.g. user-defined and class-similarity based approaches. In the handwritten digits recognition problem for instance, the user might construct two meta-classes by separating the digits $\{0, 1, 2, 3, 4\}$ in one meta-class and the rest in the other meta-class. If the class

hierarchy is based on the relationships among classes, it provides important domain knowledge that might lead to improve the classification accuracy [105, 132]. That is, the class hierarchy should satisfy the well-known cluster assumption: similar classes should belong to the same meta-class while dissimilar classes should belong to different meta-classes. There are two approaches to exploit the similarity among classes: the bottom-up approach defined in Algorithm 5 and the top-down approach defined in Algorithm 6. The resultant binary tree has K leaf nodes, one for each original class and K-1 internal nodes, each associated with two (meta-)classes and a binary classifier. There is a number of various ways to measure the distance between two classes such as Nearest Neighbor (Single linkage), Farthest Neighbor, Average Distance and Centroid. In this study, the Euclidean distance between the centroid of the training examples that belong to ω_i and that of the examples belonging to class ω_k is used to measure the similarity between them (see Figure 4.3). In the bottom-up approach, the multi-view hierarchical clustering algorithm was proposed by Gupta and Dasgupta [74].

Algorithm 5 *BuildNode* - (Bottom-Up Approach)

Require: Ω_j - set of classes assigned to tree node j
C_j - set of centroids of classes in metaclass Ω_j
if $|\Omega_j| = 1$ **then**
  Add a leaf node j to *hierarchy* that represents class Ω_j
else
  Add an internal node j to *hierarchy* that represents meta-class Ω_j
  Initially, put each class in Ω_j in a separate cluster
  repeat
    Get the two most close clusters in Ω_j
    Merge these two clusters into a new cluster
  until the number of remaining clusters is two
  Denote the remaining clusters, Ω_{2j} and Ω_{2j+1}
  $C_{2j} \leftarrow$ set of centroids of classes in Ω_{2j}
  $BuildNode(\Omega_{2j}, C_{2j})$
  $C_{2j+1} \leftarrow$ set of centroids of classes in Ω_{2j+1}
  $BuildNode(\Omega_{2j+1}, C_{2j+1})$
end if
return *hierarchy*

In the top-down approach, the tree structure is generated by recursively applying *k-means* clustering algorithm at each node j to split its associated set of classes Ω_j into two disjoint subsets Ω_{2j} and Ω_{2j+1}, until every subset contains exactly one class. Starting from the root node recursively for each internal node j, its set of classes Ω_j is divided into two disjoint (dissimilar) subsets Ω_{2j} and Ω_{2j+1}. For instance, at the root node with index 1 (see Figure 4.2), the most distant subsets Ω_2 and Ω_3 of Ω_1 are determined by performing 2-means clustering using

Algorithm 6 *BuildNode* - (Top-Down Approach)

Require: Ω_j - set of classes assigned to tree node j
C_j - set of centroids of classes in metaclass Ω_j
if $|\Omega_j| = 1$ **then**
  create a leaf node j that represents the class Ω_j
  Add $node_j$ to *hierarchy*
else
  create an internal node j that represents the metaclass Ω_j
  Add $node_j$ to *hierarchy*
  Get the most distant classes in Ω_j: $(c_{j1}, \omega_{j1}), (c_{j2}, \omega_{j2})$
  $\{\Omega_{2j}, \Omega_{2j+1}\} = seeded\text{-}k\text{-}means(C_j, c_{j1}, c_{j2})$
  $C_{2j} \leftarrow$ set of centroids of classes in Ω_{2j}
  $BuildNode(\Omega_{2j}, C_{2j})$
  $C_{2j+1} \leftarrow$ set of centroids of classes in Ω_{2j+1}
  $BuildNode(\Omega_{2j+1}, C_{2j+1})$
end if
return *hierarchy*

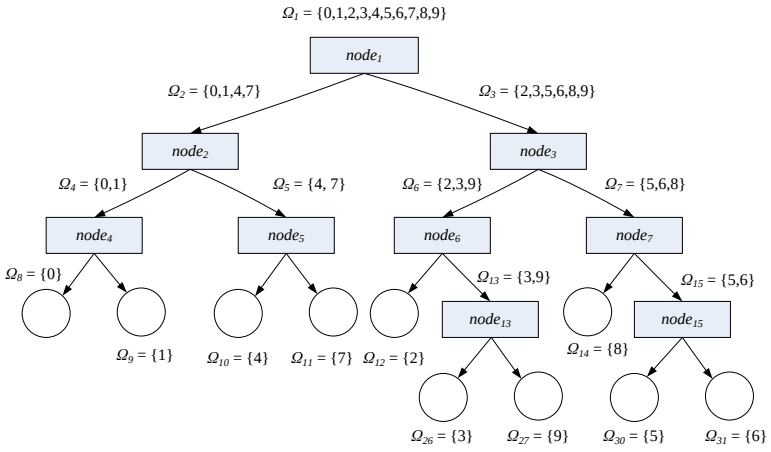


Figure 4.2: Class hierarchy constructed using Top-Down approach for the handwritten digits

the centroids of the most distant classes in Ω_1 as initial prototypes for clusters. The meta-classes Ω_2 and Ω_3 will contain the set of classes lies at the first and second cluster, respectively.

4.6.1.2 Train Binary Classifiers

After constructing the tree, a binary classifier h_j is assigned to each internal node j to discriminate between two meta-classes Ω_{2j} and Ω_{2j+1}. It is trained using the training examples in L that belong to meta-class Ω_{2j+1} as positive examples

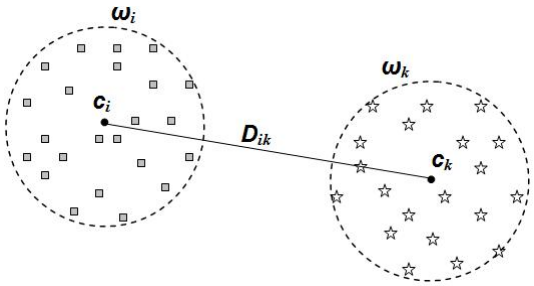


Figure 4.3: Distance between class ω_i and class ω_k according to Centroid-based distance calculation method

(labeled 1), these examples belonging to Ω_{2j} as negative examples (labeled -1) and the other examples are not taken into consideration (labeled 0), see Table 4.4. For instance, the digits recognition task is solved by construction of 9 binary classifiers.

Jun and Ghosh [91] proposed a novel multi-class boosting algorithm, called *AdaBoost.BHC*. First the tree-structured approach is used to decompose the multi-class problem into a set of binary problems. Then instead of a single binary classifier, an ensemble of binary classifiers is constructed, by the popular *AdaBoost* ensemble method, to solve each binary problem. Empirical comparisons of *AdaBoost.BHC* and other existing variants of multi-class *AdaBoost* algorithm are carried out using seven multi-class datasets from the UCI machine learning repository. Not only *AdaBoost.BHC* is faster than other *AdaBoost* variants but also it achieves lower error rates.

Table 4.4: Decomposition of the 10-class handwritten digits problem into 9 binary classification problems using the *Tree-Structured Approach*

	h_1	h_2	h_3	h_4	h_5	h_6	h_7	h_8	h_9
*digit*0	-1	-1	0	-1	0	0	0	0	0
*digit*1	-1	-1	0	1	0	0	0	0	0
*digit*2	1	0	-1	0	0	-1	0	0	0
*digit*3	1	0	-1	0	0	1	0	-1	0
*digit*4	-1	1	0	0	-1	0	0	0	0
*digit*5	1	0	1	0	0	0	1	0	-1
*digit*6	1	0	1	0	0	0	1	0	1
*digit*7	-1	1	0	0	1	0	0	0	0
*digit*8	1	0	1	0	0	0	-1	0	0
*digit*9	1	0	-1	0	0	1	0	1	0

4.6.2 Classification Phase

In the literature, there are various methods developed to combine the intermediate decisions of the binary classifiers. In the following subsections, a hard and two soft combiners will be discussed. In the second part of this thesis [3], a new soft

trainable combiner based on decision templates and RBF neural networks will be introduced.

4.6.2.1 Classical Decision Tree-Like (Hard) Combiner

A simple and fast method to get the final decision of the tree is the decision tree approach where the tree is traversed following the individual node classifiers h_j starting from the root node to a leaf node which is then representing the classification result. Each classifier decides which of its child node has to be evaluated next, until a leaf node is reached. This combination method is very fast. Drawbacks of this approach are: (1) misclassification of high-level classifiers can not be corrected; (2) it does not benefit from the discriminating information, provided by the classifiers, outside the path; (3) the output is only the predicted class label, that is $h_j : \mathbb{R}^D \rightarrow \{\Omega_{2j}, \Omega_{2j+1}\}$, and therefore voting is the only way to combine the ensemble of tree classifiers.

4.6.2.2 Product of the Unique Path Combiner

This tree combination method was proposed by Kumar et al. and applied for land cover classification using remote sensing hyperspectral data in [105, 132]. It is based on the assumption that the internal classifier h_j can estimate meta-class membership probabilities (soft classifier). Then, for given instance x, the membership probability for each class k is the product of the posterior probabilities of all the internal classifiers along the unique path from the root node to the leaf node containing class k.

4.6.2.3 Dempster-Shafer evidence theory

It is a generalization to traditional probability theory for the mathematical representation of uncertainty, which was introduced by Dempster [50] and Shafer [174]. There are many reasons for selecting this theory in the context of multiple classifiers combination. It can discriminate between ignorance and uncertainty. Since it is able to assign evidence not only to atomic but also to subsets and intervals of hypotheses, it easily represents evidences at different levels of abstraction. It can combine evidences provided by different sources and can measure the conflict among them.

Let Θ be a finite set of K mutually exclusive atomic hypotheses $\theta_1, \ldots, \theta_K$, called the *frame of discernment* and let 2^Θ denote the set of all subsets of Θ. Then a *basic belief assignment* (*BBA*) or *mass function* is defined over Θ as a function $m : 2^\Theta \rightarrow [0, 1]$ that satisfies the following conditions:

$$m(\emptyset) = 0 \quad \text{and} \quad \sum_{A \subseteq \Theta} m(A) = 1 \tag{4.9}$$

The quantity $m(A)$ can be interpreted as the belief in a hypothesis A. Differently from probability, $m(A)$ is not the sum of masses given to the elements of A. It represents the mass given to A itself and not to any of its subsets. A situation of total ignorance means that nothing is known but the fact that the true value is in the universal set and is represented by $m(\Theta) = 1$. Intuitively, a part of belief in a hypothesis A must also be committed to any hypothesis it implies. To measure the total belief in A, one must add to $m(A)$ the masses $m(B)$ for all subsets B of A. This function is called a *belief function* or *credibility* of A:

$$Bel(A) = \sum_{B:B\subseteq A} m(B) \tag{4.10}$$

It is clear that $Bel(A) = m(A)$ if A is an atomic hypothesis. The subsets A of Θ where $m(A) > 0$ are called the *focal elements* of the belief function, and their union is called its core. One can verify that the belief in some hypothesis A and the belief in its negation $\bar{A}$ do not necessarily sum to 1, which is a major difference with probability theory and leads to discriminate between ignorance and uncertainty. Consequently, $Bel(A)$ does not expose to what extent one believes in $\bar{A}$, that is to what extent one doubts in A, which is described by $Bel(\bar{A})$. The quantity $Pl(A) = 1 - Bel(\bar{A})$, called the *plausibility* of A, defines to what extent one fails to doubt in A, that is to what extent one finds A plausible. It is defined as follows:

$$Pl(A) = \sum_{B:B\cap A\neq\emptyset} m(B) \tag{4.11}$$

As demonstrated by Shafer [174], any one of the three functions m, Bel and Pl is sufficient to recover the other two.

$$m(A) = \sum_{B:B\subseteq A} (-1)^{|A|-|B|} Bel(B). \tag{4.12}$$

Dempster's unnormalised rule of combination [171] is a convenient method to combine the *BBA*s provided by n independent sources of information $m_1, \ldots, m_n$ into a single *BBA* using the orthogonal sum defined below where $m(\Theta)$ indicates the degree of conflict among sources.

$$m(A) = \sum_{A_i:\cap A_i=A} \prod_{1\leq i\leq n} m_i(A_i) \tag{4.13}$$

An approach similar to decision templates is used in [158] to apply the DS theory for multiple classifier fusion. The distances between the classifier outputs for the example to be classified and the mean of classifier outputs calculated on the training examples are transformed into basic belief assignments that are then combined using the orthogonal sum.

4.6.2.4 Evidence-theoretic Soft Combiner

In [61, 60], a combiner based on DS evidence theory was proposed for decision fusion of internal nodes classifiers where it was applied successfully for visual object recognition tasks. Using the vocabulary of DS theory, Ω can be called the frame of discernment of the task where hypothesis θ_k means that "the given instance x_u belongs to class ω_k". In addition, each internal node classifier h_j is considered as a source of evidence providing that it is soft classifier ($h_j : \mathbb{R}^D \times \{\Omega_{2j}, \Omega_{2j+1}\} \rightarrow [0, 1]$). The final decision is a combination of knowledge extracted from different sources: (i) binary classifier and (ii) tree ensemble of K-1 binary classifiers.

- **Evidence from an individual node classifier**

  Consider an internal node j within a tree, let us define a local frame of discernment Θ_j:

  $$\Theta_j = \{\Theta_{2j}, \Theta_{2j+1}\} \tag{4.14}$$

  where hypothesis Θ_{2j} means that "the given instance x_u belongs to meta-class Ω_{2j} and Θ_{2j+1} means that "it belongs to meta-class Ω_{2j+1}".

  Since h_j is a source of evidence, it can be represented by a BBA m_j. Usually, not all classifiers produce outputs that satisfy the conditions of BBA in Eq.4.9. In this case, the outputs of classifier h_j are transformed into BBA as follows: (1) all negative values are set to zero, (2) if the sum of a classifier outputs is greater than one, it is normalized to sum up to one. if $h_j(x_u, \Omega_{2j})$ ($h_j(x_u, \Omega_{2j+1})$) is high, a high belief is assigned to hypothesis Θ_{2j} (Θ_{2j+1}).

  Discounting Technique is used to propagate the outputs of high-level classifiers to the classifiers at the lower levels. That is, the output of each internal node classifier h_j is multiplied by the BBA of its parent node classifier $m_{par(j)}$ where the root node classifier output is not discounted. The motivation for discounting is the fact that a number of classifiers will be enforced to classify to examples that actually belong to classes that are unknown to them. For instance, a classifier h_j that discriminates between $\Omega_{2j} = \{\omega_1, \omega_5\}$ and $\Omega_{2j+1} = \{\omega_2, \omega_6\}$ has to classify an example x_u belonging to class ω_3. In this case, it is desirable that $h_j(x_u, \Omega_{2j})$ and $h_j(x_u, \Omega_{2j+1})$ tends to zero but at the real situation, either of them may tend to one. If at least one classifier within a certain path gives a low response to instance x_u, this leads to weaken any undesirable high responses. Therefore, BBA m_j is defined as follows:

  $$m_j(\Theta_{2j}) = m_{par(j)}(A).h_j(x_u, \Omega_{2j}) \tag{4.15}$$

  $$m_j(\Theta_{2j+1}) = m_{par(j)}(A).h_j(x_u, \Omega_{2j+1}) \tag{4.16}$$

  $$m_j(\Theta) = 1 - m_j(\Theta_{2j}) - m_j(\Theta_{2j+1}) \tag{4.17}$$

  $$m_j(B) = 0 \;\; \forall B \in 2^{\Theta} - \{\Theta, \Theta_{2j}, \Theta_{2j+1}\} \tag{4.18}$$

where $A = \Theta_{2.par(j)}$ if $j = 2.par(j)$ (node j lies at the left subtree of its parent node) and similarly $A = \Theta_{2.par(j)+1}$) if $j = 2.par(j) + 1$. Note that $m_j(\Theta)$ represents the doubt in h_j.

- **Evidence from all K-1 node classifiers within tree** Following *Dempster's unnormalized rule of combination*, the *BBA*s from the K-1 internal node classifiers within a class hierarchy t are conjunctively combined in order to calculate the evidence about a hypothesis θ_k (degree of belief provided by TC_t that an example x_u belongs to ω_k).

$$\mu_k^{(t)}(x_u) = m^{(t)}(\theta_k) = \sum_{\cap A_j = \theta_k} \prod_{1 \le j \le K-1} m_j(A_j) \text{ where } A_j = \Theta_{2j},\ \Theta_{2j+1},\ or\ \Theta \tag{4.19}$$

and

$$m^{(t)}(\Theta) = \prod_{1 \le j \le K-1} m_j(\Theta) \tag{4.20}$$

where $m^{(t)}(\Theta)$ represent the conflict among the internal classifiers $h_1, \ldots, h_{K-1}$.

4.6.3 Related Work

A similar tree-structured decomposition approach is that in [105]. They proposed a hierarchical multiple classifier architecture, called BHC, for the analysis of hyperspectral data in multi-class problems but they do not consider semi-supervised learning. An algorithm using the generalized associative modular learning ($GAML$) paradigm was developed to divide a set of classes recursively into two meta-classes and simultaneously finding the best one dimensional projected feature space that discriminates the two meta-classes using an extension of Fisher's discriminant. The soft combiner in Section 4.6.2.2 is adopted. An experimental evaluation in [153] has shown that tree-structured approach performs comparably to $ECOC$ using fewer number of binary classifiers.

In [92], the tree-structured approach was compared with four multi-class decomposition techniques: One-against-Others (Section 4.2), One-against-One (Section 4.3), DDAG (Section 4.5) and ECOC (Section 4.4). Support Vector Machines were used as binary classifiers and the performance was evaluated on a number of visual object recognition learning tasks. The results have shown that the tree-structured approach performs comparable to the other techniques.

In the margin tree algorithm [182], a class hierarchy is constructed by hierarchical agglomerative clustering (HAC) where margins between pairs of classes are used as distance measures for clustering of (meta-)classes. There are three different ways to define the margin: greedy, complete-linkage and single-linkage. Then a total of K - 1 internal nodes will be created with K leaf nodes, same as in

BHC. As opposite to BHC, in the margin tree algorithm, it is assumed that the dimensionality is always greater than the number of samples, so that the samples are always linearly separable by a maximum-margin hyperplane. If the samples are not linearly separable, using non-linear kernels such as radial basis function to make the samples separable in a higher dimensional space leads to more difficult interpretation of margins, and makes the class hierarchy more sensitive to the kernel parameters.

In [90], they try to solve the problem of small sample size that occurs during the class hierarchy generation of BHC. It is worth mentioning that the lower the position of a node at the tree, the less sample size it will have for training. They proposed a hybrid approach that combine the merits of BHC framework and margin trees. That is, at each node they check the available sample size. If number of instances is less than the number of features, the margin tree algorithm is employed instead of BHC. While BHC algorithm is applied if the samples are not guaranteed to be linearly separable.

Note that in the above mentioned work, unlabeled data was not considered to boost the classification performance when the amount of the labeled data is limited. In the second part of this thesis [9], two new architectures, called *cotrain-of-trees* and *tree-of-cotrains*, are introduced in order to deal with the problem of small sample size. In addition, a novel ensemble method, denoted as *Multi-View Forest* is proposed [6, 1, 11], in the second part of this thesis, that exploits the error difference among individual tree-structured classifiers trained using different feature types to construct a more accurate forest classifier.

4.7 Conclusion

Multi-class decomposition techniques are ensemble methods that construct a set of binary classifiers to solve a multi-class classification task. Each technique consists of three stages: (1) decomposition of the multi-class problem into a set of simpler two-class problems, (2) solving these two-class problems and (3) combination of the intermediate solutions to yield the final decision. Ensemble methods can be divided into: *Flat* and *Hierarchical*. Flat architectures are the most popular ones where the members work independently disregarding the hierarchical structure of the classes. *Tree-Structured ensembles* improves the classification performance by taking into account prior knowledge encoded into the class hierarchy.

Chapter 5

Semi-Supervised Learning

5.1 Introduction

Supervised learning algorithms require a large amount of labeled training data in order to construct models with high prediction performance, see Figure 5.1. In many practical data mining applications such as computer-aided medical diagnosis [119], remote sensing image classification [175], speech recognition [95], email classification [96], or automated classification of text documents [139, 140], there is often an extremely inexpensive large pool of unlabeled data available. However, the data labeling process is often difficult, tedious, expensive, or time consuming, as it requires the efforts of human experts or special devices. Due to

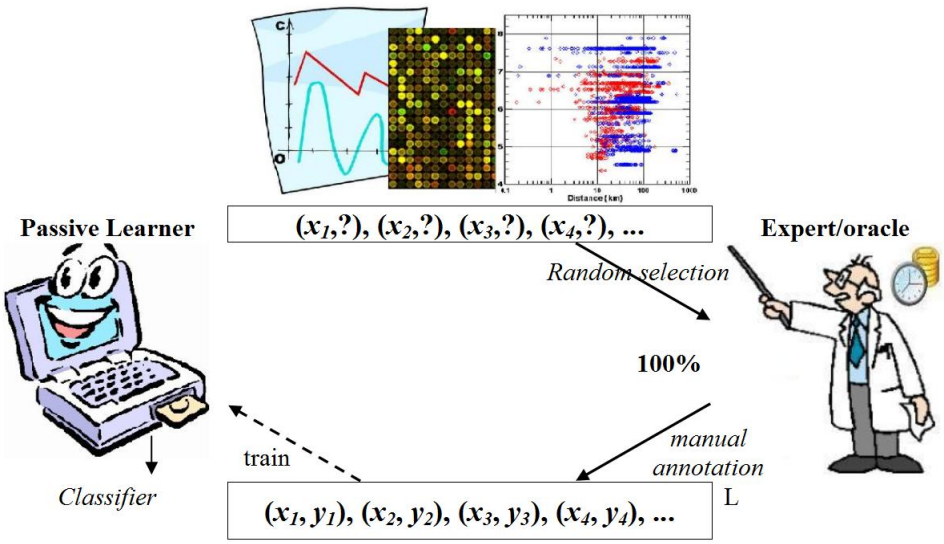


Figure 5.1: Graphical illustration of traditional supervised learning

the difficulties in incorporating unlabeled data directly into traditional supervised learning algorithms such as support vector machines and RBF neural networks

and the lack of a clear understanding of the value of unlabeled data in the learning process, the study of semi-supervised learning attracted attention only after the middle of 1990s. As the demand for automatic exploitation of unlabeled data increases, semi-supervised learning has become a hot topic.

In computer-aided diagnosis (CAD), mammography is a specific type of imaging that uses a low-dose x-ray system to examine breasts and it is used to aid in the early detection and diagnosis of breast diseases in women. There is a large number of mammographic images that can be obtained from routine examination but it is difficult to ask a physician or radiologist to search all images and highlight the abnormal areas of calcification that may indicate the presence of cancer. If we use supervised learning techniques to build a computer software to highlight these areas on the images, based on limited amount of diagnosed training images, it may be difficult to get an accurate diagnosis software. Then a question arises: can we exploit the abundant undiagnosed images [119] with the few diagnosed images to construct a more accurate software (see Figure 5.2).

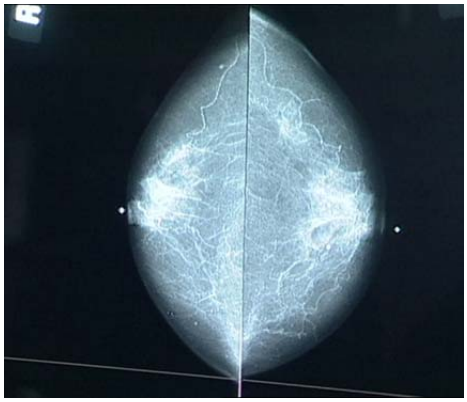


Figure 5.2: Computer-aided detection (CAD) mammogram

For remote sensing applications, the remote sensing sensors can produce data in large number of spectral bands. The objective of using such high resolution sensors is to discriminate among more ground cover classes and hence obtain a better understanding about the nature of the materials that cover the surface of the Earth. This large number of classes and large number of spectral bands require a large number of labeled training examples (pixels) from all the classes of interest. The class labels of such training examples are usually very expensive and time consuming to acquire [175]. The reason is that identifying the ground truth of the data must be gathered by visual inspection of the scene near the same time that the data is being taken, by using an experienced analyst based on their spectral responses, or by other means. In any case, usually only a limited number of training examples can be obtained. These training examples are often used for deciding which features are useful for the discrimination among classes, and for designing classifiers based on these derived features (see Figure 5.3). The

purpose of SSL is to study how to reduce the small sample size problems by using unlabeled data that may be available in large number and with no extra cost.

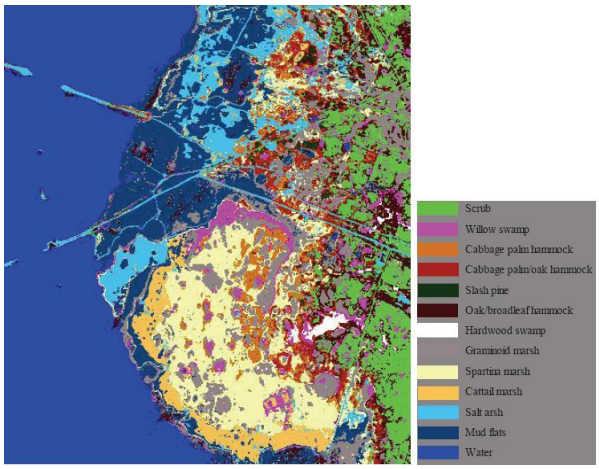


Figure 5.3: Remote-sensing image classification

Another important application for SSL is speech recognition. Speech recognition systems require large amount of transcribed data for parameter estimation. However, the manual transcription is tedious and expensive. Kemp and Waibel [95] trained an initial speech recognizer with only 30 minutes of transcriptions then an initial transcripts are generated with this recognizer for a large portion of 50 hours of untranscribed data. The experiments have shown that the word error rate on a broadcast news speech recognition task is reduced from 32% to 21.4% as a result of using the newly-transcripted materials.

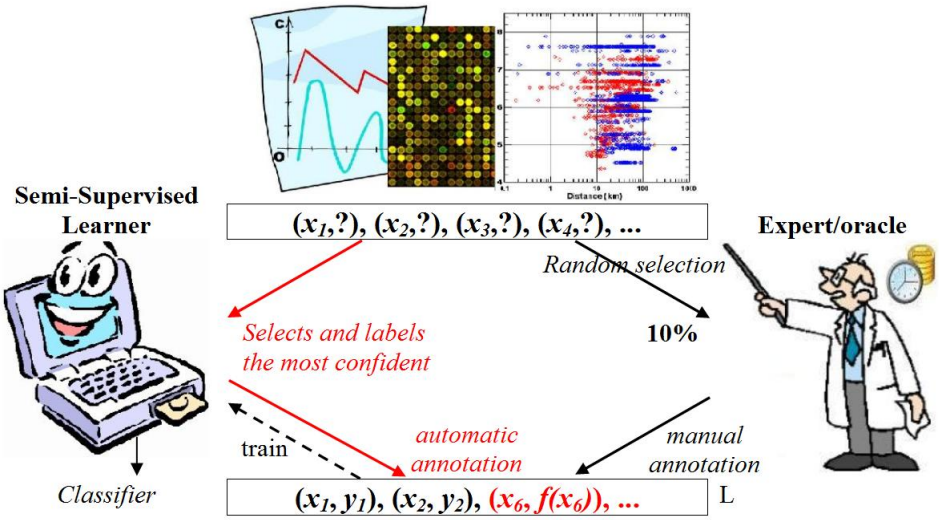


Figure 5.4: Graphical illustration of semi-supervised learning

5.2 What is Semi-Supervised Learning?

In the machine learning literature, there are mainly three paradigms for addressing the problem of combining labeled and unlabeled data to boost the performance: semi-supervised learning, transductive learning and active learning. *Semi-supervised learning* (*SSL*) refers to methods that attempt to take advantage of unlabeled data for supervised learning, see Figure 5.4, or to incorporate prior information such as class labels, pairwise constraints or cluster membership in the context of unsupervised learning. *Transductive learning* refers to methods which also attempt to exploit unlabeled examples but assuming that the unlabeled examples are exactly the test examples. *Active learning* [173] refers to methods which assume that the given learning algorithm has control on the selection of the input training data such that it can select the most important examples from a pool of unlabeled examples, then an oracle such as a human expert is asked for labeling these examples, where the aim is to minimize data utilization. *Active learning* will be discussed in more detail in the next section. The recent research of the machine learning community on semi-supervised learning (*SSL*) concentrates into four directions: semi-supervised classification [30, 140, 96, 210, 215, 119], semi-supervised regression [214], semi-supervised clustering such as constrained and seeded k-means clustering [195, 181, 19] and semi-supervised dimensionality reduction [20, 218]. Interested readers in recent advances of *SSL* are directed to the literature survey of Zhu [219]. Many semi-supervised classification algorithms have been developed. They can be divided into five categories according to [219]: (1) *Self-Training* [139], (2) semi-supervised learning with generative models [131, 140, 175], (3) S3VMs (Semi-Supervised Support Vector Machines) [88, 39, 73, 115], (4) semi-supervised learning with graphs [23, 209, 220], and (5) semi-supervised learning with committees (semi-supervised by disagreement) [30, 140, 96, 210, 215, 119, 213]. The remainder of this chapter provides an overview of these five categories.

5.3 Self-Training

Self-Training [139] is an incremental algorithm that initially builds a single classifier using a small amount of labeled data, see Figure 5.5. Then it iteratively predicts the labels of the unlabeled examples, rank the examples by confidence in their prediction and permenantly adds the *most confident examples* into the labeled training set. It retrains the underlying classifier with the augmented training set and the process is repeated for a given number of iterations or until some heuristic convergence criterion is satisfied. The classification accuracy can be improved over iterations only if the initial and subsequent classifiers correctly label most of the unlabeled examples. Unfortunately, adding mislabeling noise is not avoidable. In practical applications, more accurate confidence measures

and predefined confidence thresholds are used in order to limit the number of mislabeled examples.

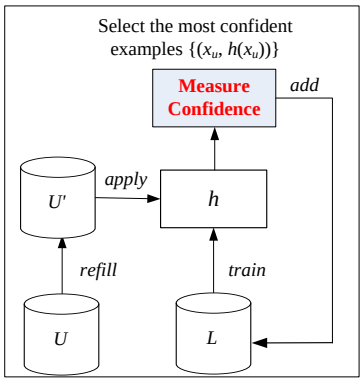


Figure 5.5: Graphical illustration of *Self-Training*

Self-Training is a wrapper algorithm that is applied on any learning algorithm. It has been appeared in the literature with several names: self-learning [170, 136], self-corrective recognition [136], naive labelling [86], and decision-directed [207]. One drawback when *Self-Training* is applied on linear classifiers such as *support vector machines* is the most confident examples often lie away from the target decision boundary (non informative examples). Therefore, in many cases this process does not create representative training sets as it selects non informative examples. Another drawback is that *Self-Training* is sensitive to outliers. For instance, compare between Figure 5.6 and Figure 5.7 when *Self-Training* is applied on *1-Nearest-Neighbor* classifier.

5.4 SSL with Generative Models

In generative approaches, it is assumed that both labeled and unlabeled examples come from the same parametric model where the number of components, prior $p(y)$, and conditional $p(x|y)$ are all known and correct. Once the model parameters are learned, unlabeled examples are classified using the mixture components associated to each class. Methods in this category such as in [140, 138] usually treat the class labels of the unlabeled data as missing values and employ the EM (Expectation-Maximization) algorithm [51] to conduct maximum likelihood estimation (MLE) of the model parameters θ. It begins with an initial model trained on the labeled examples. It then iteratively uses the current model to temporarily estimate the class labels of all the unlabeled examples and then maximizes the likelihood of the parameters (trains a new model) on all labeled examples (the original and the newly labeled) until it converges.

The methods differ from each other by the generative models used to fit the data, for example, mixture of Gaussian distributions (GMM) is used for image

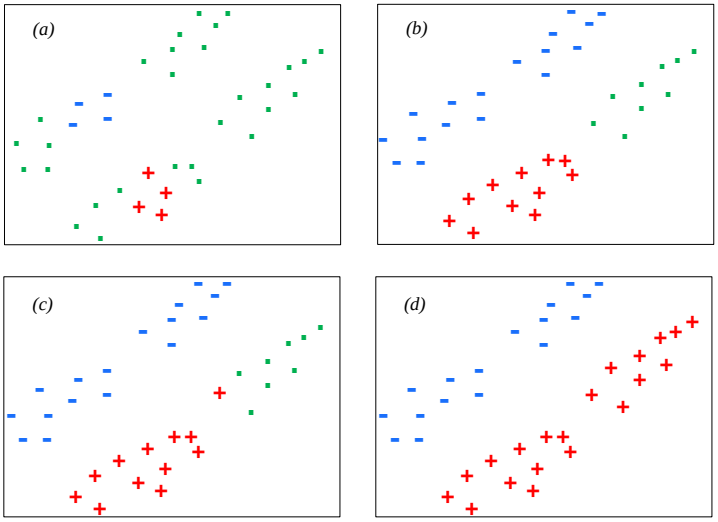


Figure 5.6: When *Self-Training* with *1-Nearest-Neighbor* classifier works

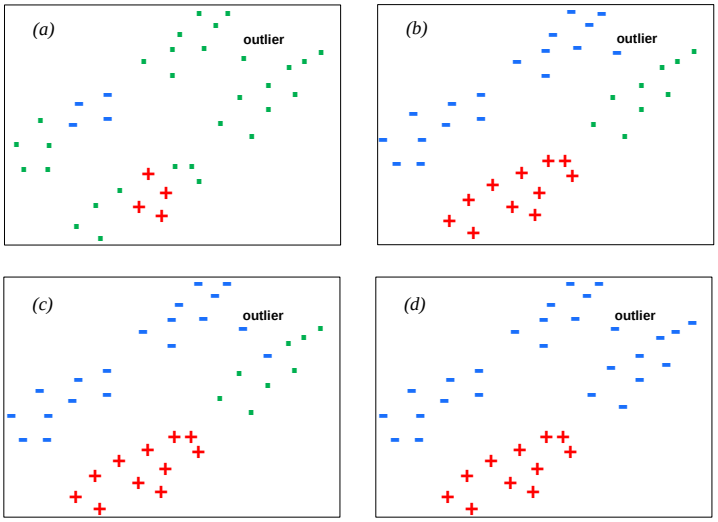


Figure 5.7: When *Self-Training* with *1-Nearest-Neighbor* classifier and a single outlier does not work

classification [175], mixture of multinomial distributions (Naive Bayes) [140, 138] is used for text categorization and Hidden Markov Models (HMM) [86] is used for speech recognition. Although the generative models are simple and easy to implement and may be more accurate than discriminative models when the number of labeled examples is a very small, the methods in this category suffer from a serious problem. That is, when the model assumption is incorrect, fitting the model using a large number of unlabeled data will result in performance degradation [45]. Thus, in order to alleviate the danger in real-world applications [219], one needs to carefully construct the generative model, for instance to construct more than one Gaussian per class. Also, one can down weight the unlabeled examples in the maximum likelihood estimation.

5.5 Semi-Supervised SVMs (*S3VMs*)

The aim of *S3VM*, sometimes called *Transductive SVM*, is to exploit the unlabeled data to adjust the decision boundary initially constructed from a small amount of labeled data, such that it goes through the low density regions while keeping the labeled examples correctly classified [88, 39], see Figure 5.8. It is an extension of the standard support vector machines. In the standard SVM, only the labeled data is used while in S3VMs the unlabeled data is also used. It firstly constructs an initial SVM classifier using labeled examples and predict the labels of the unlabeled examples. Then, it iteratively maximizes the margin over both labeled and the (newly labeled) unlabeled examples. The optimal decision boundary is the one that has the minimum generalization error on the unlabeled data. *S3VM* assumes that unlabeled data from different classes are separated with large margin. In addition, it assumes there is a low density region through which the linear separating hyperplane passes. Thus, it does not work for some domains in which this assumption is not fulfilled and a generative approach would be more suited.

5.6 Semi-Supervised Learning with Graphs

Blum and Chawla [28] proposed the first graph-based semi-supervised learning method. They constructed a graph whose nodes represent both labeled and unlabeled training examples and the edges between nodes weighted according to the similarity between the corresponding examples. Based on the graph, the aim is to find the minimum cut of the graph such that nodes in each connected component have the same label. Later, Blum et al. [29] added random noise to the edge weights and the labels of the unlabeled examples are predicted using majority voting. The procedure is similar to bagging and produces a soft minimum cut. Note that in both [28] and [29] a discrete predictive function is used that assigns

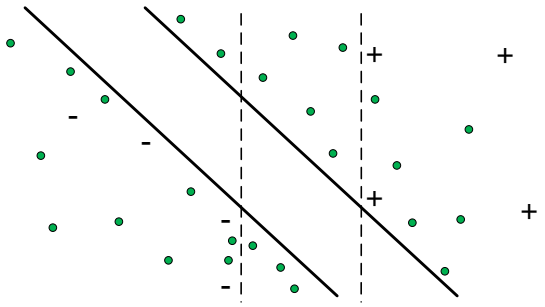
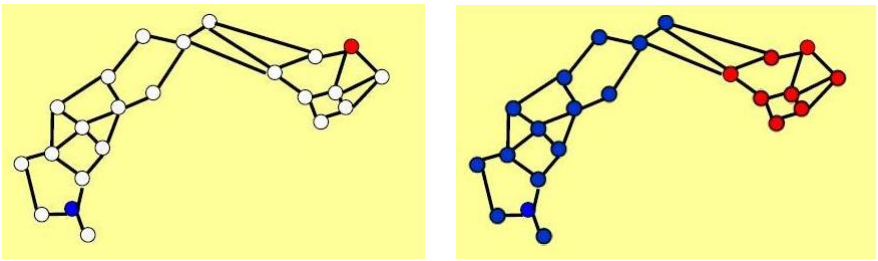


Figure 5.8: Graphical illustration of S3VMs: The unlabeled examples help to put the decision boundary in low density regions. Using labeled data only, the maximum margin separating hyperplane is plotted with the versicle dashed lines. Using both labeled and unlabeled data (dots), the maximum margin separating hyperplane is plotted with the oblique solid lines.

one of the possible labels to each unlabeled example. Zhu et al. [220] introduced a continuous prediction function. They modeled the distribution of the prediction function over the graph with Gaussian random fields and analytically showed that the prediction function with the lowest energy should have the harmonic property. They designed a label propagation strategy over the graph using such a harmonic property where the labels propagate from the labeled nodes to the unlabeled ones, see Figure 5.9. It is worth noting that all graph-based methods assume that examples connected by heavy edges tend to have the same class label and vice versa [219].



(a) Before SSL (b) After SSL

Figure 5.9: Graphical illustration of label propagation.

5.7 Semi-Supervised Learning with Committees

This section details some of the state-of-the-art algorithms that belong to the family of semi-supervised learning with committees or sometimes called semi-supervised learning by disagreement [213].

5.7.1 Multi-View Learning

Multi-view learning is based on the assumption that the instance input space $X = X_1 \times X_2$, where $X_1 \subset \mathbb{R}^{D_1}$ and $X_2 \subset \mathbb{R}^{D_2}$ represent two different descriptions of an instance, called views. These views are obtained through different physical sources and sensors or are derived by different feature extraction procedures and are giving different types of discriminating information about the instance. For instance, a web page can be represented by different views, e.g. the distribution of words used in the web page itself, the distribution of words that appear in the hyperlinks that point to this page, and any other statistical information, such as size, number of accesses, etc.

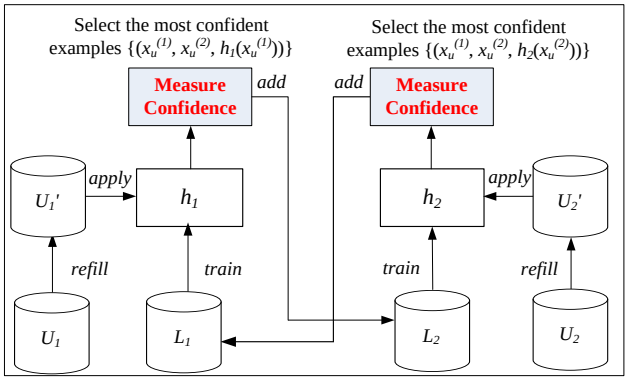
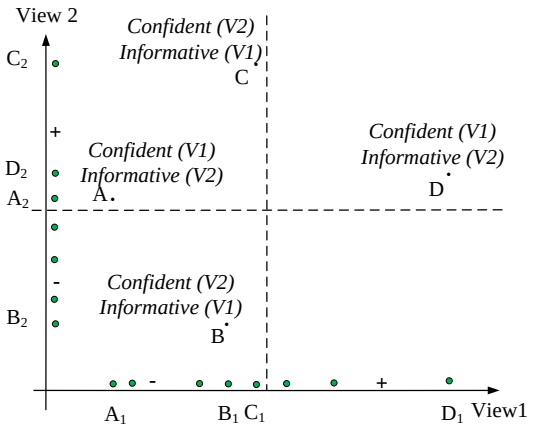


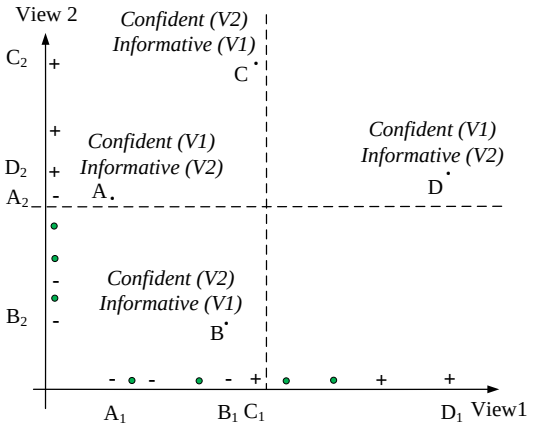
Figure 5.10: Graphical illustration of *Co-Training*

5.7.1.1 Multi-View Co-Training

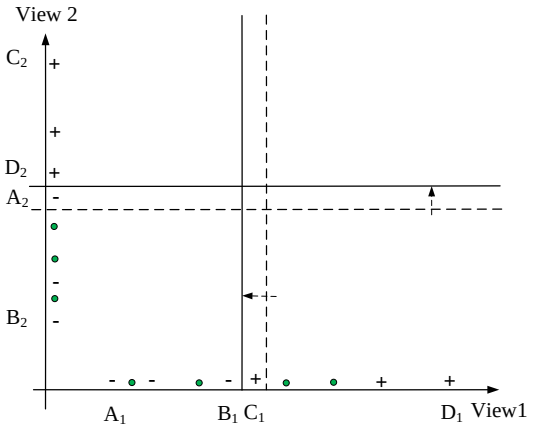
Multi-view learning was first introduced for semi-supervised learning by Blum and Mitchell in the context of *Co-Training* [30]. Blum and Mitchell state two strong requirements for successful *Co-Training*: the two sets of features should be conditionally independent given the class and each of which is sufficient for learning. The pseudo-code is shown in Algorithm 7 (see Figure 5.10) and an illustrative example is shown in Figure 5.11 where each view is a single feature. At the initial iteration, two classifiers are trained using a small amount of labeled training data. Then at each further iteration, each classifier predicts the class label of the unlabeled examples, estimates the confidence in its prediction, ranks the examples by confidence, adds the examples about which it is *most confident* into the labeled training set. The aim is that the *most confident* examples with respect to one classifier can be *informative* with respect to the other. An example is informative with respect to a classifier if it carries a new discriminating information. That is, it lies close to the decision boundary and thus adding it to the training set can improve the classification performance of this classifier. Nigam and Ghani [139] showed that *Co-Training* is sensitive to the view independence requirement.



(a) Find the most confident examples for each view: Examples A and D for view 1 and Examples B and C for view 2



(b) Label and add the most confident examples into the training set



(c) The decision boundaries shift with newly labeled data because example A is informative for view 2 and example C is informative for view 1

Figure 5.11: When *Co-Training* with two linear classifiers works

Algorithm 7 Pseudo code of *Standard Co-Training*

Require: set of labeled training examples (L), set of unlabeled training examples (U), maximum number of iterations (T),base learning algorithm ($BaseLearn$), two feature sets (views) representing an example (V_1,V_2), sample size (n), number of unlabeled examples in the pool (u) and number of classes (C)

Training Phase

1: Get the class prior probabilities, $\{Pr_c\}_{c=1}^{C}$
2: Set the class growth rate, $n_c = n \times Pr_c$ where $c = 1, \ldots, C$
3: Train initial classifiers $h_1^{(0)}$ and $h_2^{(0)}$ on the initial L
$h_1^{(0)} = BaseLearn(V_1(L))$ and $h_2^{(0)} = BaseLearn(V_2(L))$
4: **for** $t \in \{1, \ldots, T\}$ **do**
5: **if** U is empty **then**
6: $T \leftarrow t$-1 and abort loop
7: **end if**
8: **for** $v \in \{1, 2\}$ **do**
9: Apply $h_v^{(t-1)}$ on U.
10: Select a subset S_v as follows: for each class ω_c, select the n_c most confident examples assigned to class ω_c
11: Move S_v from U to L
12: **end for**
13: Re-train classifiers $h_1^{(t)}$ and $h_2^{(t)}$ on the new L
$h_1^{(t)} = BaseLearn(V_1(L))$ and $h_2^{(t)} = BaseLearn(V_2(L))$
14: **end for**

Prediction Phase

15: **return** combination of the predictions of $h_1^{(T)}$ and $h_2^{(T)}$

5.7.1.2 Co-EM

Nigam and Ghani [139] proposed another multi-view semi-supervised algorithm, called *Co-EM*. It uses the model learned in one view to probabilistically label the unlabeled examples in the other model. Intuitively, *Co-EM* runs *EM* (Section 5.4) in each view and before each new *EM* iteration, inter-changes the probabilistic labels predicted in each view. *Co-EM* is considered as a probabilistic variant of *Co-Training*. Both algorithms are based on the same idea: they use the knowledge acquired in one view, in the form of soft class labels for the unlabeled examples, to train the other view. The major difference between the two algorithms is that *Co-EM* does not commit to the labels predicted in the previous iteration because it uses probabilistic labels that may change from one iteration to the other. On the other hand, Co-Training commits to the most confident predictions that are once added into the training set are never revisited. Thus, it may add to the training set a large number of mislabeled examples.

5.7.2 Co-Training with Natural Views

The standard *Co-Training* was applied in domains with truly independent feature splits satisfying its conditions. In [96], Kiritchenko et al. applied *Co-Training* for email classification where the bags of words that represent email messages were split into two sets: the words from headers (V_1) and the words from bodies (V_2). Abdel Hady et al. [7] have combined *Co-Training* with tree-structured classifiers for multi-class decomposition. A combination method based on Dempster-Schafer evidence theory provides class probability estimates that were used to measure confidence on prediction. The approach was applied for visual object recognition where one tree classifier is based on color histograms (V_1) while the second one used orientation histograms (V_2) extracted from 2D images. Levin et al. [116] have used *Co-Training* to improve visual detector for cars in traffic surveillance video where one classifier detects cars in the original gray level images (V_1). The second one uses images where the background has been removed (V_2).

Although there are some cases in which there are two or more independent and redundant views, there exist many real-world applications in which multiple views are not available or it is computationally inefficient to extract more than one feature set for each example. There are three directions to apply *Co-Training* without natural feature splits, as shown in the following subsections.

5.7.3 Co-Training with Random Views

In some work, *Co-Training* was applied in domains without natural feature splits through splitting the available feature set into two views V_1 and V_2. Nigam and Ghani [139] investigated the influence of the views independence. They found that *Co-Training* works better on truly independent views than on random views. Also, *Co-Training* was found to outperform EM (see Section 5.4) when the views are truly independent. It was also shown that if there is sufficient redundancy in data, the performance of *Co-Training* with random splits is comparable to *Co-Training* with a natural split. There is no guarantee that random splitting will produce independent views.

5.7.4 Co-Training with Artificial Views

In a real-world application of *Co-Training*, the traditional feature subset selection algorithms based on mutual information and correlation cannot be used because they take into account the class information which is not available for the unlabeled examples. Feger and Koprinska [62] introduced a method, called *maxInd*, for splitting the feature set into two views. The aim is to minimize the dependence between the two feature subsets (*inter-dependence*), measured by conditional mutual information *CondMI*. The result is represented as an undirected graph, with features as nodes and the *CondMI* between each pair of features as weight on the

edge between them. In the second step the graph is cut into two disjoint parts of the same size. This split is performed in such a way that minimizes the sum of the cut edges in order to minimize the dependence between the two parts of the graph. They had found that *maxInd* does not outperform the random splits. A possible explanation from their perspective is that *Co-Training* is sensitive to the dependence of the features within each view (*intra-dependence*). The random split leads to *intra-dependence* lower than that of *maxInd* and the truly independent split. Their study states that there is a trade-off between the *intra-dependence* of each view, and the *inter-dependence* between the views. That is minimizing the *inter-dependence* leads to maximizing the *intra-dependence* of each view. In addition, the measurement of *CondMI* is not accurate enough because it is based on only a small number of labeled examples.

Salaheldin and El Gayar [162] introduced three new criteria for splitting features in *Co-Training* and compare them to existing artificial splits and natural split. The first feature split criterion is based on maximizing the confidence of the views. The second criterion maximizes both confidence and independence of the views. The independence of a view is measured by conditional mutual information as in [62]. For each view, a classifier is trained using the labeled data; it is then used to predict the class of the unlabeled data. The entropy of the classifier output for each input example is calculated and the average of entropies indicates the confidence of the view. They showed that splitting the features with a mixed criterion is better than using each criterion alone. Finally, they proposed a third criterion based on maximizing the views diversity. A genetic algorithm is used to optimize the fitness functions based on the three proposed criteria. The experimental results on two data sets show that the proposed splits are promising alternatives to random splitting.

5.7.5 Co-Training with Single View

In a number of recent studies [71, 210, 215, 119], the applicability of *Co-Training* using a single view without feature splitting has been investigated. The interested reader for a more extensive overview might refer to Roli's invited talk in MCS 2005 [159] and Zhou's invited talk in MCS 2009 [211].

5.7.5.1 Statistical Co-learning

Goldman and Zhou [71] first presented a single-view SSL method, called *Statistical Co-learning*. It used two different supervised learning algorithms with the assumption that each of them produce a hypothesis that partition the input space into a set of equivalence classes. For example, a decision tree partitions the input space with one equivalence class per leaf. They used 10-fold cross validation:(1) to select the most confident examples to label at each iteration and (2) to combine the two hypotheses producing the final decision. Its drawbacks are: first

the assumptions concering the used algorithms limits its applicability. Second the amount of available labeled data was insufficient for applying cross validation which is time-consuming.

5.7.5.2 Democratic Co-learning

Zhou and Goldman [210] then presented another single view method, called *Democratic Co-learning* which is applied to three or more supervised learning algorithms and reduce the need for statistical tests. Therefore, it resolves the drawbacks of *Statistical Co-learning* but it still uses the time-consuming cross-validation technique to measure confidence intervals. These confidence intervals are used to select the most confident unlabeled examples and to combine the hypotheses decisions.

5.7.5.3 Tri-Training

Zhou and Li [215] present a new *Co-Training* style *SSL* method, called *Tri-Training*, where three classifiers are initially trained on bootstrap subsamples generated from the original labeled training set. These classifiers are then refined during the *Tri-Training* process, and the final hypothesis is produced via majority voting. The construction of the initial classifiers looks like training an ensemble from the labeled data with *Bagging* [31]. At each *Tri-Training* iteration, an unlabeled example is added to the training set of a classifier if the other two classifiers agree on their prediction under certain conditions. *Tri-Training* is more applicable than previous *Co-Training*-Style algorithms because it neither requires multiple views as in [30, 139] nor does it depend on different supervised learning algorithms as in [71, 210]. There are two limitations: the ensemble size is limited to three classifiers and *Bagging* is used only at the initial iteration. Although the results have shown that using bagged ensemble of three classifiers can improve the generalization ability, better performance is expected when larger-size ensembles and other ensemble learners are used.

5.7.5.4 Co-Forest

Li and Zhou [119] proposed an extension to *Tri-Training*, called *Co-Forest*, in which an initial ensemble of random trees is trained on bootstrap subsamples generated from the given labeled data set L. To select new training examples from a given unlabeled data set U for each ensemble member h_i ($i = 1, \ldots, N$), a new ensemble H_i, called the concomitant ensemble of h_i, is defined that contains all the classifiers except h_i. At each iteration t and for each ensemble member h_i, first the error rate of H_i, $\hat{\epsilon}_{i,t}$, is estimated. If $\hat{\epsilon}_{i,t}$ is less than $\hat{\epsilon}_{i,t-1}$ (1^{th} condition), H_i predicts the class label of the unlabeled examples in $U'_{i,t}$ (random subsample of U of size $\frac{\hat{\epsilon}_{i,t-1} W_{i,t-1}}{\hat{\epsilon}_{i,t}}$). A set $L'_{i,t}$ is defined that contains the unlabeled examples

in $U'_{i,t}$ where the confidence of H_i about their prediction exceeds a predefined threshold (θ) and $W_{i,t}$ is the sum of the confidences of the examples in $L'_{i,t}$. If $W_{i,t}$ is greater than $W_{i,t-1}$ (2^{nd} condition) and $\hat{\epsilon}_{i,t}W_{i,t}$ is less than $\hat{\epsilon}_{i,t-1}W_{i,t-1}$ (3^{rd} condition), the i^{th} random tree will be re-trained using the original labeled data set L and $L'_{i,t}$. Note that the bootstrap sample used to train the i^{th} random tree at iteration 0 is discarded and $L'_{i,t}$ is not added *permenantly* into L. The algorithm will stop if there is no classifier h_i satisfying the three conditions.

I have the following comments on *Co-Forest*: First, the error rate $\hat{\epsilon}_{i,t}$ is estimated accurately only at the first iteration based on the *out-of-bag error* estimation, afterward the estimation tends to be an under-estimate as it depends on the training set. Therefore, *Co-Forest* will stop when the training error of a classifier reaches zero, for instance this is always true for the *1-nearest neighbor* classifier.

Second, setting the value of θ is not straightforward especially for multi-class problems where the confidence of the concomitant ensemble H_i is distributed among many classes. If θ is high, the 2^{nd} condition will not be fulfilled and the algorithm will stop. If θ is low, the size of $L'_{i,t}$ might be large and even equal to $U'_{i,t}$ which increases the risk that h_i will receive a lot of mislabeled examples.

Third, *Co-Forest* works in batch mode in opposite to other *Co-Training* style algorithms that work in incremental mode. That is, it evaluated the set of examples $L'_{i,t}$ as a whole, if 2^{nd} condition is fulfilled, the set $L'_{i,t}$ is used for training, otherwise, it is discarded although some examples in $L'_{i,t}$ may be beneficial. Fourth, *Co-Forest* does not take into account the class probabilities estimated by ensemble members although they can help in estimating labeling confidence.

5.7.6 Other Committee-Based SSL Algorithms

5.7.6.1 SSMBoost

d'Alché et al. [47] generalized *MarginBoost* to semi-supervised classification. *MarginBoost* is a variant of AdaBoost (Section 3.4.1.2) based on the minimization of an explicit cost function. Such function is defined for any scalar decreasing function of the margin. As the usual definition of margin cannot be used for unlabeled data, the authors extend the margin notion to unlabeled data. In practice, the margin is estimated using the *MarginBoost* classification output. Then, they reformulate the cost function of *MarginBoost* to include both the labeled and unlabeled data. A generative model is used as a base classifier and the unlabeled data is used by EM algorithms 5.4. The results have shown that *SSMBoost* outperforms the classical AdaBoost when a few amount of labeled data is available (only 5% of the training data is labeled).

5.7.6.2 ASSEMBLE

Bennet et al. [24] proposed another committee-based SSL method, called *ASSEMBLE*, which iteratively constructs ensemble classifiers using both labeled

and unlabeled data. The aim of *ASSEMBLE* is to overcome some limitations of *SSMBoost*. For example, while *SSMBoost* requires the base classifier to be a generative mixture model in order to apply EM for semi-supervision, *ASSEMBLE* is more general that can be used with any cost-sensitive base learning algorithm. At each iteration of *ASSEMBLE*, the unlabeled examples are assigning pseudo-classes using the current ensemble before constructing the next base classifier using both the labeled and newly-labeled examples. The experiments show that *ASSEMBLE* works well and it won the NIPS 2001 unlabeled data competition using decision trees as base classifiers.

5.7.6.3 DECORATE

Melville and Mooney [127] introduced an ensemble method, called *DECORATE*, which was designed to artificially generate new examples and add them to the training data in order to increase the diversity among the members of the created ensembles. An ensemble is constructed iteratively as follows: the ensemble is initialized with a single classifier trained on the original training data. At each further iteration, a new classifier is trained on the union of the original training data and the diversity data. The diversity data represents a specified number of artificial training examples generated based on a simple model of the data distribution. Actually it is inspired by the stream-based approach used for sample selection, see Section 6.2. The next step is to label these artificial generated examples. The class labels of these examples are chosen so as to differ maximally from the prediction of the current committee. Note that the new classifier is added to the current ensemble only if adding it will not increase the ensemble training error, otherwise it is discarded.

5.8 Conclusion

The work in [30, 18] has theoretically studied *Co-Training* with two views, but could not explain why the single-view variants can work. Wang and Zhou [197] provided a theoretical analysis that emphasizes that the important factor for the success of disagreement-based single-view *Co-Training style* algorithms is the creation of a large diversity (disagreement) among the co-trained classifiers, regardless of the method used to create diversity, for instance through: sufficiently redundant and independent views as in standard *Co-Training* [30, 139], artificial feature splits in [62, 162], different supervised learning algorithms as in [71, 210], training set manipulation as in [24, 215], different parameters of the same supervised learning algorithms [214] or feature set manipulation as in [119] and the proposed framework in the second part of this thesis [5, 4].

Note that Brown et al. presented in [36] an extensive survey of the various techniques used for creating diverse ensembles, and categorized them, forming

a preliminary taxonomy of diversity creation methods. One can see that multi-view Co-Training (Section 5.7.1.1) is a special case of semi-supervised learning with committees. Therefore, the data mining community is interested in a more general *Co-Training style* framework that can exploit the diversity among the members of an ensemble for correctly predicting the unlabeled data in order to boost the generalization ability of the ensemble.

There is no SSL algorithm that is the best for all real-world data sets. Each SSL algorithm has its strong assumptions because labeled data is scarce and there is no guarantee that unlabeled data will always help. One should use the method whose assumptions match the given problem. Inspired by [219], we have the following checklist: If the classes produce well clustered data, then EM with generative mixture models may be a good choice; If the features are naturally divided into two or more redundant and independent sets of features, then standard *Co-Training* may be appropriate; If *SVM* is already used, then *Transductive SVM* is a natural extension; In all cases, *Self-Training* is a practical wrapper method.

Chapter 6

Active Learning

This chapter provides a general review of the literature on active learning.

6.1 What is Active Learning?

Most of the researchers in machine learning and data mining has been so far concentrating on analyzing already labeled data and building predictive models from them, rather than on how to collect labeled data. The data labeling process is often difficult, tedious, expensive, or time consuming, as it requires the efforts of human experts or special devices. *Active learning* is another way for integrating unlabeled data into supervised learning in order to boost the generalization and to reduce the cost of data annotation. It concentrates on closing the gap between data annotation and model building. It appears with several names in the literature such as, *query learning*, *sample selection*, *selective sampling* and sometimes *experimental design* in the statistics literature.

The key hypothesis of *Active learning* is that a learning algorithm can achieve better classification performance with a fewer number of labeled examples in case it is allowed to choose the examples from which it learns a classifier, compare between Figure 6.1 and Figure 5.4. An active learner is allowed to ask queries in the form of unlabeled examples to be labeled by an oracle such as a human annotator. Note that a passive learner does not have the luxury of selecting the important examples and to ask an oracle for their labels. Active learning is an iterative process well-motivated in many machine learning applications where data may be abundant but labels are time consuming or expensive to obtain. Interested readers in recent advances of active learning are directed to the literature survey of Settles [173].

One of the popular active learning applications is remote sensing image classification [187, 146]. The remote sensing sensors can produce data in large number of spectral bands. The objective of using such high resolution sensors is to discriminate among more ground cover classes and hence obtain a better understanding

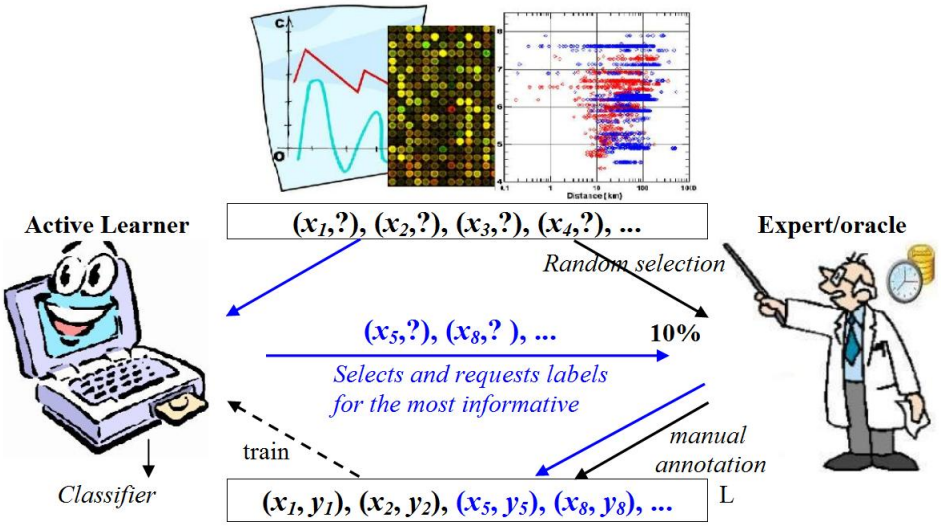


Figure 6.1: Graphical illustration of active supervised learning

about the nature of the materials that cover the surface of the Earth. This large number of classes and large number of spectral bands require a large number of labeled training examples (pixels) from all the classes of interest. The class labels of such training examples are usually very expensive and time consuming to acquire [175]. The reason is that identifying the ground truth of the data must be gathered by visual inspection of the scene at the same time that the data is being collected, by using an experienced analyst based on their spectral responses, or by other means. In any case, usually only a limited number of training examples can be obtained. These training examples are often used for deciding which features are useful for the discrimination among classes, and for designing classifiers based on these derived features (see Figure 5.3). The purpose of active learning is to study how to reduce the cost of data annotation by selecting the most informative training examples from a large amount of unlabeled data. Active learning algorithms can be divided into two categories: (i) stream-based selective sampling, and (ii) pool-based active learning.

6.2 Stream-Based Selective Sampling

An unlabeled example is randomly sampled from the actual distribution or from an approximation of the training-data distribution. Then the underlying classifier decides whether this example is informative or not. This approach is sometimes called stream-based or sequential active learning. The stream-based approach has been studied in several real-world applications, including part-of-speech tagging [46], sensor scheduling [102], and learning ranking functions for information retrieval [208]. Fujii et al. [67] applied active learning for word sense disam-

biguation, e.g., determining if the word bank means land alongside a river or a financial institution in a given context (they had studied only Japanese words). The approach not only reduces annotation effort, but also limits the size of the training set used in learning nearest-neighbor classifier, which in turn reduces the classification time. Drawbacks of the stream-based approach [125] are that it only sparsely samples the full distribution of possible examples labeling requests, and that the decision to label is made on each example individually, regardless of the other alternative examples.

6.3 Pool-Based Active Learning

For many real-world applications, large amount of unlabeled data can be collected at once. This motivates pool-based active learning [118], which assumes that there is a small set of labeled data L and a large pool of unlabeled data U available. Typically, unlabeled examples are queried in a greedy fashion, according to a utility or informativeness measure used to evaluate all examples in the pool or a subsample of the pool if U is very large. The pool-based approach has been studied for many real-world machine learning applications, such as cancer diagnosis [122], text categorization [118, 184], image classification and retrieval [183] and speech recognition [189]. Although it solves some drawbacks of the stream-based approach, it has its own drawback [125] that it may select examples that have high utility value but are in unimportant, sparsely populated regions (outliers). The labeling of these outliers will not improve classification accuracy of typical examples. An important factor for the success of any active learner is how to measure the utility or informativeness of an unlabeled example x_u before asking for its label. The more efficient active learner is the one that can achieve a target accuracy with the minimum number of queries.

6.4 Active Learning Algorithms

6.4.1 Uncertainty Sampling

It is the simplest and most commonly used active learning algorithm [118]. An initial classifier is created with a few labeled examples. Then for a predefined number of iterations, the current classifier predicts the class labels of the unlabeled examples and ranks them according to its confidence in its prediction. It queries the examples about which it is least confident. The classifier is retrained with the original labeled training set and the newly-labeled examples. For instance, when using a probabilistic binary classifier, the *Uncertainty Sampling* selects the example whose probability of being positive is near 0.5. For a probabilistic multi-class classifier, the utility of an example x_u is the Shannon entropy of the class

probability distribution $P = \{p_k = P(\omega_k|x_u) : k = 1, \ldots, K\}$ assigned to x_u:

$$H(P) = -\sum_{k=1}^{K} p_k \log p_k \tag{6.1}$$

The example with the maximum entropy is the least confident example. Tong and Koller [184] applied uncertainty sampling to support vector machines such that the examples closest to the linear decision boundary are the most informative examples. Figure 5.8 illustrates the influence of the unlabeled data on the decision boundary. Lewis and Catlett [117] applied uncertainty sampling on decision tree classifier after modifying it to have probabilistic output. Similarly, Fujii et al. [67] applied active learning to a probabilistic version of the nearest-neighbor classifier. The posterior membership probability of a given example to a class is defined as the proportion of the number of neighbor votes given to this class to the total number of neighbors. *Uncertainty Sampling* has limited applicability because it requires a probabilistic model that accurately estimates the confidence in its prediction. For instance, decision tree classifier is known to be inaccurate class probability estimator.

6.4.2 Query by Committee (QBC)

The QBC framework [66] involves maintaining an ensemble (committee) H of diverse classifiers (hypotheses) h_i which are all trained on the current labeled set L, see Chapter 3 for an overview of ensemble learning. Each committee member is then allowed to predict the class labels of unlabeled examples. The most informative unlabeled example to be queried is considered to be the example on which the disagreement on its prediction among the committee members is the greatest. The key idea of QBC is to reduce the hypotheses space $\mathbb{F}$, which is (as mentioned in Section 3.1) the set of all possible classifiers that are consistent with the current labeled training set L. Any base learning algorithm can be considered as searching for the best model within the hypotheses space, then the aim of active learning is to reduce the size of this space as much as possible (so that the search can be more precise) with as few labeled examples as possible. This is exactly what QBC does, by querying in controversial regions of the hypotheses space [173]. The QBC framework as shown in Algorithm 8 consists of two main steps: (1) construct a committee of classifiers that approximate different regions of the hypotheses space and (2) measure the disagreement among the committee member on predicting the class label of an unlabeled example x_u, denoted as the utility or the informativeness of x_u. Freund et al. [66] showed that under certain assumptions, *Query by Committee* can achieve an exponential decrease in the number of examples required to achieve a particular level of accuracy, as compared to random sampling. However, these theoretical results assume that the Gibbs algorithm is used to generate the committee of hypotheses used for

Algorithm 8 The pseudo code of *Query by Committee*

Require: set of labeled training examples (L), set of unlabeled training examples (U), maximum number of iterations (T), ensemble learning algorithm ($EnsembleLearn$), base learning algorithm ($BaseLearn$), committee size (N), sample size (n)

Training Phase

Construct a committee of N classifiers, $H = EnsembleLearn(L, BaseLearn, N)$
for $t \in \{1, \ldots, T\}$ **do**
  for each $x_u \in U$ **do**
    calculate the utility of x_u based on the current committee, $Utility(x_u, H)$
  end for
  Rank the examples in U based on their utility
  Select a subset S of n examples from U with the maximum utility
  Ask an oracle to label examples in S
  $U \leftarrow U \setminus S$, and $L \leftarrow L \cup S$
  Retrain or update the committee, $H = EnsembleLearn(L, BaseLearn, N)$
end for

Prediction Phase

return $H(x) = \sum_{i=1}^{N} w_i h_i(x)$ for a given sample x

sample selection. The Gibbs algorithm for most interesting problems is computationally expensive. In order to tackle this problem, Abe and Mamitsuka [12] have introduced *Query by Boosting* (QBoost) and *Query by Bagging* (QBag), which employ the well-known ensemble learning methods boosting FS97 and bagging Br96 to construct committees. In their approach, the utility of candidate examples is based on the margin of an example; where the margin is defined as the difference between the number of votes in the current committee for the most popular class label, and that for the second most popular label. Examples with smaller margins are considered to have higher utility.

Melville and Mooney [128] have proposed another variant of *Query by Committee*, called *ACTIVE-DECORATE* where *DECORATE* is used to construct the committee members. *DECORATE* [127] is an ensemble learning algorithm proposed previously by the same authors, see Section 5.7.6.3. To measure the expected utility of unlabeled examples, they used a generalized definition of margins, different from Abe and Mamitsuka [12], that take into account the probabilistic outputs of committee members instead of just crisp classifiers. Given the class membership probabilities predicted by the committee, then the margin is defined as the difference between the highest and the second highest membership probabilities. Again the most informative example is the example with the minimum margin. In order to compare the efficiency of different active learners, data utilization is used. The data utilization is the number of training exam-

ples required to achieve a target error rate. The more efficient active learner is the one that requires a smaller data utilization rate. *ACTIVE-DECORATE* outperforms *DECORATE*, *QBoost* and *QBag* in terms of data utilization. Dagan and Engelson [46] measured the informativeness of an unlabeled example x_u based on *vote entropy*, which is the entropy of the class probability distribution $P = \{p_k = P(\omega_k|x_u) : k = 1, \ldots, K\}$ assigned to x_u based on the majority votes of the committee members.

$$H(P) = -\sum_{k=1}^{K} p_k \log p_k, \text{ where } p_k = \frac{1}{N} \sum_{i:h_i(x_u)=\omega_k} 1 \tag{6.2}$$

McCallum and Nigam [125] proposed to use an information-theoretic utility measure that is *Jensen-Shannon* (JS) divergence if probabilistic classifiers are used to build the committee. Let P_i be the class probability distribution given to x_u by the i^{th} committee member, then JS divergence of a committee of size N is,

$$JS(P_1, \ldots, P_N) = H(\sum_{i=1}^{N} w_i P_i) - \sum_{i=1}^{N} w_i H(P_i) \tag{6.3}$$

where w_i is the weight of the i^{th} committee member and $H(P)$ is the Shannon entropy defined in Eq. (6.1). A high value of $JS(P_1, \ldots, P_N)$ indicates a high variance in the predicted class probability distributions P_i. It is zero if the distributions are identical.

6.4.3 Co-Testing

Co-Testing, which is the first approach to multi-view active learning, was proposed by Muslea et al. in [134]. It is inspired by the popular multi-view semi-supervised learning method, *Co-Training* [30]. That is, it requires two or more redundant and independent views of the data. Co-Testing algorithms work as follows: first, a set of classifiers $h_1, \ldots, h_N$ is trained by applying the base learning algorithm to the projection of the examples in L onto each view V_i. Then $h_1, \ldots, h_N$ are applied to all unlabeled examples in U and create the set of contention points U', which consists of all unlabeled examples for which at least two of these hypotheses disagree about its prediction. That is, $U' = \{x_u = (x_u^{(1)}, \ldots, x_u^{(N)}) \in U$ where $\exists i, j : h_i(x_u^{(i)}) \neq h_j(x_u^{(j)})\}$. Finally, they select to label one of the contention points and then repeat the whole process for a number of iterations. In [134], three types of utility measures are proposed for sample selection:

1. *naive*: randomly select one of the contention points. This measure is relevant for base classifiers that can not provide an accurate class probability estimates P_i because it leads to inaccurate confidence in their predictions where $Confidence(h_i(x_u)) = \max_{1 \leq k \leq K} P_i(\omega_k|x_u)$.

2. *aggressive*: select as query the contention point $x_{j^*} \in U'$ on which the least confident of the classifiers $h_1, \ldots, h_N$ makes the most confident prediction;

$$j^* = arg \max_{x_u \in U'} \min_{1 \leq i \leq N} Confidence(h_i(x_u)) \tag{6.4}$$

3. *conservative*: select the contention point $x_{j^*} \in U'$ on which the confidence in the predictions made by $h_1, \ldots, h_N$ is as close as possible (ideally, they would be equally confident in predicting different class labels); that is,

$$j^* = arg \max_{x_u \in U'} \left(\max_{1 \leq i \leq N} Confidence(h_i(x_u)) - \min_{1 \leq i \leq N} Confidence(h_i(x_u)) \right) \tag{6.5}$$

The shortcoming of this active learner is its multi-view requirement because most of the real-world applications do not have multiple views.

6.4.4 Active Learning for Regression

Krogh and Vedelsby Krogh95 considered committees of neural networks for learning real-valued functions (regression). The committee consists of N networks and the output of network i on example x is $h_i(x)$. The final output of the ensemble is the weighted average of the outputs of its member networks, $H(x) = \sum_{i=1}^{N} w_i h_i(x)$. They defined the *ensemble ambiguity* on the given example x as the variance in the predictions of the committee members:

$$\bar{a}(x) = \sum_{i=1}^{N} w_i (h_i(x) - H(x))^2. \tag{6.6}$$

It measures the disagreement among the ensemble members on x. They decomposed the *ensemble generalization error* into two terms, that is

$$E = \bar{E} - \bar{A} \tag{6.7}$$

where $\bar{E}$ is the weighted average of the generalization errors of the ensemble members ($\bar{E} = \sum_i w_i E_i$) and $\bar{A}$ is the weighted average of the ambiguities ($\bar{A} = \sum_i w_i A_i$) which is called the *ensemble ambiguity*. Note that A_i and E_i are the averages over the input distribution. In this work, a generalization of *Query-by-Committee* (*QBC*), that was developed for classification, was proposed. At each iteration of *QBC*, the unlabeled example for which the ambiguity is maximal, where the committee's variance is highest, is selected for labeling. If an example yields a high ambiguity, then it will have a high average error. Since the *ensemble generalization error* is not negative, $\bar{A}$ is the lower bound of $\bar{E}$ ($\bar{E} \geq \bar{A}$). Thus the aim of *QBC* is to select for labeling the unlabeled examples that minimize the variance in order to minimize the average error $\bar{E}$. The experiments have shown that active selection of training data led to improved performance compared to random selection.

6.4.5 Active Learning with Structured Instances

Settles [172] argued that many interesting real-world applications of machine learning involve learning from structured instances such as sequence labeling and multiple-instance learning.

6.4.5.1 Multi-Instance Active Learning

In MI learning problems, instances are naturally organized into bags and it is the bags, instead of individual instances, that are labeled for training. Active learning in MI settings is considered as a way to reduce the labeling burden in problem domains where labels can be acquired at both bag-level and instance-level granularities. This approach is well motivated in learning settings where it is inexpensive to acquire labels for bags and possible (but expensive) to acquire more fine-grained instance labels. Settles [172] proposed and explored four different active learning scenarios for MI problems, and presented a training algorithm that learns from labels at these mixed levels of granularity. He also introduced and evaluated several active query selection strategies motivated by the MI setting. Experiments have shown that active learning with instance labels can significantly improve the performance of an MI learning algorithm.

6.4.5.2 Active Learning for Sequence Labeling

The areas of natural language processing and bioinformatics, involve labeling and segmenting sequences. For instance, one can extract important organization names from a sentence (which is a sequence of words) or identify genes in DNA (which is a sequence of nucleic acids). Although there has been much work on active learning for classification, active learning for sequence labeling has received less attention. Settles [172] has presented two major advances in active learning research for sequence labeling tasks. First, he motivated and introduced a number of new query strategies for probabilistic sequence models. Second, he conducted an empirical analysis of previously proposed active learning methods along with his algorithms to compare their performance on multiple benchmark data sets.

6.5 Conclusion

Self-Training is an iterative *semi-supervised learning* algorithm corresponding to *Uncertainty Sampling* (Section 6.4.1) in which the *most confident* examples are selected to be automatically classified before they are included into the training set. *Co-Training* (*CT*) is a multi-view *semi-supervised learning* algorithm is corresponding to *Co-Testing* (Section 6.4.3) in which an ensemble of classifiers are trained using multiple redundant and independent sets of features (views). Each classifier classifies the unlabeled examples, adds the examples about which it is

most confident into the training set. The aim is that the *most confident* examples with respect to one classifier can be informative with respect to the other. In the contribution part of this thesis, a new committee-based single-view framework, denoted *Co-Training by Committee* will be introduced. It is inspired by *Query by Committee* (Section 6.4.2) active learning algorithm, in which an ensemble of diverse classifiers is constructed. Then the ensemble members are applied to unlabeled examples. The ones which the ensemble members are mostly confident in their predictions, are selected and added to the labeled training set. Now one can see that *semi-supervised learning* and *active learning* tackle the same problem but from different directions. That is both attempt to exploit the unlabeled data to improve the recognition rate of supervised learning algorithms and to minimize the cost of data labeling (see Table 6.1). *Semi-supervised learning* exploits the unlabeled data in which the committee members are *most confident* through their automatic annotation. *Active learning* exploits the unlabeled data in which the individual classifiers are *least confident* as they convey new discrimination information to the classifiers and manually label them. The main difference between

Table 6.1: Taxonomy of SSL and AL algorithms

Description	**SSL algorithm**	**AL algorithm**
Single-view, Single-learner Single-classifier	*Self-Training* [139] *EM* [51]	*Uncertainty Sampling* [118]
Multi-view, Single-learner Multiple classifiers	*Co-Training* [30] *Co-EM* [139]	*Co-Testing* [134]
Single-view, Multi-learner Multiple classifiers	*Statistical Co-Learning* [71] *Democratic Co-Learning* [210]	
Single-view, Single-learner Multiple classifiers	*Tri-Training* [215], *Co-Forest* [119] **Co-Training by Committee**	*Query by Committee* [66]

stream-based and pool-based active learning is that the former scans through the data sequentially and makes query decisions individually, whereas the latter evaluates and ranks the entire collection before selecting the best query. While the pool-based approach appears to be much more common, one can imagine situations where the stream-based approach is more relevant. For instance, when memory or processing power may be limited, as with mobile and embedded devices. Another situation where data is being generated continuously in a changing environment and thus storing data for pool-based approach is impractical.

Chapter 7

Applications and Evaluation Method

7.1 Applications for Visual Object Recognition

The recognition of 2D and 3D visual objects from 2D camera images is one of the most important goals in computer vision. All the real-world data sets used for visual object recognition in this thesis are described in Table 7.1. I intentionally select data sets with variance in number of features (D), number of classes (K) and number of data points (M). In the following sections, I will discuss in more details the feature extraction algorithms used to extract features from the images of each data set such as color histogram, orientation histogram, principle component analysis and optimal flow.

7.1.1 Fruits Image Recognition

In the context of the EU project on biomimetic multimodal learning in a mirror neuron-based robot *MirrorBot* [198], a scenario has been defined where the robot is situated infront of a table and different objects are lying on this table. The robot has to respond to spoken commands such as grasping or pointing to a certain object. The setup is shown in Figure 7.1. The commands are formulated in a simple language with a restricted vocabulary and an elementary grammar. For instance, "Bot show red apple".

The images of fruits placed on a white table were taken from the robot's point of view under different challenging conditions such as occlusion, changing lighting conditions, varying object views and positions. In a preprocessing step, the objects are localized and the regions of interest containing the objects are detected such that each image contains only one object. At the end, there are 840 colored images (resolution 384x288 pixels), exactly 120 images per each of the seven classes: green apple, red apple, tangerine, orange, yellow plum, red plum and lemon (see Figure 7.2).

In the feature extraction step, Fay [60] has extracted different feature types

Figure 7.1: MirrorBot test setup.

from the images and used them for object recognition. The experimental results have shown that orientation and color histograms are the most suitable feature types for object recognition under the challenging conditions of real-world robotic applications. It was shown that the results improve if several histograms are calculated from different parts of an image, instead of only one histogram per image. Each image is therefore divided into $m \times m$ potentially overlapping sub-images of equal size. If the parts overlap the result improves further as this compensates for non-optimally detected regions of interest. Note that a 20% overlapping percentage between the sub-images was shown to work well. For each sub-image, a separate histogram of b bins is calculated. Then the $m \times m$ histograms are concatenated to form the final $b \times m \times m$-dimensional feature vector that represents the whole image.

7.1.1.1 Color Histogram

Color histograms are calculated on the original RGB color space. A color histogram is calculated for each of the three color channels separately. The parameter b specifies the number of bins of the histogram where b color ranges $[a_i, b_i]$ are

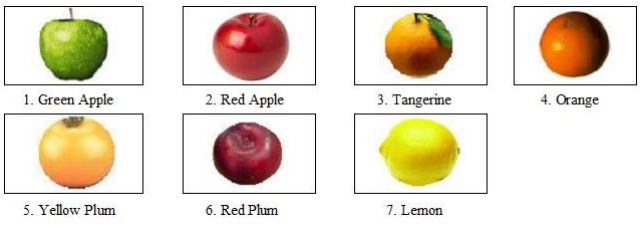


Figure 7.2: A sample of the images in the fruits data set

defined as $a_i = i\frac{256}{b}$ and $b_i = (i+1)\frac{256}{b}$ for $i = 0, 1, \ldots, b-1$. Then, each bin c_i will represent the number of pixels whose color values belonging to its range. For a colored image, the color histograms are represented by $3 \times b \times m \times m$-dimensional feature vector. For a gray scale image, gray value histograms are calculated and the dimension of the feature vector is $b \times m \times m$.

7.1.1.2 Orientation Histogram

The orientation histogram of an image [64, 43] provides information about the directions of the edges and their intensity. When graphically visualizing orientation histograms the x-axis specifies the different orientations and the y-axis gives the frequency of occurrence of these orientations.

To calculate the orientation histograms the gradient in x and y direction of the gray value image $I(x, y)$ is calculated using an edge detector such as the Sobel or the Canny edge detector (Figure 7.3). The gradient angles are discretized through dividing them into b ranges. The discrete gradient directions are weighted with the absolute gradient value and summed to form the orientation histogram.

- *Orientation histogram based on Sobel edge detection.* The edges are represented by areas with strong intensity contrasts, i.e. a strong ascent or descent of the intensity over a short distance. Thus the one-dimensional shape of an edge is a ramp. The presence of an edge can be indicated by consequently locating the maxima and minima of the first derivative of an image. The Sobel operator [72] convolves the gray-value image $I(x, y)$ with the two 3×3 convolution masks S_x and S_y respectively.

$$S_x = \frac{1}{8}\begin{pmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{pmatrix} \text{ and } S_y = \frac{1}{8}\begin{pmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{pmatrix} \tag{7.1}$$

  The resulting images $I_x(x, y)$ and $I_y(x, y)$ give the orientations in the x-direction (columns) and in the y-direction (rows) respectively:

$$I_x(x, y) = I(x, y) * S_x \tag{7.2}$$
$$I_y(x, y) = I(x, y) * S_y \tag{7.3}$$

  Thus the gradient $\bigtriangledown I(x, y)$ is given by

$$\bigtriangledown I(x, y) = \begin{pmatrix} I_x(x, y) \\ I_y(x, y) \end{pmatrix}, \tag{7.4}$$

  the gradient direction (orientation) $\theta(x, y)$ is calculated as

$$\theta(x, y) = arctan\frac{I_y(x, y)}{I_x(x, y)} \tag{7.5}$$

and the gradient strength (magnitude) $m(x, y)$ is defined as

$$m(x, y) = | \bigtriangledown I(x, y)| = \sqrt{I_x(x, y)^2 + I_y(x, y)^2} \tag{7.6}$$

To calculate the orientation histogram the gradient directions $\theta(x, y)$ are

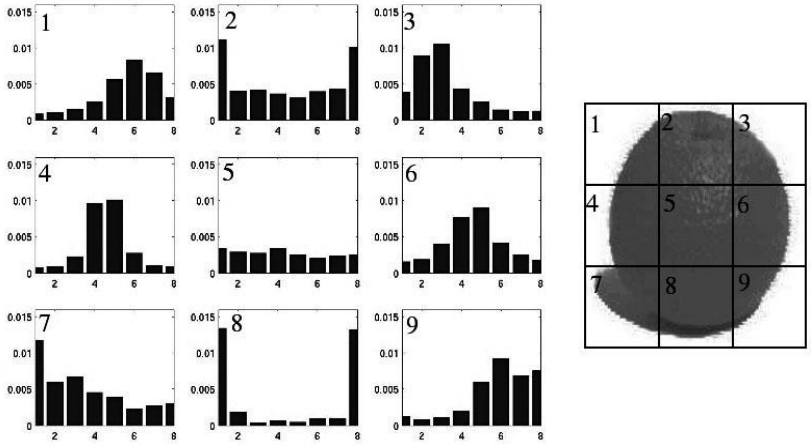


Figure 7.3: The image was divided into 3×3 sub-images. For each sub-image an orientation histogram with 8 bins is calculated. For sake of simplicity non-overlapping sub-images are depicted. (taken from [60])

discretized as follows: The parameter b specifies the number of discrete orientations d_i (number of bins) where the b orientation ranges $[a_i, b_i]$ are defined as $a_i = i\frac{360^o}{b}$ and $b_i = (i + 1)\frac{360^o}{b}$ where $i = 0, 1, \ldots, b - 1$. The corresponding discrete orientations is $d_i = (i + \frac{1}{2})\frac{360^o}{b}$ where $i = 0, 1, \ldots, b - 1$. The gradient direction $\theta(x, y)$ is then assigned the discrete direction d_i if $a_i \leq \theta(x, y) \leq b_i$. Then, for each discrete direction d_i, the gradient magnitude m of the pixels having this gradient direction is summed up.

- *Orientation histogram based on Canny edge detection.* Another more sophisticated way of calculating edges within an image is the Canny edge detector [38]. It adds some processing steps to the Sobel procedure, to obtain more concise edges. First, the noise is reduced by convolving the image with a Gaussian mask filter. The result is a less noisy image although it is blurred. In the second step, the gradient strength and direction are calculated from the intensity gradient of the smoothed image using the Sobel procedure described above. In the third step, local maxima in the direction of the gradient are found while suppress all others ensuring only one response to a single edge. Pixels with high intensity gradients are more likely to belong to an edge. In the final hysteresis step, a thresholding is performed to discard pixels with low gradient strength. Finally, the orientation histogram is then calculated analog to the orientation histogram described above.

- *Orientation Histogram Based on Opponent Colors.* Neither Sobel edge detection nor Canny edge detection mentioned above take into account color information. Another complex feature type are orientation histograms calculated on opponent color channels. They take into consideration the color information as well as the form information. The initial RGB trichromatic color space is transformed into an achromatic (A: black/white) and two opponent chromatic channels (P: red/green and Q: yellow/blue). The following transformation is used to convert an image from the RGB color space to the APQ color space.

$$\begin{pmatrix} A \\ P \\ Q \end{pmatrix} = \begin{pmatrix} 0.887 & 0.461 & 0.0009 \\ -0.46 & 0.88 & 0.01 \\ 0.004 & -0.01 & 0.99 \end{pmatrix} \begin{pmatrix} R \\ G \\ B \end{pmatrix} \tag{7.7}$$

  An orientation histogram is calculated on each of these three channels analog to the Sobel procedure on the gray-value images described above. This results in three different types of feature vectors that combines both color and form information.

7.1.2 StatLog Handwritten Digits

This StatLog data set [130] consists of 18000 images representing the 10 handwritten digits gathered from German postcodes (1800 images per class). They were read by one of the automatic address readers built by a German company. The handwritten digits were digitized onto images with 16×16 pixels where each pixel represented in 8-bit gray levels. They are scaled in width and height. Figure 7.4 shows some examples of the digits within this data set.

Figure 7.4: A sample of the handwritten digits data set

7.1.2.1 Principal Component Analysis (*PCA*)

In *PCA* [89], the data is transformed into other orthogonal dimensions. These new dimensions are identified by the eigenvectors of the covariance matrix of the input data. This technique can be used for dimensionality reduction because the dimensions with the highest variance in the data are the eigenvectors associated with the highest eigenvalues, called *principle components*. Each 16×16 image matrix is reshaped into 256-dimensional vector. Then *PCA* is performed and the 256-dimensional vectors are projected onto the top 40 principal components.

7.1.2.2 Orientation Histogram

Each image was divided into $m \times m$ overlapping sub-images (for $m = 2, 3$). Then, an orientation histogram with b bins was extracted from each sub-image as described in Section 7.1.1.2. The histograms were concatenated to form a single feature vector.

7.1.3 UCI Handwritten Digits

The Handwritten Digits that are described by four sets of features and are publicly available at UCI Repository [27]. The digits were extracted from a collection of Dutch utility maps. A total of 2000 patterns (200 patterns per class) have been digitized in binary images (see Figure 7.5). See Table 7.1 for more details about the extracted feature types.

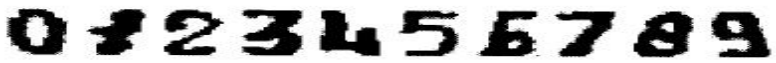


Figure 7.5: Sample of the handwritten digits

Figure 7.6: Examples of the COIL data set

7.1.4 Columbia Object Image Library (COIL)

The Columbia Object Image Library [137] has a data set that consists of 1440 size-normalized gray-scale images of 20 different three-dimensional objects infront of a black background. The objects represent cups, toys, drugs and cosmetics. The images are of size 128×128 pixels. For each object, there are 72 images that are taken from different views at pose intervals of 5 degree covering a total

Table 7.1: Description of the data sets

Data set	K	M	Feature set	D	Description	Chapter
ionosphere	2	351	-	34	see UCI Repository [27]	9, 10
digits I	10	20000	*image-vector*	256	A 256-dim vector results from reshaping the image 16x16 pixels matrix	13,14
			pca-40	40	A feature vector results from projecting the *image-vector* onto the first 40 principal components of PCA	8,13,14
			orienthist2x2	32	An image was divided into $2x2$ overlapped sub-images. An orientation histogram with 8 bins is calculated from each sub-image as described in Section 7.1.1.2. The four histograms were concatenated to form a 32-dimensional feature vector	8
			orienthist3x3	144	An image was divided into $3x3$ overlapped sub-images. An orientation histogram with 16 bins is calculated from each sub-image as described in Section 7.1.1.2. The nine histograms were concatenated to form a 144-dimensional feature vector	14
			rows-sum	160	A 160-dim vectors representing the sums over the rows of the original image and images results from rotating it 9 times	14
			cols-sum	160	A 160-dim vectors representing the columns over the rows of the original image and images results from rotating it 9 times	14
digits II	10	2000	*mfeat-pix*	240	240 pixel averages in 2 x 3 windows	9,10
			mfeat-kar	64	64 Karhunen-Love coefficients	9,10
			mfeat-fac	216	216 profile correlations	9,10
			mfeat-fou	76	76 Fourier coefficients of the character shapes	9,10
fruits	7	840	*colorhist3x3*	216	nine color histograms	8,9,10,13,14
			sobel4x4	128	16 orientation histograms based on Sobel detector	9,10,13,14
			canny3x3	128	9 orientation histograms based on Canny detector	8,14
			APQ-BW2x2	128	4 orientation histograms based on opponent colors	14
			APQ-RG4x4	128	16 orientation histograms based on opponent colors	14
COIL20	20	1440	*colorhist1x1*	24	a color histogram with 24 bins	9,10,13
			colorhist2x2	96	four color histograms with 24 bins	8
			orienthist2x2	32	four orientation histograms based on Sobel Detector	8,9,10,13
Cohn-Kanade	4	358	*mouth-orienthist2x2*	48	see Section 7.1.5	12
			face-optical-flow	48		
			mouth-optical-flow	48		
texture	11	1100	-	40	modified moments in four orientations	9,10,13
letters	26	2000	-	16	see UCI Repository [27]	13
satimage	6	1286	-	36	see UCI Repository [27]	13

of 360 degrees. Figure 7.6 shows frontal views of the twenty different objects. In her dissertation, Fay [60] has extracted different feature types from the images and used them for object recognition. The experimental results have shown that orientation and color histograms are the most relevant feature types. Each image

was divided into $m \times m$ overlapping sub-images (for $m = 1, 2$).

7.1.4.1 Color Histogram

A color histogram with 24 bins was extracted from each sub-image as described in Section 7.1.1.1. The histograms were concatenated to form the feature vector used for classification.

7.1.4.2 Orientation Histogram

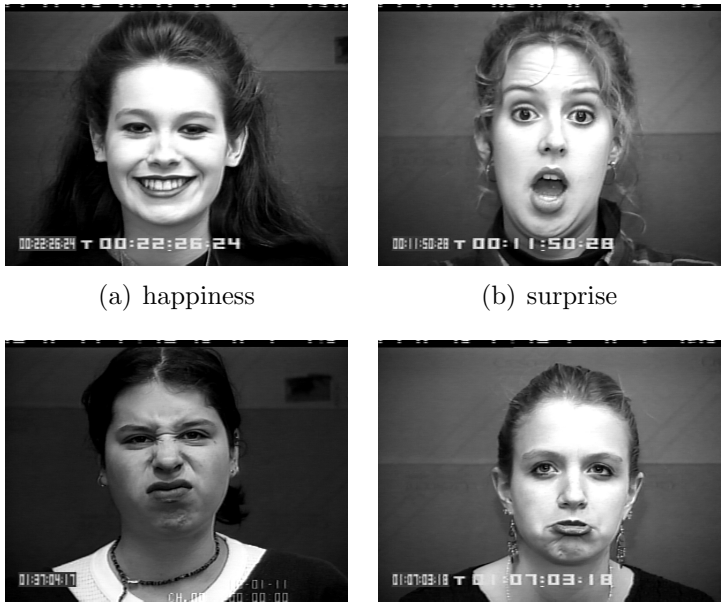
An orientation histogram with 8 bins was extracted from each sub-image based on Sobel edge detection described in Section 7.1.1.2. Then the histograms were concatenated to form the input feature vector.

7.1.5 Emotion Recognition from Facial Expressions

The Cohn-Kanade dataset is a collection of image sequences with emotional content [93], which is available for research purposes. It contains image sequences, which were recorded with a Panasonic WV3230 camera and digitized to have a resolution of 640×480 (sometimes 490) pixels with a temporal resolution of 33 frames per second. Every sequence is played by an amateur actor who is recorded from a frontal view. In his Masters thesis, Schels [163] has studied to recognize emotions from facial expressions based on different areas of the face and using different feature types. The data set contained 488 sequences from 97 individuals. He omitted 53 complex sequences that do not correspond to any observable emotion and used the other 432 sequences in his experiments. The sequences always start with a neutral facial expression and end with the full blown emotion which is one of the six categories "happiness", "anger","surprise", "disgust", "sadness" or "fear". Figure 7.7 shows four of the studied facial expressions.

7.1.5.1 Data Annotation

To acquire a suitable label the sequences were presented to 15 human test persons (13 male and two female). The sequences were presented as a video. After the play-back of a video the last image remained on the screen and the test person was asked to select a label. Thus, a label for every sequence was created as the majority vote of the 15 different opinions. The result of the labeling process is given in Table 7.2, showing the confusion matrix between the majority of decisions and individual decisions. It is revealed that the human data annotation is difficult which is one of the motivations of semi-supervised learning. For instance, the consensus between the majority the 15 persons and the individuals that the sequences are labeled as "disgust" is only 67%. Due to their sparse appearance, the classes "fear" (25 videos) and "anger" (49 videos) were excluded from my



(a) happiness (b) surprise

(c) disgust (d) sadness

Figure 7.7: Example images used to test and train the recognition system

experiments. More details on the preprocessing procedure and the annotation process can be found in [163].

7.1.5.2 Feature Extraction

The main problem to design an automatic facial expression recognition systems is how to categorize the emotions and how to find the most relevant features: one way is to model emotions through a finite set of emotional classes such as anger, joy, sadness, etc, another way is to model emotions by a continuous scales, such as valence (the pleasantness of the emotion) and arousal (the level of activity) of an expression [113]. In this experiment, a discrete representation in six emotions is used. Using his segmentation tool, Schels [163] have identified for each image in a sequence four prominent regions: the full facial region, the left eye, the right eye and the mouth. For these regions orientation histograms, principal components and optical flow features have been computed.

- *Orientation histograms.* They were successfully applied for the recognition of hand gestures [64] and faces [169] from single images. Each of the segmented regions is divided into four overlapping sub-images and an orientation histogram with 12 bins was calculated from each sub-image using Sobel edge detection (Figure 7.8) as described in Section 7.1.1.2. Thus, each facial area is represented by a 48-dimensional feature vector.

Table 7.2: Confusion matrix of the majority vote (rows) against the individual test persons decisions (columns), given as average. The last column shows the total number of sequences per emotion as determined by the majority of the test persons. For instance, 25 sequences have been annotated as "fear" by the majority but 27% of them were mislabeled by the individuals.

maj. \indiv.	happ.	ang.	surp.	disg.	sad.	fear	no. samples
happiness	**0.99**	0	0	0	0	0.01	105
anger	0	**0.8**	0	0.12	0.07	0.01	49
surprise	0.01	0	**0.78**	0	0.01	0.19	91
disgust	0.01	0.15	0.01	**0.67**	0.01	0.15	81
sadness	0	0.08	0.02	0.02	**0.88**	0.01	81
fear	0.01	0.01	0.14	0.27	0.01	**0.56**	25

Figure 7.8: The Sobel edge detection filter applied to an image from the Cohn-Kanade database

- *Optical flow.* The motivation to extract this type of features is to analyze the motion field of a sequence, that is to project the motion onto a 2D image plan. In order to extract the facial motion in these regions, optical flow features from each pair of consecutive images have been computed, as suggested in [161]. The optical flow measures shifts in gray values and can serve as an estimation for the motion field. A biologically inspired optical flow estimator is used, which was developed in the Vision and Perception Science Lab of the Institute of Neural Processing at the University of Ulm [22].

- *Principal Component Analysis.* The images representing the four facial regions are transformed into standard dimensions as follows: $40x40$ pixels for the full facial image, $10x20$ for the mouth and $20x20$ for the eyes. Then *PCA* is performed as described in Section 7.1.2.1. Then the image data is projected onto the top 150 principle components resulting in a 150-dimensional feature vector for each area.

Experiments have been conducted by Schels [163] used the three feature types extracted from four facial regions. The results have shown that out of the 12 used feature types, Optical flows and orientation histograms from the mouth region and Optical flows from the full facial region are the most suitable feature types. Thus, I restrict my experiment in Chapter 12 on these three types.

7.1.6 Benchmark Data Sets

7.1.6.1 Letters Image Recognition

The Letter Image Recognition data set [27] consists of 20000 images of the 26 capital letters in the English alphabet. From black and white images, 16 primitive integer-valued features were derived representing simple statistical characteristics of the pixel distribution. They are linearly scaled to a range from 0 to 15. To generate the images 20 different fonts were used and randomly distorted resulting in 20,000 unique samples. Figure 7.9 shows examples of the letters.

Figure 7.9: A sample of the letters Image Recognition Data

7.1.6.2 Texture

The aim is to distinguish between 11 different textures in the Brodatz album (Grass lawn, Pressed calf leather, Cotton canvas, Beach sand, ...), each pixel (data point) being characterized by 40 attributes built by the estimation of fourth order modified moments in four orientations: 0, 45, 90 and 135 degrees [17]. The data set contains 500 instances for each class but for simplicity we used only a random subsample of 100 instances per class.

7.2 Performance Evaluation

For comparison of different learning algorithms, it is necessary to evaluate their performance. The result of a single run of the algorithm is neither reliable

nor meaningful because the performance of algorithms shows a certain variance. Moreover it is not sufficient to evaluate the learning algorithm only on the training data because the result is too optimistic. Thus it is necessary to use an unseen test data set different from the training data set for more realistic estimation of its generalization ability. A method accounting for this is the cross-validation approach. This approach estimates the classification accuracy of the evaluated learning algorithm. The estimated classification accuracies of two learning algorithms can than be compared by means of statistical significance tests which determine whether the difference between the performance of the two algorithms is only by chance or a considerable difference.

7.2.1 Cross-Validation

Cross-validation is a common technique for evaluating the performance of learning algorithms when only a limited number of examples is available. The idea behind it is not to use the complete data set for training but to use only a part of the data set for training and the rest for testing the performance of the algorithm.

To conduct one run of cross-validation, randomly permute the data and divide it into k parts of equal size where it might not always be possible to split the data into parts of exactly the same size. These k parts are called *folds*. The number of folds is naturally limited to $2 \leq k \leq M$ where M is the total number of examples in the data set. Then k experiments are performed in each of which one of the k parts is respectively used as test set and the remaining k-1 parts are used as training set. If the permutation and splitting of the data is repeated more than one time this is referred to as repeated cross-validation where each iteration is called a *run*. Thus for r-times k-fold cross-validation, $r \times k$ experiments are conducted where a_{ij} is the accuracy of the evaluated algorithm in the j^{th} fold of the i^{th} run such that $i = 1, \ldots, r$ and $j = 1, \ldots, k$. Remember that the examples in the j^{th} part of the i^{th} run is used for testing. Thus the $r \times k$ accuracies a_{ij} can then be used to calculate a mean accuracy $a = \frac{1}{rk} \sum_{i=1}^{r} \sum_{j=1}^{k} a_{ij}$.

There are different variants of cross-validation, depending on the choice for k. The *holdout method* is the simplest form of cross-validation with $k = 2$. It means that the data set is split into two parts, the training and the test set. The most costly form of cross-validation is *leave-one-out* cross-validation with $k = M$, i.e. the number of folds is equal to the number of examples in the data set. With $2 < k < M$ the variant is called *k-fold cross-validation*.

A special version called *stratified cross-validation* accounts for potential imbalanced data set where there are differences in the class frequencies. It considers the relative class frequencies when splitting the data set into folds such that the relative class frequencies in each fold are the same as in the complete data set.

7.2.2 Significance Test

In order to compare two learning algorithms A and B, r-times k-fold cross-validation is conducted for each algorithm. That is, $r \times k$ experiments are conducted with both algorithms using exactly the same training and test data sets and the respective test accuracies a_i and b_i are recorded. Thus the $n = r \times k$ accuracies are paired and the differences of the accuracies $d_i = a_i - b_i$ with $i = 1, \ldots, n$ can be used as input for paired statistical significance tests.

Significance tests are used to statistically detect differences on the basis of observed values. It examines previously formulated hypotheses where the null hypothesis H_0 assumes that "there is no significant difference between A and B" and the alternative hypothesis H_1 assumes that "there is a significant difference between A and B". Significance tests can determine whether the difference is only by chance or a considerable difference with a low probability of error. This probability of error is defined by the significance level α that limits the error probability to reject the null hypothesis although the null hypothesis is correct.

The quality of a statistical test is typically evaluated on the basis of type I and type II errors. A type I error is the erroneous rejection of the null hypothesis, i.e. detecting a difference when actually there is no difference. The probability of committing a type I error is specified by the significance level α. A type II error corresponds to the erroneous acceptance of the null hypothesis, i.e. indicating that there is no difference when actually there is difference. The probability of the occurrence of a type II error is denoted by β. There exists an interdependency between the two types of errors. The reduction of the probability of making one error increases the probability of the occurrence of the other error. The size of a statistical test is the probability of a type I error. The power of a statistical test is defined by the probability of correctly rejecting a false null hypothesis. The power is defined as $1 - \beta$.

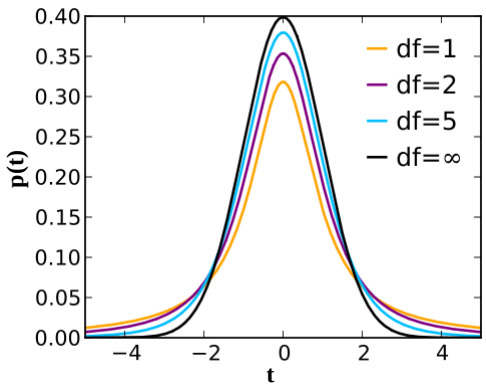


Figure 7.10: Student's t-distribution

Significance tests usually provide a test statistic t that is used to evaluate the statistical significance. The distribution of the test statistic (see Figure 7.10)

specifies the probability that the test statistic takes a certain value depending on the number of observations and the utilized test procedure. The corresponding probability is the so-called p-value. This value is used to decide whether the observed difference is statistically significant or not. The observation is regarded as statistically significant if the p-value is smaller than the previously defined significance level α, in this thesis $\alpha = 0.05$. The null hypothesis H_0 can than be rejected in favor of the alternative hypothesis H_1. Statistical significance test can be divided into parametric and non-parametric tests. Parametric tests, such as the *t-test*, require the observed values to follow a particular distribution and thus rely on the estimation of parameters specifying this distribution. Non-parametric or distribution-free test, such as the *maximum test*, the *sign test* or the *signed rank test*, make no requirements concerning the distribution the data follows.

7.2.3 Paired t-Test

It is a commonly used significance test and it requires require the observed values to follow the Students t-distribution. The mean m and variance σ^2 are calculated on d_i as follows:

$$m = \frac{1}{n}\sum_{i=1}^{n} d_i \tag{7.8}$$

and

$$\hat{\sigma}^2 = \frac{1}{n-1}\sum_{i=1}^{n}(d_i - m)^2. \tag{7.9}$$

Then the test statistic t is given by

$$t = \frac{m}{\sqrt{\frac{1}{rk}\hat{\sigma}^2}} \tag{7.10}$$

and follows a t-distribution with $df = kr - 1$ degrees of freedom. This test statistic t is then compared against the Students t-distribution to determine the corresponding p-value. This value is then used to determine whether the observed difference is statistically significant or not.

Although for one cross-validation run there is no overlap of the k different test data sets, the data sets used for training overlap considerably as each two training sets always consist of k - 2 identical folds. Considering different runs, there is also overlap for the training data as well as for the test data. This violates the independence assumption most significance tests require. Thus the variance is underestimated and the standard *t-test* is not applicable to these experiments.

Nadeau and Bengio [135] proposed to compensate the highly violated independence assumption by correcting the variance, resulting in the corrected test statistic $\tilde{t}$ as follows:

$$\tilde{t} = \frac{m}{\sqrt{(\frac{1}{rk} + \frac{n_2}{n_1})\hat{\sigma}^2}} \tag{7.11}$$

where n_1 is the number of training examples and n_2 is the number of test examples. The test statistic $\tilde{t}$ is then used to determine the corresponding p-value based on a Students t-distribution with $df = kr - 1$ degrees of freedom.

Part II

Contributions

Chapter 8

Co-Training with Class Hierarchies

8.1 Introduction

In this chapter, the problem of how to exploit unlabeled data to boost the classification performance is addressed in the application domains characterized by: (1) multiple sufficient and redundant views, (2) a large number of classes, (3) a small amount of labeled examples, and (4) a large amount of unlabeled data. Despite the practical benefits of combining *semi-supervised learning* (Chapter 5) and multi-class decomposition schemes (Chapter 4), there is not much related work in the machine learning literature, see [69].

The main contribution of this chapter is the combination of the tree-structured approach (Section 4.6) with the *Co-Training* semi-supervised learning algorithm (Section 5.7.1.1) through two different architectures. In the first architecture, a tree-structured ensemble of binary RBF networks is trained on each given view. Then, using *Co-Training* the most confident unlabeled examples labeled by each tree ensemble classifier are added to the training set of the other tree classifier; we call this scheme *cotrain-of-trees* (see Figure 8.1). In the second architecture, first the given K-class problem is decomposed into K-1 simpler binary problems using the tree-structured approach. Then using *Co-Training* a binary RBF network is trained on each given view to solve each binary problem; we call this last scheme *tree-of-cotrains* (see Figure 8.2). In order to combine the intermediate results of the internal nodes within each tree, a combination method based on Dempster-Shafer evidence theory is used [61]. Then *cotrain-of-trees* and *tree-of-cotrains* were evaluated on three real-world 2D and 3D visual object recognition tasks. Let $L = \{(X_\mu, y_\mu)|X_\mu = (x_\mu^{(1)}, x_\mu^{(2)}), y_\mu \in \Omega, \mu = 1, \ldots, m\}$ be the set of labeled training examples where X_μ is an example described by two D_i-dimensional feature vectors $x_\mu^{(i)} \in \mathbb{R}^{D_i}$, y_μ denotes the class label of X_μ and $\Omega = \{\omega_1, \ldots, \omega_K\}$ is the set of target classes (ground truth). Also let $U = \{X_u = (x_u^{(1)}, x_u^{(2)})|u = 1, \ldots, n\}$ be the set of unlabeled data. The work in this chapter has been previously published ([7, 9]).

8.2 Co-Training of Tree-Structured Ensembles

The formal description of the first architecture, *cotrain-of-trees*, is provided in Algorithm 9 with an illustration in Figure 8.1.

Algorithm 9 Co-Training of Tree-Structured Ensembles

Require: set of m labeled training examples (L), set of unlabeled examples (U), two example representations (V_1,V_2), maximum number of co-training iterations (T), tree ensemble learning algorithm ($TreeLearn$), incremental tree learning algorithm ($OnlineTreeLearn$), base learning algorithm ($BaseLearn$), incremental base learning algorithm ($OnlineBaseLearn$), number of classes (K), number of unlabeled examples in the pool (u), prior probability of classes $\{Pr_k\}_{k=1}^{K}$

Training Phase

Construct two tree ensembles using initial L,
$H_1^{(0)} = TreeLearn(V_1(L), BaseLearn)$ and
$H_2^{(0)} = TreeLearn(V_2(L), BaseLearn)$
for $t = 1$ to T **do**
  if U is empty **then**
    Set $T = t$-1 and abort loop
  end if
  for $i = 1$ to 2 **do**
    Create a pool U' of u examples from U
    Apply the tree ensemble $H_i^{(t-1)}$ on U'.
    Select a subset $\pi_{i,t}$ as follows: for each class ω_k, select the $n_k \propto Pr_k$ most confident examples assigned to class ω_k
    Set $U' = U' \setminus \pi_{i,t}$, $L_{2-i+1} = L_{2-i+1} \cup \pi_{i,t}$ and $U = U \cup U'$
  end for
  Update the tree ensembles,
  $H_1^{(t)} = OnlineTreeLearn(V_1(L_1), H_1^{(t-1)}, OnlineBaseLearn)$ and
  $H_2^{(t)} = OnlineTreeLearn(V_2(L_2), H_2^{(t-1)}, OnlineBaseLearn)$
end for

Prediction Phase

return $\frac{H_1^{(T)}(x)+H_2^{(T)}(x)}{2}$ for a given example x

Given a set L of labeled examples, and a set U of unlabeled examples, the algorithm begins by constructing two *Single-View Trees* $H_1^{(0)}$ and $H_2^{(0)}$ using the tree ensemble learning algorithm $TreeLearn$ (Section 4.6) where $V_1(L)$ and $V_2(L)$ are used as input feature set, respectively. The following steps are repeated for T times or until U becomes empty. For each iteration t and for each view i, a set U' is created of u examples randomly drawn from U without replacement. It is computationally more efficient to use a pool U' instead of using the whole set U. Then, $H_i^{(t-1)}$ is applied to each example $X_u = (x_u^{(1)}, x_u^{(2)}) \in U'$ in order to predict

the class label of $x_u^{(i)}$. Afterward, the unlabeled examples are ranked by the confidence in the class prediction. A set $\pi_{i,t}$ is created that contains the n_k most confident examples assigned to class ω_k. Then $\pi_{i,t}$ is removed from U' and inserted into the training set of the other tree ensemble. Then, $H_1^{(t)}$ and $H_2^{(t)}$ are refined using an online version of the tree ensemble learning algorithm $OnlineTreeLearn$ (See Appendix 1) on their augmented training sets. Like *Standard Co-Training* the objective is that the confident examples with respect to the tree ensemble $H_i^{(t-1)}$ can be informative with respect to the other tree ensemble $H_{2-i+1}^{(t-1)}$. In the classification phase, the final output for a given example is the average of the outputs of the two tree classifiers created at the final *Co-Training* iteration, $H_1^{(T)}$ and $H_2^{(T)}$. It is expected that the proposed *committee-based* confidence measure (that is based on an ensemble of K-1 binary classifiers) is more accurate than a *single classifier based* one. However, mislabeling of unlabeled examples is not avoidable so that H_i receives noisy examples from time to time. Fortunately, Goldman and Zhou [71] shows that the negative effect caused by adding such mislabeling noise could be compensated by augmenting the training set with sufficient amount of newly labeled examples.

8.2.1 Confidence Measure

An important factor that affects the performance of any Co-Training style algorithm is how to measure the confidence in predicting the class label of an unlabeled example which determines its probability of being selected. An inaccurate confidence measure leads to adding mislabeled examples to the labeled training set which leads to performance degradation during the *SSL* process. The confidence is measured based on the ensemble of binary classifiers H_i.

8.2.1.1 Estimating Class Probabilities

The confidence in predicting the class label of an unlabeled example can be measured as the highest predicted class probability.

$$Confidence(X_u, H_i^{(t-1)}) = \max_{1 \leq k \leq K} H_i^{(t-1)}(X_u, \omega_k) \tag{8.1}$$

Unfortunately, the classical *decision tree-like* approach to combine the K-1 binary classifiers within a class hierarchy, discussed in Section 4.6.2.1, does not provide a class probability distribution. Thus, the *evidence-theoretic* combiner discussed in Section 4.6.2.4 will be adopted where the confidence is defined as

$$Confidence(X_u, H_i^{(t-1)}) = \max_{1 \leq k \leq K} m^{(i)}(\theta_k) \tag{8.2}$$

and the predicted class label is

$$\hat{y} = arg \max_{1 \leq k \leq K} m^{(i)}(\theta_k) \tag{8.3}$$

where $m^{(i)}(\theta_k)$ is the belief in the hypothesis θ_k that an example X_u belongs to class ω_k provided by tree ensemble H_i trained at iteration $t-1$.

8.3 Tree-Structured Co-Training

The second architecture, *tree-of-cotrains*, is formally defined in Algorithm 10 and illustrated in Figure 8.2. Given a classification task with K classes, a set L of labeled examples and a set U of unlabeled examples where each example is described by two sets of features (V_1 and V_2). The algorithm begins by decomposing the K-class problem into K-1 binary problems using the tree ensemble learning algorithm $TreeLearn$ (See Section 4.6) where the concatenation of the feature vectors of the two views is used as an input feature set to construct the *multi-view tree*. Then for each binary problem j, *Co-Training* (see Section 5.7.1.1) is applied using $L_j \subseteq L$ and $U_j \subseteq U$ as input data where L_j is the set of training examples that are members of (meta-)class Ω_j,

$$L_j = \{(X,t)|(X,y) \in L, t = 1 \text{ if } y \in \Omega_{2j} \text{ and } t = 2 \text{ if } y \in \Omega_{2j+1}\} \tag{8.4}$$

and U_j is the set of unlabeled examples that are assigned to (meta-)class Ω_j by the predecessor node classifiers,

$$U_j = \{X_u|X_u \in U, \Omega_j = H_{par(j)}(X_u)\}. \tag{8.5}$$

That is, for each node j, two binary classifiers $h_{j1}^{(0)}$ and $h_{j2}^{(0)}$ are trained using $BaseLearn$ (Appendix 2) and $L_j = (L_{j1}, L_{j2})$. Then for T times or until U_j becomes empty, for each view i, $h_{ji}^{(t-1)}$ is used to predict the class labels of the unlabeled examples in U'. The most confident examples assigned to (meta-)class Ω_{2j} and (meta-)class Ω_{2j+1} are removed from U' and added with their predicted class label to $L_{j,2-i+1}$. Then, $h_{j1}^{(t)}$ and $h_{j2}^{(t)}$ are updated with the augmented training set using an online version of the base learning algorithm $OnlineBaseLearn$ such as RBF network online learning algorithm defined in Appendix 3.

In classification phase, the decision of each node j is the average of the predictions of the two binary classifiers created at the final *Co-Training* iteration, $h_{j1}^{(T)}$ and $h_{j2}^{(T)}$. Then, the final decision of the whole class hierarchy is the combination of the intermediate decisions of the K-1 nodes using either hard combiner or evidence-theoretic soft combiner (see Section 4.6.2.4). It is worth mentioning that there are two sources of knowledge transfer in *tree-of-cotrains*: (1) Through the co-training between the pair of binary classifiers h_{j1} and h_{j2} at each node j. (2) Through the selection of U_j as defined in Eq. (8.5) where each parent node transfers knowledge to its child nodes.

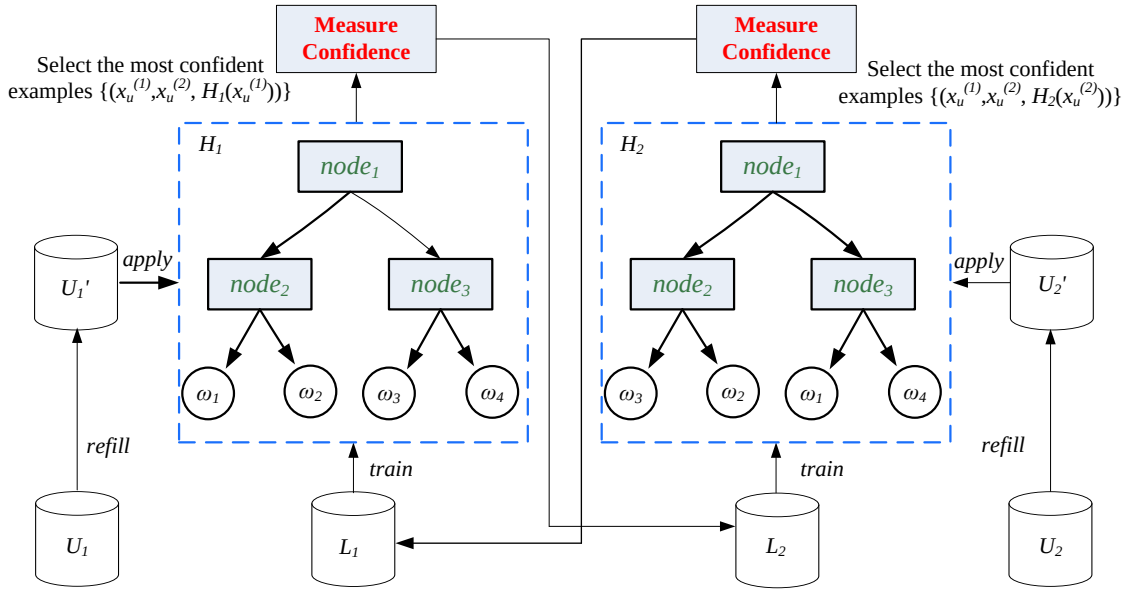


Figure 8.1: Architecture I: cotrain-of-trees

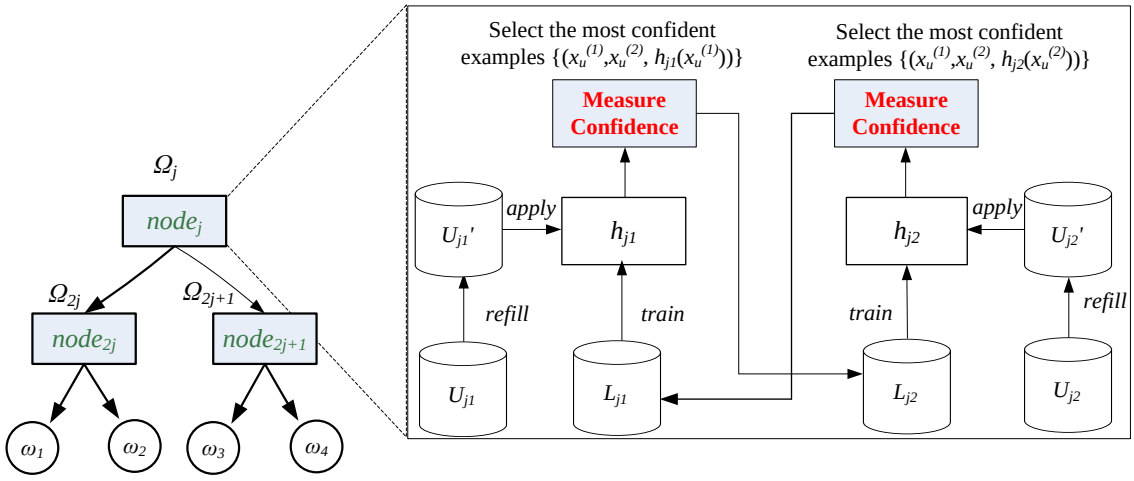


Figure 8.2: Architecture II: tree-of-cotrains

Algorithm 10 Tree-Structured Ensemble of Co-Training

Require: set of m labeled training examples (L), set of unlabeled examples (U), two example representations (V_1,V_2), base learning algorithm ($BaseLearn$), maximum number of *Co-Training* iterations (T), number of unlabeled examples in the pool (u), hierarchical combination method ($TreeCombiner$)

Training Phase

1: $\Omega_1 = \Omega$

2: Generate Class Hierarchy as follows:

1. $C = \{(c_k, \omega_k)\}_{k=1}^{K} = GetClassCentroids(L)$
2. $hierarchy = BuildNode(\Omega_1, C)$

3: **for** each internal node j at $hierarchy$ **do**

4: Filter the training examples L,
$L_j = \{(x,t)|(x,y) \in L$ and $t = 1$ if $y \in \Omega_{2j}$ and $t = 2$ if $y \in \Omega_{2j+1}\}$

5: Filter the unlabeled data U,
$U_j = \{x|x \in U$ that is assigned by the higher level nodes to $\Omega_j\}$

6: Train binary classifier,
H_j= *Co-Training*$(L_j, U_j, BaseLearn, T, u)$

7: **end for**

Prediction Phase

8: **return** $TreeCombiner(x, hierarchy)$ for a given x

8.3.1 Confidence Measure

Unlike the first architecture, the unlabeled examples are labeled and the labeling confidence is measured at each node j based on a single binary classifier, that is either h_{j1} or h_{j2}. Many classifiers can provide class probability estimates (CPE) such as Naive Bayes classifier or return real-valued outputs that can be transformed to CPEs such as neural networks and decision trees. That is, $h_{ji} : V_i \times \{\Omega_{2j}, \Omega_{2j+1}\} \rightarrow [0, 1]$. Therefore, the confidence in the class label prediction of an unlabeled example X_u can be measured as the highest predicted class probability.

$$Confidence(X_u, h_{ji}^{(t-1)}) = \max\{h_{ji}^{(t-1)}(x_u^{(i)}, \Omega_{2j}), h_{ji}^{(t-1)}(x_u^{(i)}, \Omega_{2j+1})\} \tag{8.6}$$

and the predicted class label is

$$\hat{y} = arg\max\{h_{ji}^{(t-1)}(x_u^{(i)}, \Omega_{2j}), h_{ji}^{(t-1)}(x_u^{(i)}, \Omega_{2j+1})\} \tag{8.7}$$

where $h_{ji}^{(t-1)}(x_u^{(i)}, \Omega_{2j})$ is the probability given by the classifier h_{ji} that an example X_u belongs to meta-class Ω_{2j} at iteration t.

8.4 Application to Visual Object Recognition

The recognition of visual objects from 2-D camera images is one of the most important goals in computer vision. The proposed architectures have been applied to two 3-D object and one 2-D object recognition tasks. Each image was represented by two redundant and independent sets of features (views).

8.4.1 Fruits Dataset

The fruits data set was defined in Section 7.1.1 (see Figure 7.2). Each image was divided into 3×3 overlapping sub-images. Firstly, a color histogram was extracted from each sub-image (see Section 7.1.1.1) and then the nine histograms were concatenated to form the first input feature set (V_1). The orientation histogram of an image Freeman,Coppola provides information about the directions of the edges and their intensity. Thus, an orientation histogram based on Canny edge detection was extracted from each sub-image. Then the nine histograms were concatenated to form another set of features (V_2).

8.4.2 Handwritten Digits Dataset

The StatLog handwritten digits data set was defined in Section 7.1.2 (see Figure 7.4). In our study I used only 200 images per class. Each image is represented by two views: a 40-dimensional vector that results from performing Principal Component Analysis (PCA) (see Section 7.1.2.1) and projecting the 256-dimensional vector onto the top 40 principal components (V_1). Each image in the dataset was divided into 2×2 overlapping sub-images. Then, an orientation histogram was extracted from each sub-image (see Section 7.1.1.1). The four histograms were concatenated to form the second view (V_2).

8.4.3 COIL-20 Dataset

This Columbia Object Image Library benchmark dataset was defined in Section 7.1.4 (see Figure 7.6). Each image was divided into 2×2 overlapping sub-images [60]. Firstly, a color histogram was extracted from each sub-image (see Section 7.1.1.1) and then the four histograms were concatenated to form the first input feature set (view) for classification (V_1). Then, an orientation histogram based on Sobel edge detection was extracted from each sub-image (see Section 7.1.1.2) and the four histograms were concatenated to form the second feature set (V_2).

8.5 Experimental Evaluation

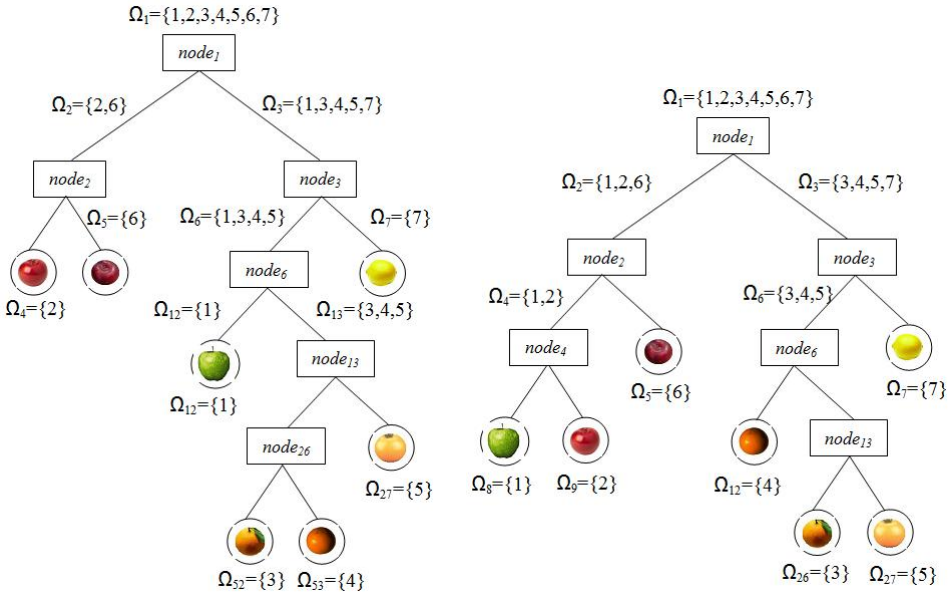
8.5.1 Methodology

An experimental study was conducted in order to evaluate the two architectures using *Co-Training*, *cotrain-of-trees* and *tree-of-cotrains*, on the three real-world recognition tasks described in Section 8.4. All experiments were carried out using WEKA library WEKA and using the *RBF Network* algorithm (see Appendix 2) as the base learning algorithm. For each experiment, 5 runs of 4-fold cross-validation have been performed to evaluate the classification accuracy of the underlying learning algorithm. The training examples are randomly divided into the labeled and unlabeled sets L and U where 20% are selected as L (18, 30 and 11 for fruits, digits and COIL-20, respectively) and the remaining training examples are used as unlabeled set U. For comparison purpose, the performance of two alternative architectures based on *Self-Training* (Section 5.3) is evaluated, which is denoted by *selftrain-of-trees* and *tree-of-selftrains*. For *selftrain-of-trees* and *tree-of-selftrains*, at each *SSL* iteration t, the base classifier at each view i selects the most confident examples $\pi_{i,t}$ and adds them to its own labeled training set L_i (no knowledge exchange between classifiers). Both *tree-of-cotrains* and *tree-of-selftrains* are based on a single class hierarchy (*Tree(V_1&V_2)*) that is generated by concatenating the two feature sets (V_1 and V_2) into a single feature vector. Unlike *tree-of-cotrains*, *tree-of-selftrains* applies *Self-Training* at each node instead of *Co-Training* using an *RBF Network* trained on the concatenation of the two views (V_1&V_2). On the other hand, both *cotrain-of-trees* and *selftrain-of-trees* based on two class hierarchies generated on each view independently.

8.5.2 Results and Discussion

The average test errors and standard deviations are shown in Table 12.3. Table 8.1(a) presents the performance in case of supervised learning when trained on the full training set ($L \cup U$) ($1^{st}Baseline$) and Table 8.1(b) presents the performance when trained on only 20% of the training set without performing *SSL* ($2^{nd}Baseline$). Table 8.1(c) and Table 8.1(d) present the test errors after the final *SSL* iteration of exploiting the unlabeled data for *selftrain-of-trees* and *cotrain-of-trees*, respectively. Table 8.1(e) and Table 8.1(f) present the test errors for *tree-of-selftrains* and Table 8.1(g) and Table 8.1(h) for *tree-of-cotrains* after 5, 10, 15, 20, 25, 30 and 35 *SSL* iterations, respectively. The results where *SSL* leads to significant improvements are marked with (*) using *corrected paired t-test* PairedTTest at 0.05 significance level.

Figures 8.4(a), 8.5(a) and 8.6(a) present a box and whisker plot for the test errors after a given iteration. The boxes have lines at the lower quartile, median, and upper quartile values. The whiskers are lines extending from each end of the boxes to show the extent of the rest of the data. Outliers are data with



(a) using color histogram (V_1) (b) using orientation histogram (V_2)

Figure 8.3: Class hierarchy for the fruits

values beyond the ends of the whiskers. Figures 8.3(a) and 8.3(b) show the class hierarchy for the fruits data sets at one of the runs based on color histograms (V_1) and orientation histograms (V_2), respectively.

From Figures 8.4(b), 8.5(b) and 8.6(b), one can conclude the following observations on *cotrain-of-trees*:

1. For all three data sets, after the final *SSL* iteration the test error of *Single-View Tree* classifier is significantly improved: 61.7%, 45.2% and 40.6% for *Tree(V_1)* and 51.6%, 21.9% and 55.6% for *Tree(V_2)*.

2. For all data sets, after the final *SSL* iteration the test error of *Multi-View Forest* classifier is significantly improved: 43.6%, 29.9% and 9.8% .

3. For all data sets, before and after *SSL* the *Multi-View Forest* performs better than its individual *Single-View Tree* classifiers due to the diversity between the tree classifiers caused by training them using different views.

4. For all data sets, the improvement achieved by *cotrain-of-trees* is more than the improvement gained by *selftrain-of-trees* due to the knowledge exchange between pairs of cotrained tree-structured ensembles.

In addition, one can observe the following findings on *tree-of-cotrains*:

1. The *tree-of-cotrains* can exploit unlabeled data to improve the test error on *fruits* and *digits* while on *COIL* data set the performance is degenerated.

2. The error improvement achieved by *tree-of-cotrains* is less than the improvement achieved by *cotrain-of-trees*. This is attributed to the fact that *tree-of-cotrains* uses a single class hierarchy which reduce the benefits of *multi-view learning* and knowledge exchange between cotrained classifiers. In contrast to *cotrain-of-trees* that uses a different class hierarchy for each view (for example, see Figures 8.3(a) and 8.3(b)).

3. The *tree-of-cotrains* performs comparable to *tree-of-selftrains*. That is attributed to the fact that both architectures use a single class hierarchy. Like *tree-of-selftrains*, the confidence in *tree-of-cotrains* is measured by a single classifier while it is measured by a committee of classifiers in *cotrain-of-trees*.

8.6 Related Work

8.6.1 Tree-Structured Approach and Margin Trees

In the margin tree algorithm [182], a class hierarchy is constructed by hierarchical agglomerative clustering (HAC) where margins between pairs of classes are used as distance measures for clustering of (meta-)classes. There are three different ways to define the margin: greedy, complete-linkage and single-linkage. Then a total of K - 1 internal nodes will be created with K leaf nodes, same as in BHC. As opposite to BHC, in the margin tree algorithm, it is assumed that the dimensionality is always greater than the number of samples, so that the samples are always linearly separable by a maximum-margin hyperplane. If the samples are not linearly separable, using non-linear kernels such as radial basis function to make the samples separable in a higher dimensional space leads to more difficult interpretation of margins, and makes the class hierarchy more sensitive to the kernel parameters.

In [90], Jun and Ghosh tried to solve the problem of small sample size that occurs during the class hierarchy generation of BHC. It is worth mentioning that the lower the position of a node at the tree, the less sample size it will have for training. They proposed a hybrid approach that combine the merits of BHC framework and margin trees. That is, at each node they check the available sample size. If number of instances is less than the number of features, the margin tree algorithm is employed instead of BHC. While BHC algorithm is applied if the samples are not guaranteed to be linearly separable. Both *cotrain-of-trees* and *tree-of-cotrains* also deal with the problem of small sample size but they exploit the unlabeled data to increase the sample size.

8.6.2 Multi-Class Decomposition and *SSL*

Ghani [69] investigated the combination of *ECOC* and *Co-Training* in order to decompose the multi-class text classification tasks using *ECOC* then to apply

Table 8.1: Mean and standard deviation of the test error for the three recognition tasks

(a) for *supervised learning (*100% Labeled*)*

classifier	*Tree(V_1)*	*Tree(V_2)*	*Forest*	*Tree(V_1&V_2)*
Fruits	4.83% ± 1.55	9.52% ± 1.98	2.64% ± 1.22	1.89% ± 1.05
Digits	11.72% ± 1.46	17.09% ± 1.40	8.89% ± 1.19	8.81% ± 1.38
COIL-20	4.79% ± 1.39	4.58% ± 1.06	1.67% ± 0.69	0.66% ± 0.40

(b) for *supervised learning (*20% Labeled*)*

classifier	*Tree(V_1)*	*Tree(V_2)*	*Forest*	*Tree(V_1&V_2)*
Fruits	10.93% ± 3.45	15.26% ± 2.52	6.72% ± 2.69	5.53% ± 2.55
Digits	18.41% ± 2.38	20.86% ± 1.47	12.44% ± 1.29	12.17% ± 1.40
COIL-20	9.93% ± 2.49	14.26% ± 2.53	6.22% ± 1.70	6.52% ± 1.90

(c) for *selftrain-of-trees* (*20% Labeled + Unlabeled Data*)

classifier	*Tree(V_1)*	*Tree(V_2)*	*Forest*
Fruits	10.81% ± 2.42	13.05% ± 3.37	5.79% ± 1.97
Digits	11.74% ± 1.87 *	18.84% ± 1.73 *	9.29% ± 1.14 *
COIL-20	9.29% ± 2.02	12.69% ± 2.69	6.07% ± 1.62

(d) for *cotrain-of-trees* (*20% Labeled + Unlabeled Data*)

classifier	*Tree(V_1)*	*Tree(V_2)*	*Forest*
Fruits	4.19% ± 2.19 *	7.38% ± 2.17 *	3.79% ± 1.64 *
Digits	10.09% ± 1.46 *	16.29% ± 1.48 *	8.72% ± 1.15 *
COIL-20	5.90% ± 2.02 *	6.32% ± 2.01 *	5.61% ± 1.98

(e) for *tree-of-selftrains* (*20% Labeled + Unlabeled Data*)

classifier	*Tree(V_1&V_2)-5*	*Tree(V_1&V_2)-10*	*Tree(V_1&V_2)-15*	*Tree(V_1&V_2)-20*
Fruits	5.60% ± 2.74	5.48% ± 2.82	5.82% ± 3.01	5.79% ± 2.87
Digits	11.62% ± 1.71	11.40% ± 1.53	11.33% ± 1.50	10.85% ± 1.67 *
COIL-20	6.84% ± 2.01	7.45% ± 2.01	7.78% ± 1.83	8.31% ± 1.88

(f) for *tree-of-selftrains* (*20% Labeled + Unlabeled Data*)

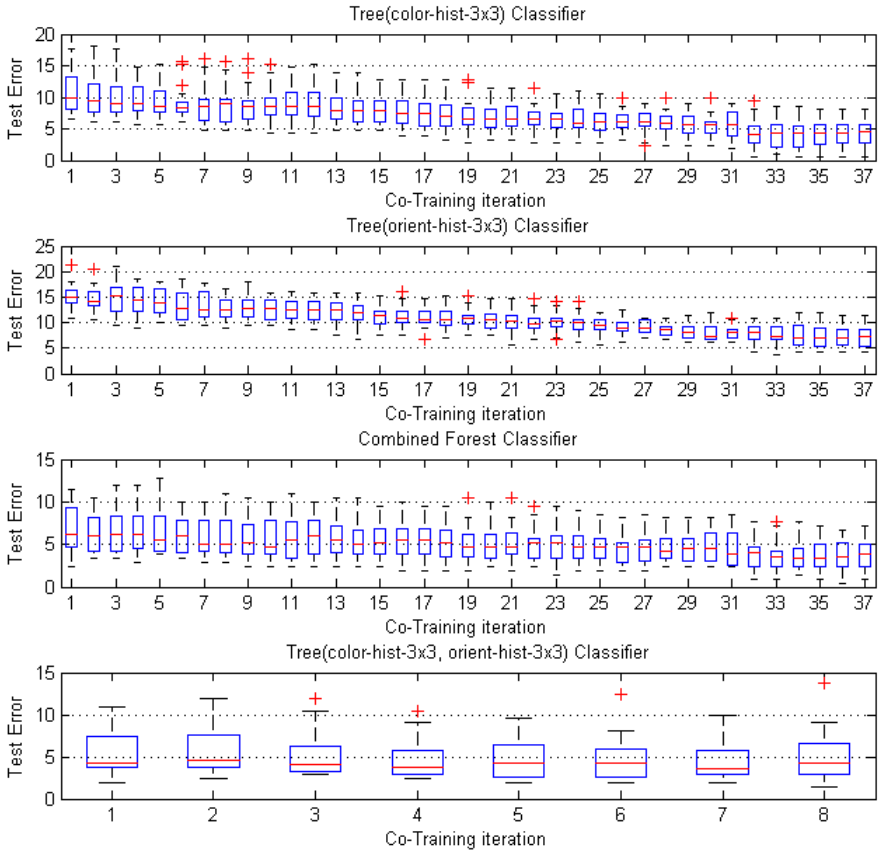
classifier	*Tree(V_1&V_2)-25*	*Tree(V_1&V_2)-30*	*Tree(V_1&V_2)-35*
Fruits	5.67% ± 2.66	5.27% ± 2.90	5.36% ± 3.48
Digits	11.01% ± 1.34	10.77% ± 1.22 *	10.83% ± 1.58
COIL-20	8.42% ± 2.17	–	–

(g) for *tree-of-cotrains* (*20% Labeled + Unlabeled Data*)

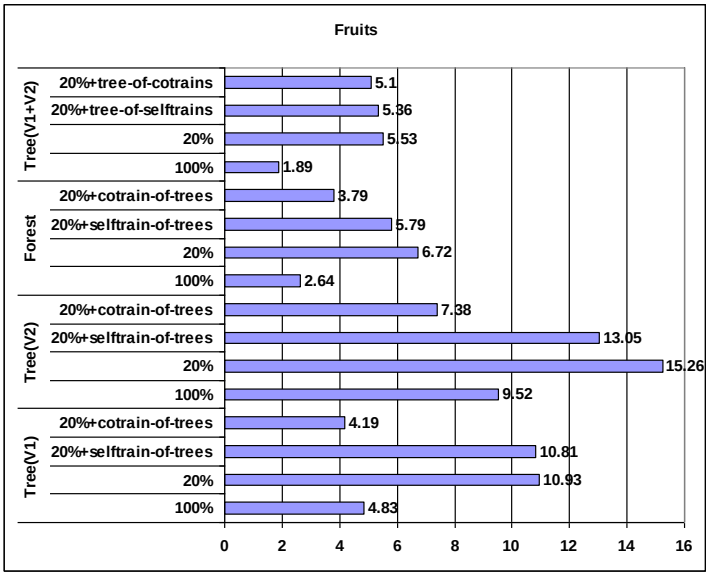
classifier	*Tree(V_1&V_2)-5*	*Tree(V_1&V_2)-10*	*Tree(V_1&V_2)-15*	*Tree(V_1&V_2)-20*
Fruits	5.77% ± 2.93	5.29% ± 2.68	4.77% ± 2.40	4.60% ± 2.22
Digits	11.40% ± 1.42	10.99% ± 1.70	10.43% ± 1.43 *	10.26% ± 1.18 *
COIL-20	7.10% ± 2.14	7.03% ± 1.63	7.88% ± 2.51	7.75% ± 2.51

(h) for *tree-of-cotrains* (*20% Labeled + Unlabeled Data*)

classifier	*Tree(V_1&V_2)-25*	*Tree(V_1&V_2)-30*	*Tree(V_1&V_2)-35*
Fruits	4.70% ± 2.54	4.34% ± 2.07	5.10% ± 3.01
Digits	10.12% ± 1.60 *	10.12% ± 1.43 *	10.04% ± 1.34 *
COIL-20	8.09% ± 2.46	–	–

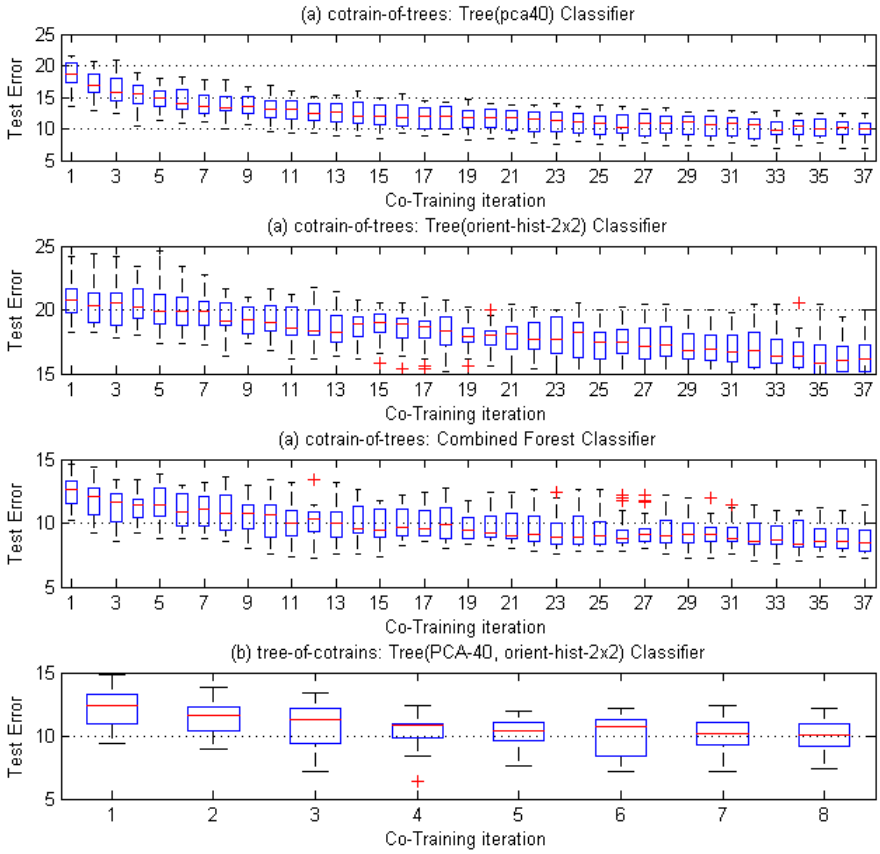


(a) box plot

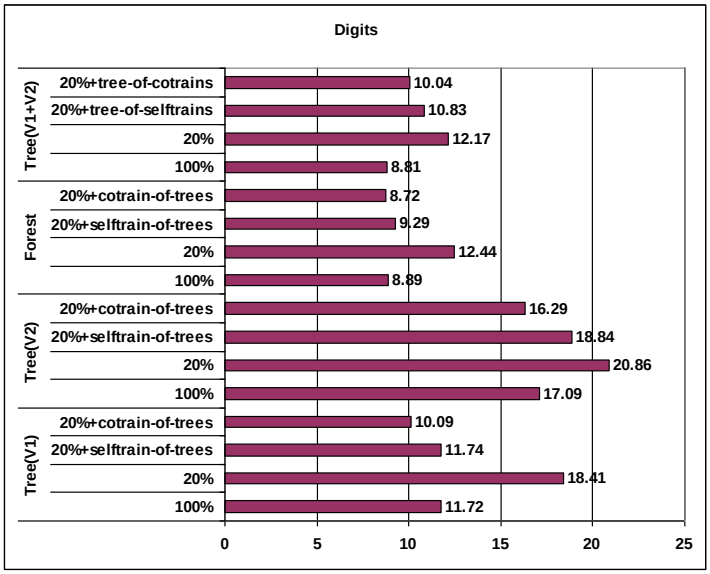


(b) bar graph

Figure 8.4: Test error for fruits data set

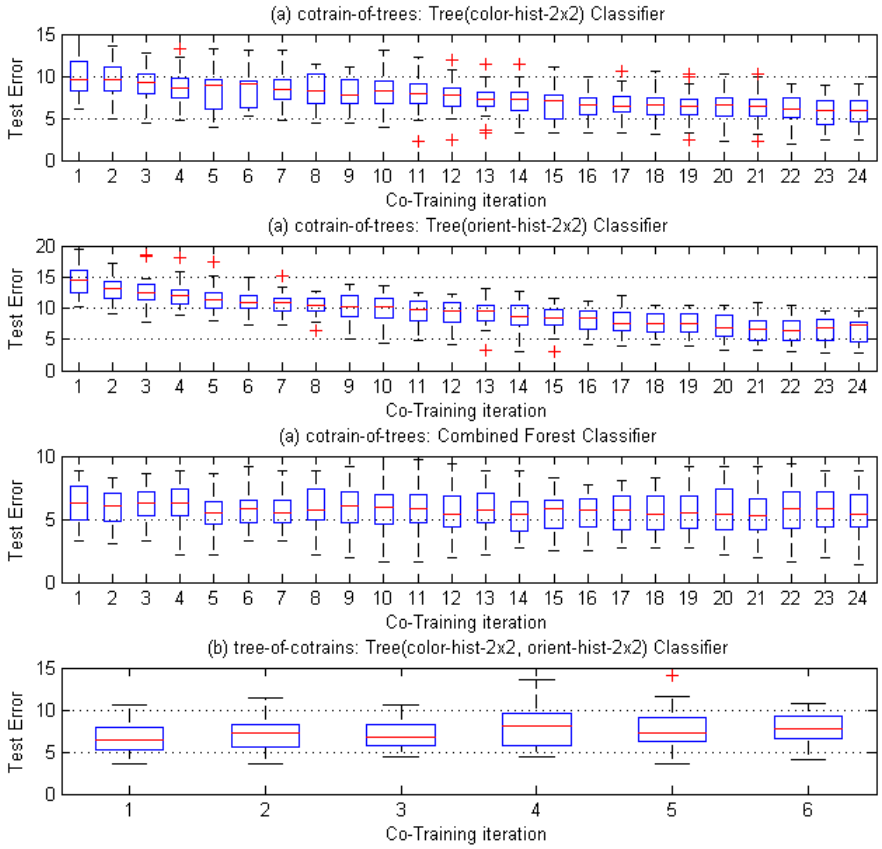


(a) box plot

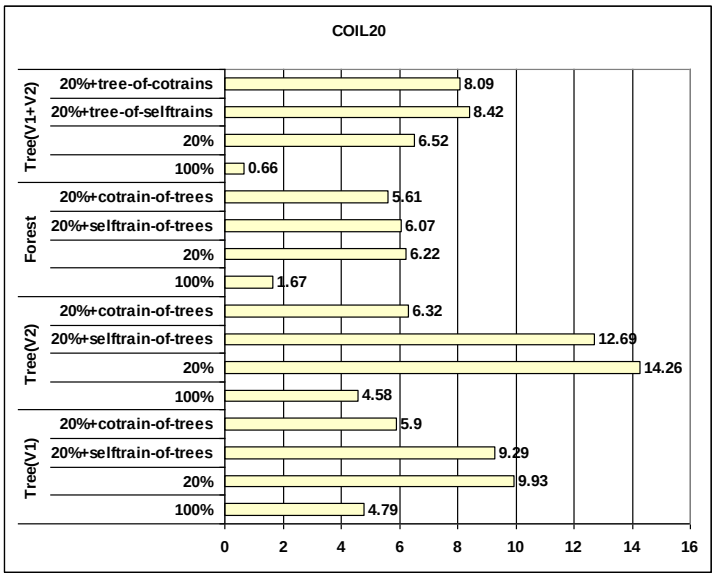


(b) bar graph

Figure 8.5: Test error rate for handwritten digits data set



(a) box plot



(b) bar graph

Figure 8.6: Test error rate for COIL data set

Co-Training for each binary problem to exploit the unlabeled text documents to improve the classification performance. The results have shown that this hybrid approach outperforms both standalone *ECOC* and *Co-Training* using Naive Bayes as binary text classifier.

8.6.3 Tree-Structured Approach and Boosting

In [91], a novel multi-class boosting algorithm, *AdaBoost.BHC*, is proposed. First the tree-structured approach is used to decompose the multi-class problem into a set of binary problems then an ensemble of binary classifiers is constructed, by the popular *AdaBoost* ensemble method (Section [65]), to solve each binary problem instead of depending on a single binary classifier. Empirical comparisons of *AdaBoost.BHC* and other existing variants of multi-class *AdaBoost* algorithm are carried out using seven multi-class datasets from the UCI machine learning repository. Not only *AdaBoost.BHC* is faster than other *AdaBoost* variants but also it achieves lower error rates. Like *AdaBoost.BHC*, *tree-of-cotrains* constructs an ensemble of binary classifiers but the objective is to exploit the unlabeled examples to improve the classification performance.

8.6.4 Tree-Structured Approach and Neural Combiners

In [3], Abdel Hady and Schwenker introduced a trainable fusion method that integrates statistical information about the individual classifier outputs (clustered decision templates) into a Radial Basis Function (RBF) network. The neural combination model was compared with the decision templates combiner and the existing non-trainable tree ensemble fusion methods: classical decision tree-like approach (see Section 4.6.2.1), product of the unique path and Dempster-Shafer evidence theory based method (see Section 14.3.1). The experiments have shown that the RBF Network tree combiner significantly outperforms the three existing nontrainable tree combiners and the decision templates combiner proposed by Kuncheva. This neural combiner is shown to be robust to changes in the training set size and the number of decision templates per class.

8.7 Conclusions

The main objective of this chapter is to show that there is an improvement from using unlabeled data when training tree-structured (hierarchical) ensembles. I proposed two learning architectures to combine the benefits of *Co-Training* algorithm and the tree-structured multi-class decomposition approach, which are denoted by (*cotrain-of-trees* and *tree-of-cotrains*). To study the influence of *multi-view learning*, I replaced *Co-Training* with *Self-Training* in the two architectures (*selftrain-of-trees* and *tree-of-selftrains*). I have the following conclusions:

- It was shown that *cotrain-of-trees* achieves performance improvement more than *selftrain-of-trees* and *tree-of-cotrains* performs comparable to *tree-of-selftrains*. That is, *Co-Training* using two class hierarchies generated based on different views (*cotrain-of-trees*) benefits from *multi-view learning* more than *Co-Training* using a single class hierarchy generated based on the concatenation of both views into a single feature set (*tree-of-cotrains*).

- These results emphasize the conclusion of Gupta and Dasgupta in [74] that using the individual views independently each based on its distance measure works better than combining the two distance measures or concatenating the feature vectors of the different views into a single input vector. This preserves the classes separation in each individual view.

- An important factor that influence the performance of any *Co-Training* style algorithm is how to measure the confidence on predicting the label of an unlabeled example which determine its probability of being selected. The results shows that the evidence-theoretic tree combination method can provide effective estimates of class probabilities that are used by *Co-Training* to measure confidence.

8.8 Future Work

The following are directions for further investigation:

- Although *RBF networks* are used as binary classifiers within the tree ensemble, both architectures are applicable to any other type of classifiers such as support vector machines.

- Motivated by the empirical results provided in [3] that *trainable* neural combiner outperforms the *fixed* evidence-theoretic combiner, I have the following open questions. First, can *cotrain-of-trees* and *tree-of-cotrains* using the neural combiner outperform their current implementation based on evidence-theoretic combiner? Second, can semi-supervised learning exploit the unlabeled data to improve the trainable neural combiner performance as it improves the hierarchical classifiers?

- Active learning (selective sampling) algorithms are used to select the most informative examples from a given unlabeled data set as labeled training examples. *Co-Testing* [134] is a *multi-view* active learning framework that is inspired by *Co-Training*. The application of *Co-Testing* instead of the current random sampling, is an open issue that deserve investigation. That is, *Co-Testing* can be used with a smaller randomly-selected labeled training set L' (e.g., 5% of training examples) to train initial classifiers. Then it can

iteratively select the *most informative examples* with respect to these classifiers (e.g., 15%). These selected examples are labeled by human experts and added to the labeled training set. Finally this augmented data set will be the starting point for *cotrain-of-trees* and *tree-of-cotrains*.

Appendix 1: Online Tree Ensemble Learning

The proposed online version of tree learning algorithm ($OnlineTreeLearn$) takes as input an existing tree ensemble and a set of new training examples R. The algorithm returns an updated tree ensemble that reflect the new examples (See Algorithm 11). Such algorithms have advantages over typical batch algorithms in situations where data arrive continuously which is the case for *Co-Training*. They need only one pass through each training example unlike the batch algorithms that require multiple passes which would require a prohibitively large training time. At each iteration, I keep the initially generated class hierarchy and just update the internal node classifiers with the newly-labeled training examples using an incremental base learning algorithm (see Algorithm 13). If one uses the batch version (See Section 4.6), the current tree will be discarded and a class hierarchy will be generated from scratch with the augmented training set.

Algorithm 11 Online Tree Ensemble Learning Algorithm

Require: set of n newly-labeled training examples (R), online base learning algorithm ($OnlineBaseLearn$), the class hierarchy generated before ($hierarchy$)

1: **for** each binary classifier h_i at $node_j$, ($i \in \{1, \ldots, K-1\}$) in $hierarchy$ **do**
2: Filter the new training set R as follows:
$R_j \leftarrow \{(x,t)|(x,y) \in R$ and $t = 1$ if $y \in \Omega_{2j}$ and $t = 2$ if $y \in \Omega_{2j+1}\}$
3: Update binary classifier,
$h_i = OnlineBaseLearn(h_i, R_j)$ (See Algorithm 13)
4: **end for**

Appendix 2: Binary RBF Network Learning

The two-phase learning algorithm discussed in Section 2.1.2 is used for training RBF networks at the internal nodes (see Algorithm 12). The *multivariate Gaussian radial basis function* ϕ_j is used as an activation function at each hidden node of the network. At the first phase, the RBF centers are determined by applying class-specific c-means clustering algorithm MacQueen67. It is assumed that all the Gaussians are radially symmetric, therefore the Euclidean distance between a prototype c_j and the nearest prototype multiplied by α is used as the width of the

j^{th} RBF neuron (σ_j) where α controls the extent of overlap between a Gaussian function and its nearest neighbor (in this experiments, α=1.0 and c=10). At the second phase, the output layer weights W are computed by minimizing the MSE at the network output (over the m training instances) by a matrix pseudo-inverse technique using singular value decomposition.

Algorithm 12 Binary RBF Network Learning

Require: set of m labeled training examples ($L = \{(x_i, y_i)|x_i \in \mathbb{R}^D, y_i \in \{1, 2\}, i = 1, \ldots, m\}$), number of RBF neurons per class (c), a parameter controls the width of an RBF (α)

Training Phase

{Calculate the RBF centers}

Set $C = \emptyset$
for each class $k \in \{1, 2\}$ **do**
  X_k = set of examples belonging to class k
  $C_k = \{(\mu_j, k)\}_{j=1}^{c} = c\text{-}means(X_k, c)$
  $C = C \cup C_k$, add the new clusters
end for

{Calculate the RBF widths}

for each prototype $(\mu_j, k_j) \in C$ **do**
  $\sigma_j = \alpha \min\left\{\|\mu_j - \mu_i\|_2 : (\mu_i, k_i) \in C, i \neq j, k_i \neq k_j\right\}$
end for
Define the Gaussian radial basis activation function:
$\phi_j(x; \mu_j, \sigma_j) = exp(-\frac{\|x-\mu_j\|_2^2}{2\sigma_j^2})$

{Calculate the output layer weights}

for each training example $(x_i, y_i) \in L$ **do**
  Get activation vector $\Phi_{ji} = \phi_j(x_i; \mu_j, \sigma_j)$
  Get target output vector $T_{ik} = I(k = y_i)$
end for
Calculate Φ^+, the pseudo-inverse of Φ
Set $W = \Phi^+ T$ where T is the target matrix and Φ is the activation matrix.

Prediction Phase

for each class $k \in \{1, 2\}$ **do**
  $\hat{y}_k = \sum_{j=1}^{2\times c} w_{jk}\phi_j(x; \mu_j, \sigma_j)$
  if $\hat{y}_k < 0$ **then** $\hat{y}_k \leftarrow 0$ **end if**
end for
if $\hat{y}_1 + \hat{y}_2 > 1$ **then**
  $\hat{y}_k = \hat{y}_k/(\hat{y}_1 + \hat{y}_2)$ for each class $k \in \{1, 2\}$
end if
return the class probability distribution $\{\hat{y}_1, \hat{y}_2\}$ for a given instance x

Appendix 3: Online RBF Network Learning

Co-Training is an incremental learning algorithm, because at each iteration new examples are added to the labeled training set of the underlying RBF Network. Since the c-means clustering algorithm is sensitive to the selection of initial prototypes and to ensure a stable behaviour for the learning curve during *Co-Training*, the underlying RBF Networks should be trained using an incremental learning algorithm. Motivated by this argument, an online version of RBF network learning algorithm (see Algorithm 13) is presented. The seeded c-means clustering algorithm [19] is used. Thus, instead of randomly initializing prototypes, the mean of the j^{th} cluster is initialized with the mean of the existing j^{th} RBF neuron.

Algorithm 13 Online Binary RBF Network Learning

Require: set of n newly labeled examples ($R = \{(x_i, y_i) | x_i \in \mathbb{R}^D, y_i \in \{1, 2\}, i = 1, \ldots, n\}$)

for each example $(x_i, y_i) \in R$ **do**
  $updated = true$
  $X = X \cup \{(x_i, y_i)\}$, add the new instance
  $d = \min\{\|c_j - x_i\|_2 : c_j \in C\}$
  if $d > \sigma_j$ **then**
    $update \leftarrow false$
    $C = C \cup \{(x_i, y_i)\}$, add a new cluster
    {Update the RBF centers}
    C_{y_i} = set of clusters belonging to class y_i
    $C = C - C_{y_i}$,
    X_{y_i} = set of examples belonging to class y_i
    $C_{y_i} = seeded\text{-}c\text{-}means(X_{y_i}, C_{y_i})$
    $C = C \cup C_{y_i}$, add the new clusters
    Calculate the RBF widths (see Algorithm 12)
    Calculate the output weights (see Algorithm 12)
  end if
end for
if $update = true$ **then**
  Calculate the RBF centers (see Algorithm 12)
  Calculate the RBF widths (see Algorithm 12)
  Calculate the output weights (see Algorithm 12)
end if

Chapter 9

Co-Training by Committee for Semi-supervised Classification

9.1 Introduction

Many data mining applications such as content-based image retrieval [212], computer-aided medical diagnosis [119], object detection and tracking [116], web page categorization [140], or e-mail classification [96], there is often an extremely large amount of data but labeling data is usually difficult, expensive, or time consuming, as it requires human experts for annotation. *Semi-supervised learning* (Chapter 5) addresses this problem by using unlabeled data together with labeled data in the training process. *Co-Training* (Section 5.7.1.1) is a popular *semi-supervised learning* algorithm that requires each example to be represented by multiple sets of features (views) where these views are sufficient for learning and independent given the class. However, these requirements are hard to be satisfied in many real-world domains because there are not multiple representations available or it is computationally inefficient to extract more than one feature set for each example.

In this chapter, a single-view variant of *Co-Training*, called *Co-Training by Committee* (*CoBC*), is proposed, in which an ensemble of diverse classifiers is used instead of redundant and independent views required by the conventional *Co-Training* algorithm. The aim of *CoBC* is to exploit the unlabeled data to improve the recognition rate of the underlying supervised ensemble learning algorithm and to minimize the cost of data labeling. The method used to measure the confidence in predicting the class label of an unlabeled example is an important factor for the success of any *Co-Training* style algorithm. Although the confidence method depends on class probability estimates, many classifier types can not provide an accurate class probability estimates. Thus, a new method is introduced to measure the confidence that is based on estimating the local accuracy of the committee members on the neighborhood of a given unlabeled example. The work in this chapter has been previously published ([5, 4]).

9.2 Co-Training by Committee (*CoBC*)

The pseudo-code of the *CoBC* framework is given in Algorithm 14 and illustrated in Figure 9.1. Let $L = \{(x_\mu, y_\mu)|x_\mu \in \mathbb{R}^D, y_\mu \in \Omega, \mu = 1, \ldots, m\}$ be the set of labeled training examples where each example is described by a D-dimensional feature vector $x_\mu \in \mathbb{R}^D$, y_μ denotes the class label of x_μ and $\Omega = \{\omega_1, \ldots, \omega_K\}$ is the set of target classes (ground truth). Also let $U = \{x_u|u = 1, \ldots, n\}$ be the set of unlabeled data. *CoBC* works as follows: firstly the class prior probabilities are determined then an initial committee of N diverse accurate classifiers $H^{(0)}$ is trained on L using the given ensemble learning algorithm *EnsembleLearn* and base learning algorithm *BaseLearn*. Then the following steps are repeated until the maximum number of iterations T is reached or U becomes empty. For each iteration t and for each classifier i, a set $U'_{i,t}$ of u examples drawn randomly from U without replacement. It is computationally more efficient to use $U'_{i,t}$ instead of using the whole set U.

The method *SelectCompetentExamples* (see Algorithm 15) is applied to estimate the competence of each unlabeled example in $U'_{i,t}$ given the companion committee $H_i^{(t-1)}$. Note that $H_i^{(t-1)}$ is the ensemble of all base classifiers trained in the previous iteration except $h_i^{(t-1)}$. A set $\pi_{i,t}$ is created that contains the n_c most competent examples assigned to each class ω_c. Then $\pi_{i,t}$ is removed from $U'_{i,t}$ and inserted into the set L'_t that contains all the examples labeled at iteration t. The remaining examples in $U'_{i,t}$ are returned to U. We have two options: (1) if

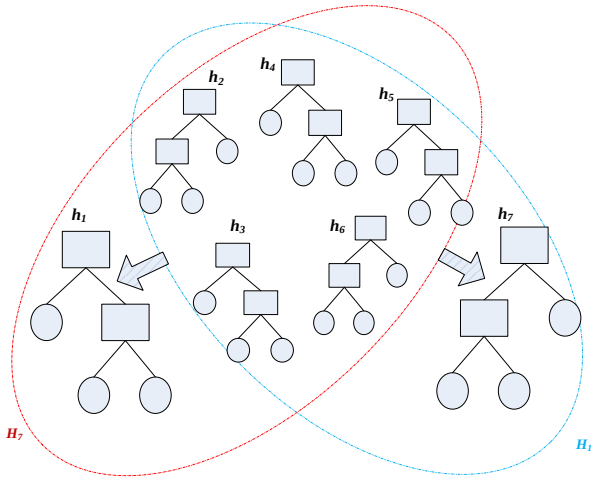


Figure 9.1: Graphical Illustration of CoBC

the underlying ensemble learner depends on training set perturbation to promote diversity, then insert $\pi_{i,t}$ only into L_i. Otherwise, $h_i^{(t)}$ and $h_j^{(t)}$ $(i \neq j)$ will be identical because they are refined with the same newly labeled examples. This will degrade the ensemble diversity and therefore degrades the relative improvement expected due to exploiting the unlabeled data. One can observe that if the

Algorithm 14 Pseudo code of *CoBC* for classification

Require: set of labeled training examples (L), set of unlabeled training examples (U), maximum number of iterations (T), ensemble learning algorithm ($EnsembleLearn$), base learning algorithm ($BaseLearn$), ensemble size (N), number of unlabeled examples in the pool (u), number of nearest neighbors (k), sample size (n), number of classes (C) and an initial committee ($H^{(0)}$)

Training Phase

1: Get the class prior probabilities, $\{Pr_c\}_{c=1}^{C}$
2: Set the class growth rate, $n_c = n \times Pr_c$ where $c = 1, \ldots, C$
3: **if** $H^{(0)}$ is not given **then**
4: Construct an initial committee of N classifiers, $H^{(0)} = EnsembleLearn(L, BaseLearn, N)$
5: **end if**
6: **for** $t \in \{1, \ldots, T\}$ **do**
7: $L'_t \leftarrow \emptyset$
8: **if** U is empty **then** $T = t$-1 and abort loop **end if**
{Get most confident examples ($\pi_{i,t}$) using companion committee $H_i^{(t-1)}$ }
9: **for** $i \in \{1, \ldots, N\}$ **do**
10: $U'_{i,t} \leftarrow RandomSubsample(U, u)$
11: $\pi_{i,t} \leftarrow SelectCompetentExamples(i, U'_{i,t}, H_i^{(t-1)}, k, \{n_c\}_{c=1}^{C}, C)$
12: $L'_t \leftarrow L'_t \cup \pi_{i,t}$, $U'_{i,t} \leftarrow U'_{i,t} \setminus \pi_{i,t}$ and $U \leftarrow U \cup U'_{i,t}$
13: **end for**
14: **if** L'_t is empty **then** $T = t$-1 and abort loop **end if**
{Re-train the N classifiers using their augmented training sets }
15: **for** $i \in \{1, \ldots, N\}$ **do**
16: $L_i = L_i \cup L'_t$
17: $h_i^{(t)} = BaseLearn(L_i)$
(for incremental learning, $h_i^{(t)} = BaseLearn(h_i^{(t-1)}, L'_t)$)
18: **end for**
19: **end for**

Prediction Phase

20: **return** $H^{(T)}(x) = \frac{1}{N} \sum_{i=1}^{N} h_i^{(T)}(x)$ for a given example x

ensemble members are identical, *CoBC* will degenerate to *Self-Training*. (2) If ensemble learner employs another source of diversity, then it is not a problem to insert $\pi_{i,t}$ into the training sets of all classifiers as shown in step 16. Then, *CoBC* does not recall $EnsembleLearn$ but only the N committee members are retrained using their updated training sets L_i. It is worth noting that: (1) *CoBC* can improve the recognition rate only if the most confident examples with respect to the companion committee H_i are informative examples with respect to h_i. (2) Although *CoBC* selects the most confident examples, adding mislabeled examples

to the training set (*noise*) is unavoidable but the negative impact of this *noise* could be compensated by augmenting the training set with sufficient amount of newly labeled examples.

9.2.1 Complexity of *CoBC*

The time complexity of the *CoBC* algorithm is $O(TNg(BaseLearn))$ where the term $g(BaseLearn)$ represents the complexity of the underlying base learning algorithm ($BaseLearn$) which depends on the number of training examples in L_i. On way to improve the complexity of *CoBC* is to reduce the ensemble size N through selecting the most accurate and diverse classifiers such as information-theoretic approach defined in Chapter 15. Another way is to reduce $g(BaseLearn)$ through using an incremental version of $BaseLearn$. In each *CoBC* iteration, each ensemble member h_i is updated with the newly-labeled examples $\pi_{i,t}$ instead of retrain it with the whole training set L_i. For instance, a method to improve the complexity of *RSM* with *kNN* is introduced in Section 9.2.4.

9.2.2 Confidence Measure

An important factor that affects the performance of any *Co-Training* style algorithm is how to measure the confidence about the labeling of an unlabeled example which determines its probability of being selected. An inaccurate confidence measure leads to adding mislabeled examples to the labeled training set which leads to performance degradation during the *SSL* process. For *CoBC*, it is assumed that the underlying ensemble employs soft combiner (Section 3.3.2) in order to provide a class probability distribution, that is $H^{(t-1)} : \mathbb{R}^D \times \Omega \rightarrow [0, 1]$.

9.2.2.1 Estimating Class Probabilities

Many classifiers can provide class probability estimates (*CPE*) such as Naive Bayes classifier or return real-valued outputs that can be transformed to *CPE*s such as neural networks and decision trees. If a classifier estimates the probability that an example x_1 belongs to classes ω_1 and ω_2 is 0.9 and 0.1, respectively, while that for an example x_2 is 0.6 and 0.4, respectively, then the classifier is more confident that x_1 belongs to class ω_1 than x_2. Therefore, the confidence in predicting the class label of an unlabeled example x_u by $H_i^{(t-1)}$ is,

$$Confidence(x_u, H_i^{(t-1)}) = \max_{1 \leq c \leq C} H_i^{(t-1)}(x_u, \omega_c) \tag{9.1}$$

Unfortunately, many classifiers do not provide an accurate *CPE*. For instance, traditional decision tree partitions the input space into regions and provides piecewise constant probability estimates. That is, all unlabeled examples x_u which lie into a particular leaf node (region), will have the same *CPE*s because the *CPE*

depends on class frequencies and not on the exact value of x_u. This suffers from high bias and high variance. The bias is high because tree learner tries to make leaves homogeneous (pure), therefore, the class probabilities are shifted toward zero or one. The variance is high because when the number of examples per leaf is small, the class probabilities are unreliable. Therefore, this leads to unreliable *CPE* by the companion committee.

The following example illustrates the potential problem with probability estimation provided by decision trees. Assume that a decision tree defines two regions R_1 and R_2. If R_1 comprises a subset of 100 training examples, 90 of which are one class (let it be the positive class), then during classification, any unlabeled example x_u^1 that falls into R_1 is assigned the positive class with a probability of 0.9 (90/100). If R_2 contains only 3 training examples, all of which belongs to the positive class and an example x_u^2 lies into R_2, then the probability estimator gives an estimate of 1.0 (3/3) that x_u^2 will be positive. Probably the evidence based on 3 examples for such a strong statement is not strong enough compared to the evidence based on 90 examples. That is, the region R_2 will be more confident than R_1. Smoothing of probability estimates is a partial solution to this problem. For instance, the Laplace estimate calculates the estimated probability as $\frac{n_k+1}{n+C}$ while the frequency estimate yields $\frac{n_k}{n}$. Therefore, for a two-class problem the Laplace estimate yields a probability of $\frac{3+1}{3+2} = 0.8$ for x_u^2 and $\frac{90+1}{100+2} = 0.89$ for x_u^1. That is, the region R_1 will be more confident than R_2. Laplace correction solves part of the problem but still the tie between the examples within the same region can not be broken. Note that a lot of work such as [179, 150, 120] has addressed the problem of improving the probability-based ranking provided by decision trees.

9.2.2.2 Estimating Local Competence

We introduce a new confidence measure as shown in Algorithm 15. Our motivation is to compensate the inaccurate probability-based ranking provided by traditional decision trees. This measure depends on estimating the companion committee accuracy on labeling the neighborhood of an unlabeled example x_u. This local accuracy represents the probability that the companion committee correctly predicts the class label of x_u. The local competence of an unlabeled example x_u given a companion committee $H_i^{(t-1)}$ can be defined as follows:

$$Comp(x_u, H_i^{(t-1)}) = \sum_{\substack{(x_n,y_n)\in N_k(x_u) \\ y_n=\hat{y}_u}} W_n.H_i^{(t-1)}(x_n, \hat{y}_u) \tag{9.2}$$

where

$$W_n = \frac{1}{||x_n - x_u||_2 + \epsilon}, \tag{9.3}$$

$$\hat{y}_u = arg \max_{1\leq c\leq C} H_i^{(t-1)}(x_u, \omega_c), \tag{9.4}$$

$H_i^{(t-1)}(x_n, \hat{y}_u)$ is the probability given by $H_i^{(t-1)}$ that neighbor x_n belongs to the same class assigned to x_u ($\hat{y}_u$), W_n is the reciprocal of the Euclidean distance between x_u and its neighbor x_n and ϵ is a constant added to avoid zero denominator. The neighborhood could also be determined using a separate validation

Algorithm 15 Pseudo Code of the *SelectCompetentExamples* method

Require: pool of unlabeled examples ($U'_{i,t}$), the companion committee of classifier $h_i^{(t-1)}$ ($H_i^{(t-1)}$), number of nearest neighbors k, growth rate ($\{n_c\}_{c=1}^{C}$) and number of classes (C)

$\pi_{i,t} \leftarrow \emptyset$
for each class $\omega_c \in \{\omega_1, \ldots, \omega_C\}$ **do**
  $count_c \leftarrow 0$
end for
for each $x_u \in U'_{i,t}$ **do**
  $H_i^{(t-1)}(x_u) = \frac{1}{N-1} \sum_{j=1,\ldots,N, j\neq i} h_j^{(t-1)}(x_u)$
  Apply the companion committee $H_i^{(t-1)}$ to x_u,
  $\hat{y}_u \leftarrow arg\max_{1\leq c\leq C} H_i^{(t-1)}(x_u, \omega_c)$
  Find the k nearest neighbors of x_u,
  $N_k(x_u) = \{(x_n, y_n) | (x_n, y_n) \in Neighbors(x_u, k, L)\ \}$
  Calculate $Comp(x_u, H_i^{(t-1)})$ as defined in Eq. (9.2) and Eq. (9.3)
end for
Rank the examples in $U'_{i,t}$ based on competence (in descending order)
{Select the n_c examples with the maximum competence for class ω_c}
for each $x_u \in U'_{i,t}$ **do**
  if $Comp(x_u, H_i^{(t-1)}) > 0$ and $count_{\hat{y}_u} < n_{\hat{y}_u}$ **then**
    $\pi_{i,t} = \pi_{i,t} \cup \{(x_u, \hat{y}_u)\}$ and $count_{\hat{y}_u} = count_{\hat{y}_u} + 1$
  end if
end for
return $\pi_{i,t}$

set (a set of labeled examples that is not used for training the classifiers), but it may be impractical to spend a part from the small-sized labeled data for validation. To avoid the inaccurate estimation of local accuracy that may result due to overfitting, the newly-labeled training examples $\pi_{i,t}$ will not be involved in the estimation. That is, only the initially (manually) labeled training examples are taken into account. Then, the set $N_k(x_u)$ is defined as the set of k nearest labeled examples to x_u.

The local competence assumes that the actual data distribution satisfies the well-known cluster assumption: examples with similar inputs should belong to the same class. Therefore, the local competence of x_u is zero if there is not any neighbor belongs to the predicted class label $\hat{y}_u$ which contradicts the cluster

assumption. Therefore, one can observe that $\hat{y}_u$ is incorrect class label of x_u ($\hat{y}_u \neq y_u$). In addition, the local competence increases as the number of neighbors that belong to $\hat{y}_u$ increases and as the distances between these neighbors and x_u decreases.

9.2.3 Random Subspace Method (*RSM*)

Many classification problems involve a high dimensional input space and a limited amount of training data available which leads to the problem known as *curse of dimensionality*. Therefore, it is necessary to increase the quantity of labeled training data or to reduce the size of the input space. For the first solution, semi-supervised and active learning (see Chapter 10) is used and for the second solution, the random subspace method is adopted. In [82, 81], the experiments have shown that both multiple decision trees [82] and *nearest-neighbor* classifiers [81] constructed in randomly selected subspaces can be combined to achieve accuracy better than the classifier constructed in the original feature space. In contrast to the methods that suffer due to the curse of dimensionality, *RSM* effectively takes advantages of high dimensionality by constructing a set of classifiers that are mutually independent to a certain extent. Since random subsets of the original feature set are used, *RSM* works only for problems with a relatively large number of features such as image and speech recognition. For problems with smaller number of features, the number of features can be increased with certain simple functions of the original features (e.g. pairwise sums, differences, or products). In addition, the features must be redundant otherwise the output classifiers will be very weak as they are trained with small random subsets of the features.

9.2.4 *RSM* with *kNN*

The calculation of nearest neighbors involves two steps: calculating and sorting the distances. The random subspace method seems to be computationally expensive, since the nearest neighbors calculation is done several times. However, the Euclidean distance between two D-dimensional feature vectors x_1 and x_2 can be organized in a way that the per-feature differences and their squares be computed only once, $\delta_j = (x_{1j} - x_{2j})^2$, $j = 1, \ldots, D$. Then for each random subspace i, $i = 1, \ldots, N$, with a randomly selected subspace FS_i, only the summation of per-feature squared differences of the r used features and the sorting of the sums that are performed,

$$d_i(x_1, x_2) = \sqrt{\sum_{j=1, j \in FS_i}^{D} \delta_j} \tag{9.5}$$

Given a labeled training set L, D-dimensional original feature space and N r-dimensional random subspaces, the run time complexity for N times nearest

neighbors calculation of an unlabeled example is $O(|L| \times D + N \times (|L| \times r + |L| \times \log\ |L|))$, and that for single calculation in the original feature space is $O(|L| \times D + |L| \times \log\ |L|)$. For classification, the probability assigned by classifier h_i that x_u belongs to class ω_k is calculated as follows,

$$h_i(x_u, \omega_k) = \frac{\sum_{x_n \in N_i(x_u), x_n \in \omega_k} W_n + 1/|L|}{\sum_{c=1}^{C} \sum_{x_n \in N_i(x_u), x_n \in \omega_c} W_n + C/|L|} \tag{9.6}$$

where W_n is defined as in Eq. (9.3) and Laplace correction is applied for *smoothing* the class probability distribution. The final probability assigned by the ensemble H that x_u belongs to class ω_k is,

$$H(x_u, \omega_k) = \frac{\sum_{i=1}^{N} h_i(x_u, \omega_k)}{\sum_{c=1}^{C} \sum_{i=1}^{N} h_i(x_u, \omega_c)} \tag{9.7}$$

Since *CoBC* is an incremental iterative method, it will avoid complete re-calculating and re-sorting of all the distances by utilizing geometrical constraints such as the triangular inequality. First, for each unlabeled example $x_u \in U$ and for each random subspace i, define the neighborhood of x_u given the initial L, $N_i(x_u)$. Afterward at each iteration t, only calculating and sorting the distances between x_u and the newly-labeled examples in L'_t. Then in the light of this, the triangular inequality is used to update the neighborhood $N_i(x_u)$ where the run time complexity is $O(|L'_t| \times D + N \times (|L'_t| \times r + |L'_t| \times \log\ |L'_t| + k))$.

9.3 Application to Visual Object Recognition

The experiments in this chapter are conducted on ten data sets representing five real-world image classification tasks. They are described in Chapter 7 (see Table 7.1). We intentionally select data sets with variance in number of features, number of classes and number of examples. It is worth mentioning that there are no missing values of any feature in all data sets.

9.3.1 UCI Handwritten Digits Recognition

The UCI handwritten digits data set was defined in Section 7.1.3 (see Figure 7.5). Each digit is described by four feature types: *mfeat-pix*, *mfeat-kar*, *mfeat-fac* and *mfeat-fou*.

9.3.2 Fruits Recognition

The fruits data set was defined in Section 7.1.1 (see Figure 7.2). Each image is described by two feature types: First, each image was divided into 3×3 overlapping sub-images. A color histogram was extracted from each sub-image (see Section

7.1.1.1) and then the nine histograms were concatenated to form the first input feature set for classification (*colorhist3x3*). In addition, each image was divided into 4×4 overlapping sub-images. Then, an orientation histogram based on Sobel edge detection was extracted from each sub-image (see Section 7.1.1.2). The 16 histograms were concatenated to form the second set of features (*sobel4x4*).

9.3.3 COIL-20 Objects Recognition

This Columbia Object Image Library benchmark dataset was defined in Section 7.1.4 (see Figure 7.6). First, a gray values histogram (see Section 7.1.1.1) was extracted from each image (*grayhist1x1*). In addition, each image was divided into 2×2 overlapping sub-images and an orientation histogram based on Sobel edge detection was extracted from each sub-image (see Section 7.1.1.2). Then, the four histograms were concatenated to form another set of features (*sobel2x2*).

9.4 Experimental Evaluation

9.4.1 Methodology

The effectiveness of the proposed framework is evaluated twice: using the *C4.5 pruned decision tree* (Section 2.3) with Laplace Correction (to improve the class probability estimates) and the *1-nearest neighbor* classifier (Section 2.2) as the base learning algorithms. The *RSM* (Section 9.2.3) is used to construct ensembles of size ten ($N = 10$) and each classifier uses only half of the available features that were randomly selected. For classification, *normalized sum* of the *CPE*s of the ensemble members is the final decision of an ensemble (except for *Co-Forest* where *majority vote* is applied). All algorithms are implemented using WEKA library [201]. All features are normalized to have zero mean and unit variance. For each experiment, the results are average of 4 runs of stratified ten-fold cross-validation procedure. That is, for each data set, 10% are used as test set, while the remaining 90% are used as training examples. For significance test, paired t-test, see Section 7.2.3, with 0.05 significance level is used (significance is marked with bullet($^\bullet$)).

In order to simulate the environment of *SSL*, 20% (10% for digits data sets) of the training examples are randomly selected as the initial labeled training set L while the remaining 80% (90% for digits data sets) are used as unlabeled data set U. The number of iterations T is chosen such that the *SSL* process stops when the number of labeled examples in L reaches 70% (60% for digits data sets) of the full training set size. The aim of the early stopping is to minimize the number of the potential mislabeled examples. The experiments show that the number of noisy examples increases as the number of iterations increases. The reason is that the classifiers select the easiest examples at the early iterations and keeping

the harder examples into the pool. The pool size u is set to 300 in the case of *1-nearest neighbor* and $u = 100$ for decision trees. The sample size n is one and the 10 nearest neighbors (k=10) are used to estimate the local competence.

9.4.2 Results

Tables 9.1, 9.2, 9.3 and 9.4 present the means and standard deviations of the test set error rate of the different learning algorithms. Figures 9.2 and 9.3 (for *1-nearest neighbor* classifier) and Figures 9.4 and 9.5 (for *C4.5 decision tree*) summarize and graphically compare the average test error rates of different algorithms.

9.4.2.1 RSM ensemble against single classifiers

For the *1-nearest neighbor* classifier, the RSM ensemble outperforms the single *1-NN* classifier for all datasets using 5%, 10% and 100% of training data set but the difference is not significant. For the *C4.5 decision tree*, the RSM ensemble outperforms the single decision trees for all datasets using 5%, 10% and 100% of training data set and the difference is statistically significant for most of the cases. These results are considered as a baseline and a prerequiste to perform the following experiments (see Tables 9.1(a), 9.2(a), 9.3(a) and 9.4(a)). The superior performance of the ensembles compared to the individual classifiers proves that the ensemble members are diverse. Thus, these ensembles satisfy the requirement needed to run *CoBC*.

Table 9.1: Mean and standard deviations of test error rates where *EnsembleLearn* = *RSM* and *BaseLearn* = *1-nearest neighbor* applied to handwritten digits

(a) Passive Supervised Learning (random sampling)

Data set	$\|L\|$	*mfeat-pix*	*mfeat-kar*	*mfeat-fac*	*mfeat-fou*	*ave.*
1-NN	5%	11.43(2.08)	18.60(3.01)	11.24(2.31)	35.92(2.99)	19.30
RSM(1-NN)	5%	10.98(2.10)	18.43(2.32)	11.20(2.48)	35.84(3.22)	19.11
1-NN	10%	7.01(1.43)	13.18(2.40)	8.03(1.80)	30.46(2.37)	14.67
RSM(1-NN)	10%	6.79(1.50)	12.20(2.38)	7.85(2.07)	29.61(2.62)	14.11
1-NN	100%	2.74(0.98)	4.49(1.43)	3.68(1.54)	20.84(2.63)	7.93
RSM(1-NN)	100%	2.55(0.89)	4.03(1.18)	3.49(1.44)	19.46(2.81)	7.38

(b) Passive *SSL* using CPE (Starting with 10% random sampling)

Data set		*mfeat-pix*	*mfeat-kar*	*mfeat-fac*	*mfeat-fou*	*ave.*
ST	*initial*	7.01(1.43)	13.18(2.40)	8.03(1.80)	30.46(2.37)	14.67
	final	5.66(1.65)	9.13(2.30) •	6.00(1.68) •	32.25(4.61)	13.26
	improv	19.26	30.73	25.28	-5.88	9.61
Co-Forest	*initial*	6.80(1.56)	12.86(2.51)	7.91(1.75)	30.05(3.22)	14.41
	final	4.05(1.53) •	8.23(2.19) •	6.15(1.73) •	26.56(3.24) •	11.25
	improv	40.44	36.00	22.25	11.61	21.93
CoBC	*initial*	6.79(1.50)	12.20(2.38)	7.85(2.07)	29.61(2.62)	14.11
	final	4.47(1.51) •	8.00(2.10) •	5.59(1.55) •	26.25(3.20) •	11.08
	improv	34.17	34.43	28.79	11.35	21.47

Table 9.2: Mean and standard deviations of test error rates where *EnsembleLearn* = *RSM* and *BaseLearn* = *1-nearest neighbor*

(a) Passive Supervised Learning (random sampling)

Data set	$\|L\|$	*ionosphere*	*fruits*		*COIL20*		*texture*	*ave.*
			colorhist3x3	*sobel4x4*	*grayhist1x1*	*sobel2x2*		
1-NN	10%	20.57(6.84)	14.38(4.60)	30.48(4.81	21.78(3.00)	17.92(3.87)	13.80(2.37)	19.82
RSM(1-NN)	10%	20.28(7.27)	13.75(4.75)	28.40(4.98)	21.72(2.76)	16.98(3.50)	13.80(2.69)	19.16
1-NN	20%	18.65(6.49)	8.31(2.93)	24.41(5.10)	16.58(2.97)	8.44(2.19)	11.62(2.83)	14.67
RSM(1-NN)	20%	16.45(6.23)	8.04(2.92)	20.75(4.27)	14.90(2.78)	8.27(2.72)	10.03(2.46)	13.07
1-NN	100%	13.10(5.13)	2.33(1.73)	7.18(2.67)	5.21(1.49)	0.26(0.44)	3.18(1.62)	5.21
RSM(1-NN)	100%	9.04(4.45) *	1.73(1.40)	6.05(2.21)	4.90(1.31)	0.21(0.39)	3.28(1.55)	4.20

(b) Passive *SSL* using CPE (Starting with 20% random sampling, until 70% *SSL*)

Data set		*ionosphere*	*fruits*		*COIL20*		*texture*	*ave.*
			colorhist3x3	*sobel4x4*	*grayhist1x1*	*sobel2x2*		
ST	*initial*	18.65(6.49)	8.31(2.93)	24.41(5.10)	16.58(2.97)	8.44(2.19)	11.62(2.83)	14.67
	final	17.10(8.63)	7.80(3.37)	21.02(4.64) •	15.09(2.44)	4.45(1.46) •	9.87(2.67)	12.55
	improv	9.38	6.14	13.89	8.99	47.27	15.06	14.68
Co-Forest	*initial*	17.94(6.89)	7.89(2.55)	21.88(4.02)	15.63(2.65)	9.15(2.76)	10.03(2.91)	13.75
	final	18.80(7.04)	6.44(2.71)	19.47(4.23) •	14.14(2.44) •	5.02(1.74) •	7.82(2.10)	11.95
	improv	-4.79	18.38	11.01	9.53	45.14	22.03	13.09
CoBC	*initial*	16.45(6.23)	8.04(2.92)	20.75(4.27)	14.90(2.78)	8.27(2.72)	10.03(2.46)	13.07
	final	13.31(6.38)	6.17(2.61)	16.91(4.28) •	12.07(2.46) •	3.46(1.38) •	8.03(2.43) •	9.99
	improv	19.09	23.26	18.51	18.99	58.16	19.94	23.57

9.4.2.2 CoBC against Self-Training

The performance of *CoBC* is compared with the single classifier semi-supervised learning algorithm, i.e., *Self-Training* (Section 5.3). For fair comparison, both algorithms are given as input the same L and U and allowed to label the same amount of unlabeled data. Tables 9.1(b), 9.2(b), 9.3(b) and 9.4(b) (for *CPE* based confidence measure) and Tables 9.3(c) and 9.4(c) (for *local competence* based confidence measure) present the average test error at iteration 0 (*initial*) trained only on the initially available labeled data L, after the final *SSL* iteration of exploiting the unlabeled data set U (*final*) and the relative improvement percentage ($improv = \frac{initial - final}{initial} \times 100$).

For the *1-nearest neighbor* classifier, significance test indicates that the final test error after *CoBC* is significantly better than its initial error on all the data sets except for *ionosphere* and *colorhist3x3* where the improvement is not significant. In addition, the final test error after *CoBC* is better than the final error after *Self-Training* on all the data sets.

For the *C4.5 decision tree* and *CPE*, the final test error after *Self-Training* is insignificantly better than the initial error before *Self-Training* for 5 data sets while there is no improvement for the other five datasets. For *CoBC*, there is statistically insignificant improvement after *SSL* for only 3 datasets while the performance gets worse for the other seven datasets.

For the *C4.5 decision tree* and *local competence*, significance test indicates that the final test error after *CoBC* is significantly better than its initial error

on 3 data sets (*mfear-pix*, *mfear-kar* and *mfear-fou*) while the improvement is not significant for 4 data sets (*mfear-fac*, *sobel4x4*, *grayhist1x1* and *sobel2x2*). For the other 3 data sets (*colorhist3x3*, *ionosphere* and *texture*), *CoBC* does not work. In addition, the significance test indicates that the final error after *CoBC* is significantly better than the final error after *Self-Training* for all data sets except three where the improvement is not significant.

Table 9.3: Mean and standard deviations of test error rates where *EnsembleLearn* = *RSM* and *BaseLearn* = *C4.5 pruned decision tree* applied to handwritten digits datasets

(a) Passive Supervised Learning (random sampling)

Data set	$\|L\|$	*mfeat-pix*	*mfeat-kar*	*mfeat-fac*	*mfeat-fou*	*ave.*
J48	5%	30.99(4.60)	42.04(3.62)	32.63(4.80)	41.74(4.74)	36.85
RSM	5%	22.78(4.17)$^\bullet$	31.53(4.41)$^\bullet$	23.23(4.82)$^\bullet$	35.10(3.95)$^\bullet$	28.16
J48	10%	25.48(3.45)	34.51(4.01)	25.03(2.79)	36.80(4.82)	30.45
RSM	10%	16.69(2.82)$^\bullet$	21.30(2.96)$^\bullet$	15.71(2.52)$^\bullet$	29.29(3.50)$^\bullet$	20.75
J48	100%	11.23(1.96)	17.54(2.10)	11.66(2.53)	23.71(2.63)	16.03
RSM	100%	5.10(1.61)$^\bullet$	7.90(1.52)$^\bullet$	5.33(1.71)$^\bullet$	17.69(2.42)$^\bullet$	9.00

(b) Passive *SSL* using *CPE* (Starting with 10% random sampling)

Data set		*mfeat-pix*	*mfeat-kar*	*mfeat-fac*	*mfeat-fou*	*ave.*
ST	*initial*	25.48(3.45)	34.51(4.01)	25.03(2.79)	36.80(4.82)	30.45
	final	25.01(3.50)	34.40(3.76)	25.14(3.52)	37.54(4.73)	30.52
	improv	1.84	0.32	-0.44	-2.01	-0.23
Co-Forest	*initial*	14.51(3.05)	21.83(3.39)	14.11(2.59)$^\bullet$	29.35(3.18)	19.95
	final	16.01(3.59)	21.83(3.39)	17.18(3.01)	29.24(3.42)	21.06
	improv	-10.33	0.0	-21.75	0.37	-5.56
CoBC	*initial*	16.69(2.82)	21.30(2.96)	15.71(2.52)	29.29(3.50)	20.75
	final	19.24(3.02)	21.13(3.63)	19.00(3.91)	29.31(3.80)	22.17
	improv	-15.28	0.80	-20.94	-0.07	-6.84

(c) Passive *SSL* using *Competence* (Starting with 10% random sampling)

Data set		*mfeat-pix*	*mfeat-kar*	*mfeat-fac*	*mfeat-fou*	*ave.*
ST	*initial*	25.48(3.45)	34.51(4.01)	25.03(2.79)	36.80(4.82)	30.45
	final	22.50(2.81)$^\bullet$	29.20(4.16)$^\bullet$	21.50(3.30)$^\bullet$	34.15(3.76)$^\bullet$	26.84
	improv	11.70	15.39	14.10	7.20	11.86
CoBC	*initial*	16.69(2.82)	21.30(2.96)	15.71(2.52)	29.29(3.50)	20.75
	final	13.18(3.58)$^\bullet$	15.10(2.20)$^\bullet$	14.24(3.33)	25.65(3.03)$^\bullet$	17.04
	improv	21.03	29.11	9.36	12.43	17.88

9.4.2.3 CPE against Local Competence

The performance of the proposed *local competence* confidence measure is compared with *CPE* one. For the *1-nearest neighbor* classifier, the results of using local competence as confidence measure are not reported. There is no real benefit from estimating the local competence over CPE because both of them depend on the neighborhood of an unlabeled example to estimate the confidence in the prediction of its label.

For the *C4.5 decision tree* and for a fair comparison, the same parameter setting is employed. Table 9.3(b) and Table 9.4(b) show the test errors using

CPE provided by RSM ensemble as confidence estimate. By averaging on all the data sets, the average test error after *random-then-ST* and *random-then-CoBC* using *CPE* increases by 1.5% and 6.42% (for digits data sets, 0.23% and 6.84%) respectively, while it decreases by 4.37% and 1.25% (for digits data sets, 11.86% and 17.88%) using local competence estimates. This confirms the claim that inaccurate confidence measure leads to degradation of the performace.

9.4.2.4 CoBC against Co-Forest

The performance of *CoBC* is compared with the single classifier semi-supervised learning algorithm, i.e., *Co-Forest* (Section 5.7.5.4). For a fair comparison, *Co-Forest* is applied using the random subspace method (Section 9.2.3) as ensemble learner and either *C4.5 decision tree* or *1-nearest neighbor* classifier as in *CoBC*. However, the initial test error before *Co-Forest* is different from the initial error before *CoBC* because the *Co-Forest*'s initial classifiers are not only trained using different random feature subsets but also trained using different bootstrap samples from L and *majority voting* is employed to produce the final decision.

For the *1-nearest neighbor* classifier and from Table 9.1(b) and Table 9.2(b), the final test error after *Co-Forest* is significantly better than its initial error on 7 out of ten data sets. The improvement is not significant for 2 data sets *colorhist3x3* and *texture*. Although *CoBC* achieves a relative improvement 19.09% for *ionosphere*, *Co-Forest* does not work for this data set. By averaging on all the data sets, the average test error after *CoBC* decreases by 23.57% (for digits four data sets, 21.47%) and after *Co-Forest* it decreases by 13.09% (for digits data sets, 21.93%). Note that *Co-Forest* stops if the training error reaches zero and it ignores the *CPE*s provided by single classifiers.

For the *C4.5 decision tree* and from Table 9.3(b) and Table 9.4(b), *Co-Forest* failed to improve the classification accuracy using unlabeled data. One can attribute the poor performance of *Co-Forest* to the irrelevant setting of θ for multi-class problems (similar results for $\theta = 0.75$ and $\theta = 0.6$). For *mfeat-fac* dataset, the initial ensemble before *Co-Forest* significantly outperforms the final ensemble after *Co-Forest*. The shortcomings of *Co-Forest* from my point of views are given in Section 5.7.5.4.

9.5 Related Work

9.5.1 Improving Decision Trees Class Probability Estimation

It has been observed that decision trees provide poor probability estimates. Thus, designing decision trees with accurate probability estimation, called Probability Estimation Trees (PETs), has attracted attention. Provost and Domingos [150]

Table 9.4: Mean and standard deviations of test error rates where *EnsembleLearn* = *RSM* and *BaseLearn* = *C4.5 pruned decision tree*

(a) Passive Supervised Learning (random sampling)

Data set	$\|L\|$	*ionosphere*	*fruits*		*COIL20*		*texture*	*ave.*
			colorhist3x3	*sobel4x4*	*grayhist1x1*	*sobel2x2*		
RSM(J48)	10%	16.96(7.27)	23.31(5.05)	28.19(6.21)	28.27(3.92)	26.06(4.62)	25.19(4.36)	24.66
J48	20%	13.98(6.86)	19.71(4.51)	31.02(5.96)	21.97(3.64)	30.02(3.69)	23.23(4.01)	23.32
RSM(J48)	20%	11.69(5.28)	14.47(4.37)$^\bullet$	19.17(5.03)$^\bullet$	20.44(3.45)$^\bullet$	16.69(3.20)$^\bullet$	18.55(3.76)$^\bullet$	16.83
J48	100%	10.83(4.75)	10.42(2.69)	15.63(4.00)	10.23(2.48	9.86(2.58)	12.87(2.81)	11.64
RSM(J48)	100%	7.19(4.28)$^\bullet$	4.35(2.34)$^\bullet$	7.12(2.81)$^\bullet$	9.59(2.34)	2.90(1.54)$^\bullet$	8.55(2.42)$^\bullet$	6.62

(b) Passive *SSL* using CPE (Starting with 20% random sampling, until 70% *SSL*)

Data set		*ionosphere*	*fruits*		*COIL20*		*texture*	*ave.*
			colorhist3x3	*sobel4x4*	*grayhist1x1*	*sobel2x2*		
ST	*initial*	13.98(6.86)	19.71(4.51)	31.02(5.96)	21.97(3.64)	30.02(3.69)	23.23(4.01)	23.32
	final	15.88(6.69)	19.23(4.25)	30.96(5.58)	22.30(3.42)	30.46(4.21)	23.21(4.27)	23.67
	improv	-13.59	2.44	0.19	-1.50	-1.47	0.09	-1.50
Co-Forest	*initial*	10.48(5.80)	13.46(4.21)	17.65(4.01)	17.38(2.93)	15.16(3.43)	15.62(3.55)	14.96
	final	10.54(5.15)	13.64(4.34)	19.17(4.04)	19.24(2.92)	15.04(2.82)	17.89(4.86)	15.92
	improv	-0.57	-1.34	-8.61	-10.70	0.79	-14.53	-6.42
CoBC	*initial*	11.69(5.28)	14.47(4.37)	19.17(5.03)	20.44(3.45)	16.69(3.20)	18.55(3.76)	16.83
	final	11.33(5.31)	17.15(4.16)	21.73(4.84)	20.09(3.60)	18.72(3.30)	20.46(4.74)	18.25
	improv	3.08	-18.52	-13.35	1.71	-12.16	-10.30	-8.44

(c) Passive *SSL* using Competence (Starting with 20% random sampling, until 70% *SSL*)

Data set		*ionosphere*	*fruits*		*COIL20*		*texture*	*ave.*
			colorhist3x3	*sobel4x4*	*grayhist1x1*	*sobel2x2*		
ST	*initial*	13.98(6.86)	19.71(4.51)	31.02(5.96)	21.97(3.64)	30.02(3.69)	23.23(4.01)	23.32
	final	16.46(7.00)	18.63(4.69)	31.20(5.74)	20.87(3.84)	25.04(3.92) $^\bullet$	21.60(4.75)	22.30
	improv	-17.74	5.48	-0.58	5.01	16.59	7.02	4.37
CoBC	*initial*	11.69(5.28)	14.47(4.37)	19.17(5.03)	20.44(3.45)	16.69(3.20)	18.55(3.76)	16.83
	final	14.03(6.96)	16.02(3.61)	17.15(4.28)	18.84(2.98)	15.11(3.21)	18.60(4.21)	16.62
	improv	-20.02	-10.71	10.54	7.83	9.47	-0.27	1.25

have proposed to improve C4.5 for better ranking by applying two techniques on it: turning off pruning and collapsing to keep some branches that contribute much to the quality of ranking, and using Laplace correction at leaves to smooth the generated probabilities towards the prior distribution. The new model is called C4.4. Although experiments have shown that C4.4 greatly scales up the ranking quality of decision trees, there still exist two contradictions. (i) Without pruning, C4.4 is relatively much larger than traditional decision trees, which may over-fit sample sets in the training time and the resulting probabilities are definitely inaccurate. (ii) The number of leaves will increase so that the number of examples at each leaf will become relatively small. The probabilities produced from such a small sample set are unreliable. Also, probability values could easily repeat, thus many unlabeled examples will share the same probability and will be ranked randomly. This results in a negative effect on the performance of ranking-based applications such as semi-supervised learning. Kohavi [97] proposed a Naive Bayes Tree *NBTree* that is a hybrid of decision tree and naive Bayes, where a naive Bayes classifier is deployed at each leaf to produce classification and probability

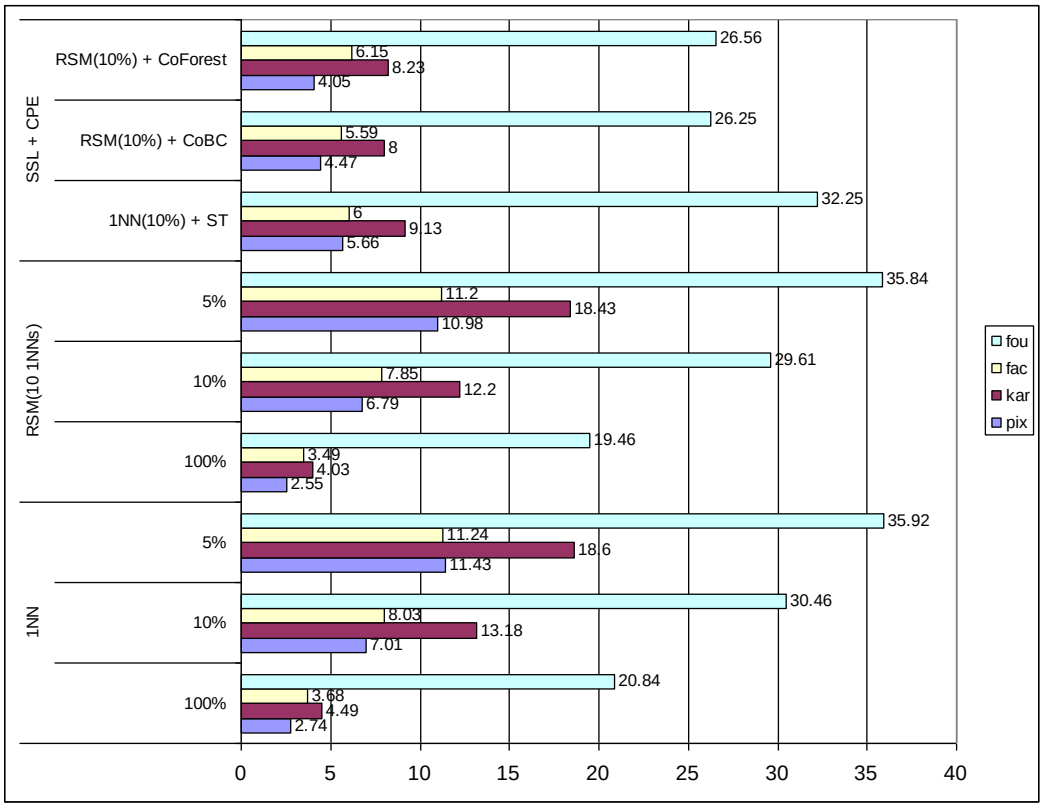


Figure 9.2: Average of test error rates using *1-nearest neighbor* classifier for digits data sets

estimation. The experiments showed that *NBTree* outperforms both naive Bayes and decision tree for large data sets. In semi-supervised learning settings where the number of examples at each leaf is small, the main drawback is that naive Bayes classifier suffers from high bias and high variance. Therefore, this leads to unreliable *CPE*.

Liang et al. [120] proposed to resolve this problem from two approaches. The first approach, Lazy Distance-based Tree *LDTree*, trains a k-nearest neighbor classifier at each leaf to explicitly distinguish the different contributions of leaf examples when estimating the probabilities for an unlabeled example. The second approach, Eager Distance-based Tree *EDTree*, improves *LDTree* by changing it into an eager algorithm. In both approaches, each unlabeled example is assigned a set of unique probabilities of class membership instead of a set of uniformed ones, which gives finer resolution to distinguish examples and leads to the improvement of ranking but avoids the high variance by weighting each leaf example based on the similarity between it and the unlabeled example. Experiments on 34 UCI data sets [27] have shown that *LDTree* and *EDTree* outperform C4.5, C4.4 and other standard smoothing methods designed for improve ranking.

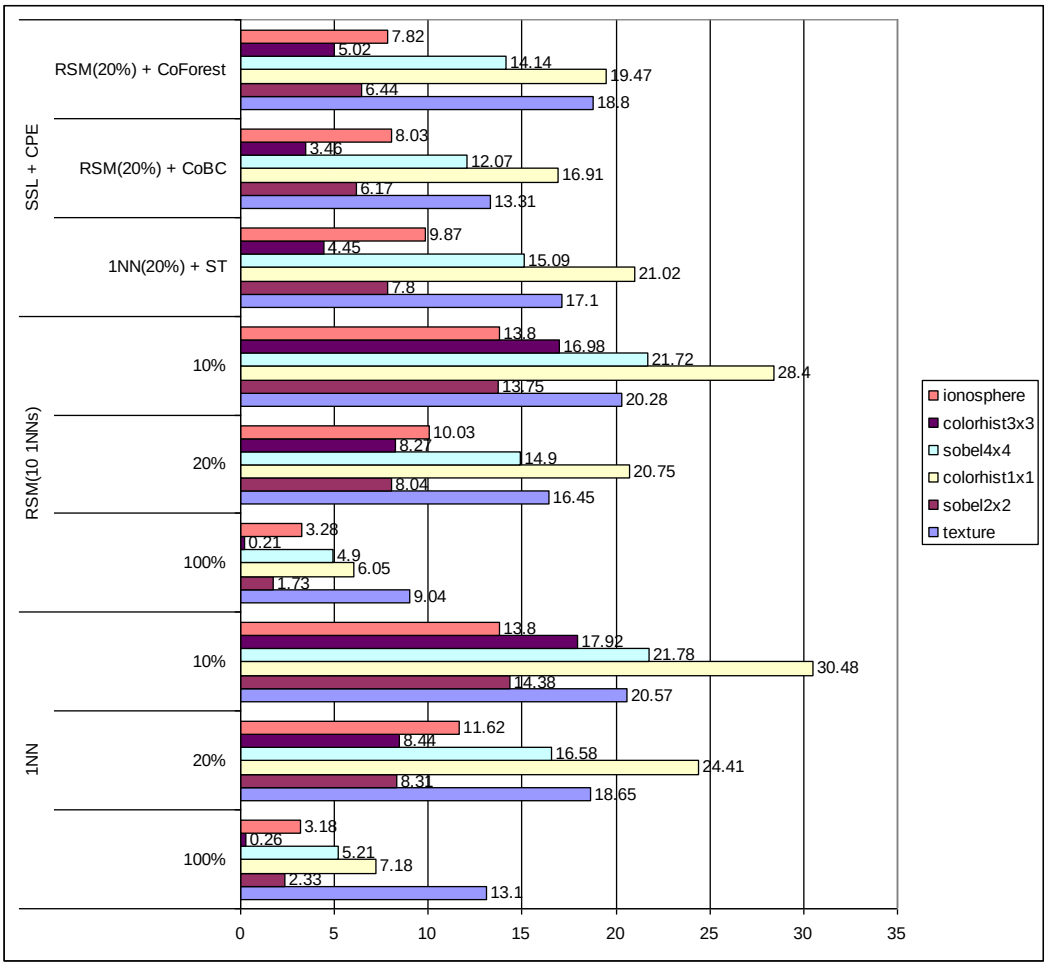


Figure 9.3: Average of test error rates using *1-nearest neighbor* classifier

9.5.2 Single-View Co-Training

A number of recent studies [71, 210, 215, 119] has investigated the applicability of the main idea of *Co-Training* using a single view where multiple redundant and independent views are not required. A brief overview of these approaches are given in Section 5.7.5.

9.6 Conclusions and Future Work

In real-world data mining application, selecting and labeling the training examples is tedious, expensive and time consuming. To minimize the cost, supervised learning algorithm can select the most informative examples for training and exploit the unlabeled data to boost its performance.

In this study, I introduced a new *committee-based single-view Co-Training*

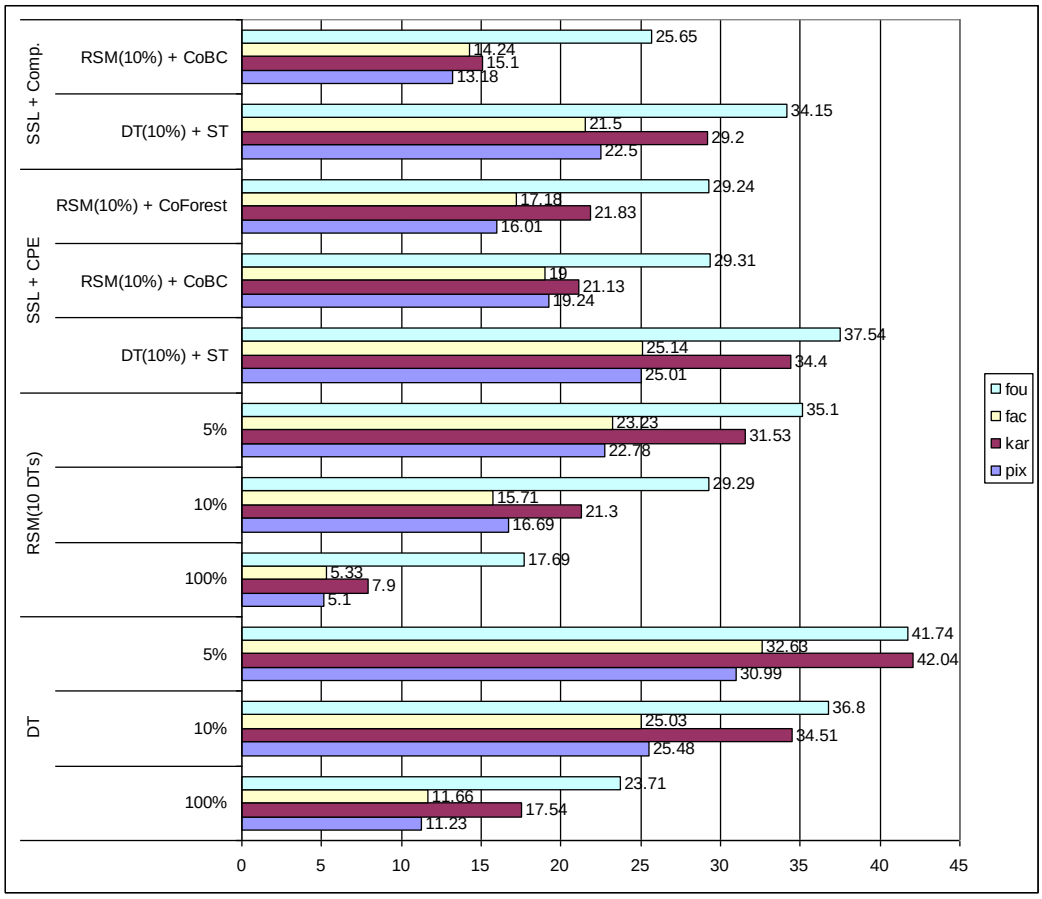


Figure 9.4: Average of test error rates using *C4.5 decision tree* for digits data sets

style algorithm for *semi-supervised learning*, *CoBC*, for application domains in which the available data is not described by multiple redundant and independent views. Experiments were conducted on ten image recognition tasks in which the random subspace method is used to construct diverse ensembles of *1-nearest neighbor* classifiers and *C4.5 decision trees*. The results verify the effectiveness of *CoBC* to exploit the unlabeled data given a small amount of labeled examples. We have the following conclusions:

1. *CoBC* can relax the strong requirements of standard *Co-Training* algorithm through using a committee of diverse classifiers instead of using redundant and independent views.

2. For the *C4.5 decision tree*, the local competence based confidence measure compensates its inaccurate class probability estimates. This improvement can be attributed to the dependence on the neighborhood of an unlabeled example to measure confidence about its predicted class label. On the

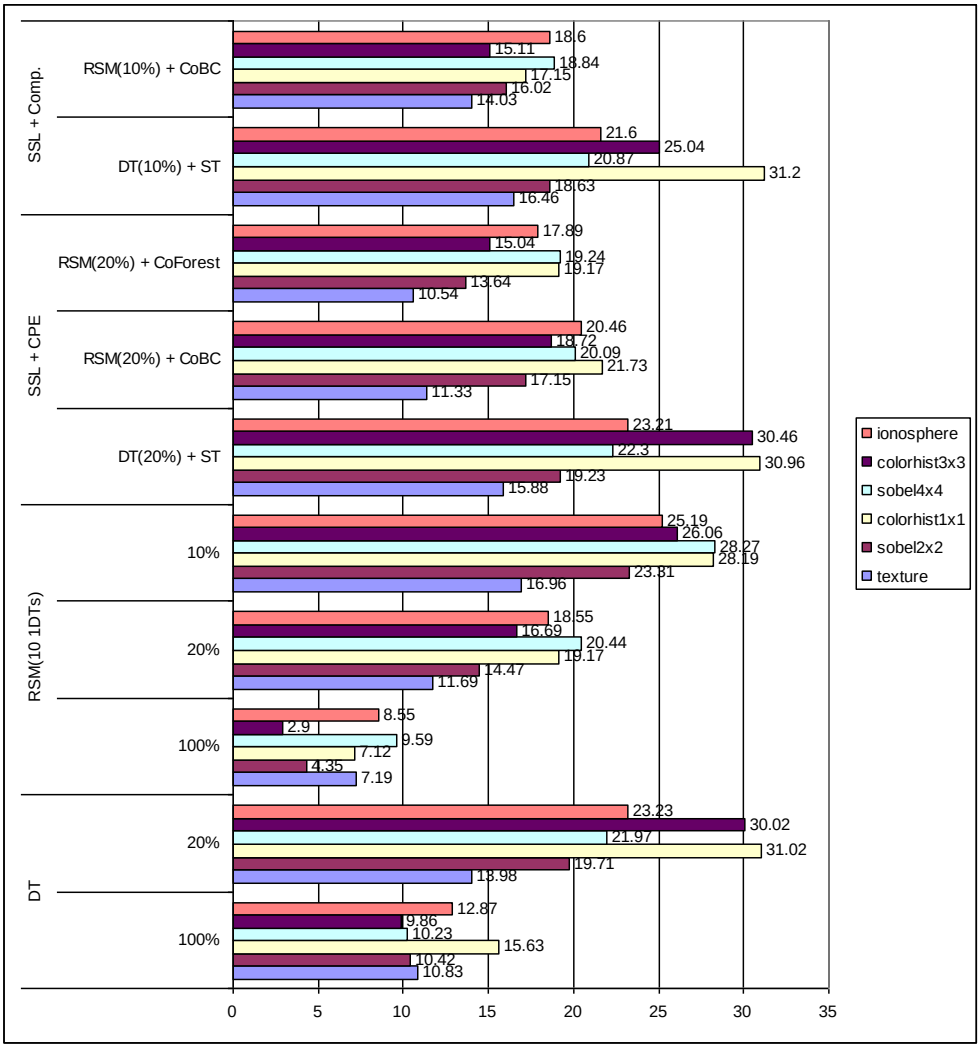


Figure 9.5: Average of test error rates using *C4.5 decision tree*

other hand, class probability estimates provided by decision tree takes into account neither the distance (or similarity) between the unlabeled example and the labeled training examples nor the distance between it and the decision boundaries.

3. *CoBC* can improve the recognition rate if the most confident examples with respect to the companion committee H_i are informative examples with respect to h_i (lie close to its decision boundary).

4. Although *CoBC* selects the most confident examples, adding mislabeled examples to the training set is unavoidable but the negative impact of this *noise* could be alleviated by adding a sufficient amount of newly labeled examples.

5. There is no *SSL* algorithm that is the best for all real-world data sets. Each *SSL* algorithm has its strong assumptions because labeled data is scarce and there is no guarantee that unlabeled data will always help. One should use the method whose assumptions match the given problem. Inspired by [219], we have the following checklist: If the classes produce well clustered data, then EM with generative mixture models may be a good choice; If the features are naturally divided into two or more redundant and independent sets of features, then standard *Co-Training* may be appropriate; If *SVM* is already used, then *Transductive SVM* is a natural extension; In all cases, *CoBC* is a practical wrapper method.

There are many interesting directions for future work.

1. As *CoBC* is general framework, we plan to evaluate it using other ensemble learners such as Bagging and AdaBoost and other base learners such as MLP, Naive Bayes and RBF Networks.

2. We are planing to study the influence of changing the values of some parameters on the performance of *CoBC* such as the number and the dimensionality of random subspaces used by *RSM*, number of nearest neighbors used for both local competence estimation and *k-nearest neighbors classifier*.

3. To investigate different ways to improve the class probabilities estimated by *C4.5 decision trees* and study the influence of this improvement on the performance of *CoBC* framework.

Chapter 10

Combining Committee-based SSL and Active Learning

10.1 Introduction

Both *semi-supervised learning* (Chapter 5) and *active learning* (Chapter 6) tackle the same problem but from different directions. That is, they aim to improve the generalization error and at the same time minimize the cost of data annotation through exploiting the abundant unlabeled data. *Semi-supervised learning* exploits the unlabeled examples where the underlying classifiers are *most confident* in the prediction of their class labels. They depend on a given confidence measure for sample selection. On the other hand, *active learning* exploits the *most informative* unlabeled examples where the underlying classifiers disagree on the prediction of their labels (contention points). The work in this chapter has been previously published ([5, 4]). They depend on what is called in the literature

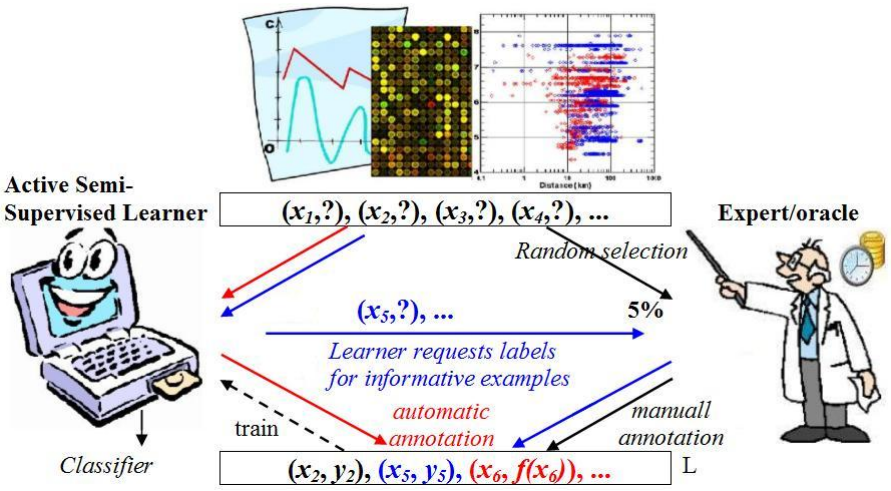


Figure 10.1: Graphical illustration of combining SSL and active learning

utility or informativeness measure for sample selection. Figure 10.2 illustrates the idea of combining both paradigms, compare between it and Figure 6.1 and Figure 5.4. *Ensemble learning* (Chapter 3) aims to improve the generalization error through constructing a set of different classifiers instead of a single classifier. They depend on reducing or alleviating the *statistical*, *computational* or *representational* problems that face any base learning algorithm (chapter 2). If one can design a multiple classifier system (ensemble) that learns from both the *most confident* examples and *most informative* examples, this will lead to better prediction results. One can see that the three paradigms complete each other. Although there are some approaches that combine *semi-supervised* and *active learning* to integrate their benefits, such as [125, 133, 212], there is no work done to investigate the combination of *committee-based semi-supervised learning* with *committee-based active learning*.

In the previous chapter, a single-view variant of *Co-Training*, called *Co-Training by Committee* (*CoBC*) is proposed, in which an ensemble of diverse classifiers is used for *semi-supervised learning* instead of multiple redundant and independent views. In this chapter, I aim to investigate the combination of the proposed framework *CoBC* with the state-of-the-art *active learning* algorithms. I introduce two new learning frameworks, denoted as *QBC-then-CoBC* and *QBC-with-CoBC*, which combine the merits of committee-based *semi-supervised learning* and *active learning*. The random subspace method is applied on both C4.5 decision trees and 1-nearest neighbor classifiers to construct the diverse ensembles used for *semi-supervised learning* and *active learning*. Experiments were conducted on the ten image recognition tasks used in the previous chapter. The results have shown that *QBC-then-CoBC* and *QBC-with-CoBC* can enhance the performance of *CoBC* and outperform other non committee-based combinations of semi-supervised and active learning algorithms. The work in this chapter has been previously published ([5, 4]).

10.2 Architecture I: *QBC* then *CoBC*

The most straightforward method of combining *QBC* and *CoBC* is to run *CoBC* after *QBC*, which is called in this thesis as *QBC-then-CoBC*. The objective is that active learning can help *CoBC* through providing it with a better starting point instead of randomly selecting examples to label for the starting point. The *CoBC* framework is outlined in Algorithm 14 in Chapter 9 and *QBC* framework is described in Algorithm 8 in Chapter 6. *QBC* selects the training examples that *CoBC* cannot reliably label on its own. Hence, we expect that *QBC-then-CoBC* will outperform both stand-alone *QBC* and stand-alone *CoBC*. In addition, one can expect that *QBC-then-CoBC* will outperform other possible combinations of non committee-based *active learning* and *semi-supervised learning* algorithms. The motivation of this hypothesis is the fact that an ensemble of diverse and

accurate classifiers outperforms its individual members [78]. In this study, *QBC* depends on the class probability estimate provided by the ensemble H in order to measure the utility (informativeness) of an unlabeled example x_u. Thus, the *most informative* example is the least confident one, where

$$Confidence(x_u, H^{(t-1)}) = \max_{1 \leq c \leq C} H^{(t-1)}(x_u, \omega_c). \tag{10.1}$$

Other utility measures can be utilized such as vote entropy, margin or *Jensen-Shannon* (JS) divergence that are described in Chapter 6. On the other hand, *CoBC* depends on the local competence estimate introduced in the previous chapter to select the *most confident* examples.

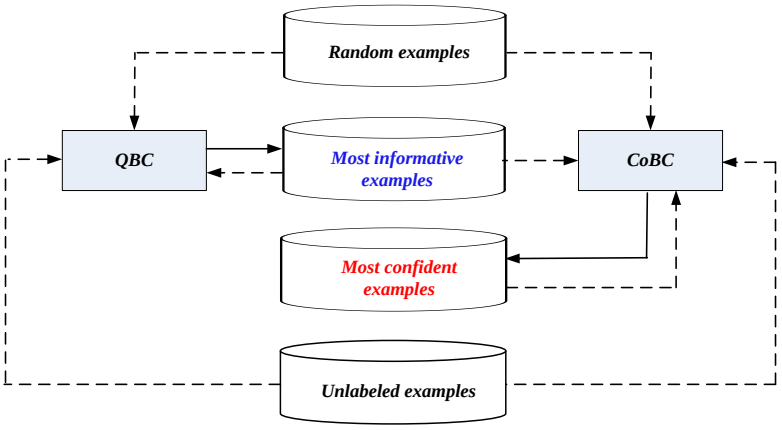


Figure 10.2: Graphical illustration of *QBC-then-CoBC*

10.3 Architecture II: *QBC* with *CoBC*

A more interesting approach is to interleave *CoBC* with *QBC*, so that *CoBC* not only runs on the results of active learning, but *CoBC* also helps *QBC* in the sample selection process as it augments the labeled training set with newly automatically labeled examples. Thus, mutual benefit can be achieved, which is called in this thesis as *QBC-with-CoBC*. It is given in Algorithm 16 and illustrated graphically in Figure 10.3. Let $L = \{(x_\mu, y_\mu) | x_\mu \in \mathbb{R}^D, y_\mu \in \Omega, \mu = 1, \ldots, m\}$ be the set of labeled training examples where each example is described by a D-dimensional feature vector $x_\mu \in \mathbb{R}^D$, y_μ denotes the class label of x_μ and $\Omega = \{\omega_1, \ldots, \omega_K\}$ is the set of target classes (ground truth). Also let $U = \{x_u | u = 1, \ldots, n\}$ be the set of unlabeled data. At each *QBC* round, we run *CoBC* for a predefined number of iterations (T_{CoBC}) (Algorithm 14). The objective is to improve the performance of the committee members through updating them with the most competent examples selected by *CoBC*. With more accurate committee members,

QBC should select more informative examples to label. Hence, we expect that *QBC-with-CoBC* will outperform both stand-alone *QBC* and *CoBC*. In addition, we expect that *QBC-with-CoBC* will outperform *QBC-then-CoBC* since both *QBC* and *CoBC* are benefiting from each other.

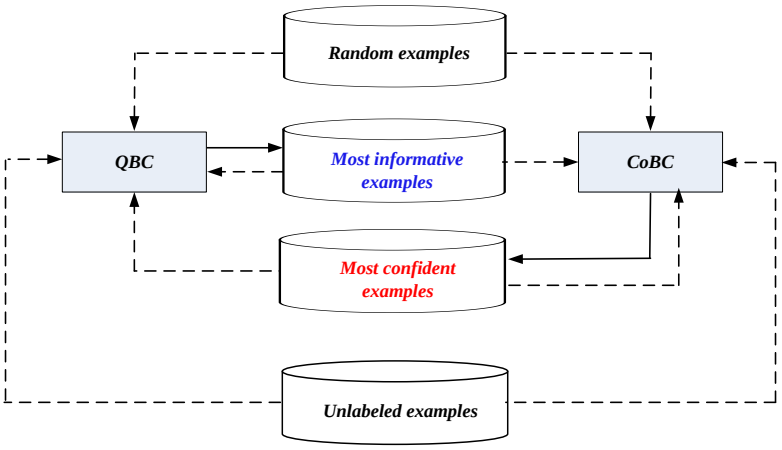


Figure 10.3: Graphical illustration of *QBC-with-CoBC*

10.4 Related Work

10.4.1 SSL with graphs

Zhu et al. [221] combine *semi-supervised learning* and *active learning* under a Gaussian random field model. Labeled and unlabeled data are represented as nodes in a weighted graph, with edge weights encoding the similarity between examples. Then the semi-supervised learning problem is formulated, in another work by the same authors [220], in terms of a Gaussian random field on this graph, the mean of which is characterized in terms of harmonic functions. *Active learning* was performed on top of the *semi-supervised learning* scheme by greedily selecting queries from the unlabeled data to minimize the estimated expected classification error (risk); in the case of Gaussian fields the risk is efficiently computed using matrix methods. They present experimental results on synthetic data, handwritten digit recognition, and text classification tasks. The active learning scheme requires a much smaller number of queries to achieve high accuracy compared with random query selection. Hoi et al. [83] proposed a novel framework that combine support vector machines and semi-supervised active learning for image retrieval. It is based on the Gaussian fields and harmonic functions semi-supervised approach proposed by Zhu et al. [220].

Algorithm 16 The pseudo code of *QBC-with-CoBC*

Require: set of labeled training examples (L), set of unlabeled training examples (U), maximum number of iterations (T_{QBC} and T_{CoBC}), ensemble learning algorithm ($EnsembleLearn$), base learning algorithm ($BaseLearn$), committee size (N), number of unlabeled examples in the pool (u), number of nearst neighbors (k), sample size (n) and number of classes (C)

Training Phase

Get the class prior probabilities, $\{Pr_c\}_{c=1}^{C}$
Set the class growth rate, $n_c = n \times Pr_c$ where $c = 1, \ldots, C$
Construct a committee of N classifiers,
$H^{(0)} = EnsembleLearn(L, BaseLearn, N)$
for $t \in \{1, \ldots, T_{QBC}\}$ **do**
  $L'_t = \emptyset$
  if U is empty **then** $T = t$-1 and abort loop **end if**
  {Get the most informative examples for each class}
  $U'_t \leftarrow RandomSubsample(U, u)$
  $\forall x_u \in U'_t$, calculate $Confidence(x_u, H^{(t-1)})$
  Rank the examples in U'_t based on confidence (in ascending order)
  Select the n_c least confident examples assigned for each class ω_c (insert them into subset L'_t) and ask an oracle to label L'_t
  $U'_t = U'_t \setminus L'_t$ and $U = U \cup U'_t$
  if L'_t is empty **then** $T \leftarrow t$-1 and abort loop **end if**
  {Re-train the N classifiers using their augmented training sets}
  for $i \in \{1, \ldots, N\}$ **do**
    $L_i = L_i \cup L'_t$ and $h_i^{(t)} = BaseLearn(L_i)$
  end for
  if $T_{CoBC} > 0$ **then**
    $H^{(t)} = CoBC\,(L, U, T_{CoBC}, EnsembleLearn, BaseLearn, N, u, k, n, C, H^{(t)})$
  end if
end for

Prediction Phase

return $H^{(T)}(x) = \frac{1}{N} \sum_{i=1}^{N} h_i^{(T)}(x)$ for a given sample x

10.4.2 SSL with generative models

McCallum and Nigam [125] presents a Bayesian probabilistic framework for text classification that reduces the need for labeled training documents by taking advantage of a large pool of unlabeled documents. First they modified the *Query-by-Committee* method of active learning (*QBC*) (Section 6.4.2) to use the unlabeled pool for explicitly estimating document density when selecting examples for labeling. Then modified *QBC* is combined with *Expectation-Maximization* (*EM*) (Section 5.4) in order to predict the class labels of those documents that re-

main unlabeled. They proposed two approaches to combine *QBC* and *EM*, called *QBC-then-EM* and *QBC-with-EM*. *QBC-then-EM* runs *EM* to convergence after actively selecting all the training examples that will be labeled. This means to use *QBC* to select a better starting point for *EM* hill climbing, instead of randomly selecting documents to label for the starting point. *QBC-with-EM* is a more interesting approach to interleave *EM* with *QBC* so that *EM* not only builds on the results of *QBC*, but *EM* also informs *QBC*. To do this, *EM* runs to convergence on each committee member before performing the disagreement calculations. The aim is (1) to avoid requesting labels for examples whose label can be reliably predicted by EM, and (2) to encourage the selection of examples that will help *EM* find a local maximum likelihood with higher classification accuracy. This directs *QBC* to pick more informative documents to label because it has more accurate committee members. Experimental results show that that using the combination of *QBC* and *EM* performs better than using either individually and requires only slightly half the number of labeled training examples required by either *QBC* or EM alone to achieve the same accuracy.

10.4.3 SSL with Committees

Muslea et al. [133] combined *Co-Testing* (Section 6.4.3) and *Co-EM* (Section 5.7.1.2) in order to produce an active multi-view semi-supervised algorithm, called *Co-EMT*. The experimental results on web page classification show that *Co-EMT* outperforms other non-active multi-view algorithms (*Co-Training* and *Co-EM*) without using more labeled data and that it is more robust to the violation of the requirements of two independent and redundant views. Zhou et al. [212] proposed an approach, called *SSAIR* (Semi-Supervised Active Image Retrieval), that attempts to exploit unlabeled data to improve the performance of content-based image retrieval (*CBIR*). In detail, in each iteration of relevance feedback, two simple classifiers are trained from the labeled data, i.e. images result from user query and user feedback. Each classifier then predicts the class labels of the unlabeled images in the database and passes the most relevant/irrelevant images to the other classifier. After re-training with the additional labeled data, the classifiers classify the images in the database again and then their classifications are combined. Images judged to be relevant with high confidence are returned as the retrieval result, while these judged with low confidence are put into the pool which is used in the next iteration of relevance feedback. Experiments show that semi-supervised learning and active learning mechanisms are both beneficial to *CBIR*. It is worth mentioning that *SSAIR* depends on single-view versions of *Co-Testing* (Section 6.4.3) and *Co-Training* that require neither two independent and redundant views nor two different supervised learning algorithm. In order to create the diversity, the two classifiers used for *Co-Testing* and *Co-Training* are trained using the Minkowsky distance metric with different distance order.

10.5 Experimental Evaluation

10.5.1 Methodology

The experiments in this chapter were conducted on the same ten real-world image classification tasks defined in Section 9.3. The methodology is the same as defined in Section 9.4.1. For *QBC-with-CoBC*, the number of *CoBC* iterations performed at each *QBC* is set to one (T_{CoBC}=1). For both *QBC* and *Uncertainty Sampling* algorithms, only 10% (5% for digits data sets) of the training examples are randomly selected as L and it selects the most informative examples from the remaining examples where the algorithms stop when an additional 10% (5% for digits data sets) have been selected and added to L. For significance test, paired t-test, see Section 7.2.3, with 0.05 significance level is used (significance is marked with bullet($^\bullet$)). Table 10.1 illustrates the results of the significance tests. Each entry $w/t/l$ in Table 10.1 indicates that the model in the corresponding row wins w data sets, ties in t data sets, and loses l data sets, in contrast with the model in the corresponding column based on significance test.

10.5.2 Results

Tables 11.2, 11.3, 10.4 and 10.5 present the means and standard deviations of the test set error rates of the different learning algorithms. Figures 10.8 and 10.9 show the learning curves of the RSM ensembles at the different learning iterations. Figures 10.4 and 10.5 (for *1-nearest neighbor* classifier) and Figures 10.6 and 10.7 (for *C4.5 decision tree*) summarize and graphically compare the average test error rates of different algorithms.

10.5.2.1 RSM ensemble against single classifiers

As shown in section 9.4.2.1, the superior performance of the ensembles compared to the individual classifiers proves that the ensemble members are diverse and accurate. Thus, these ensembles satisfy the requirement needed to run *CoBC*.

Table 10.1: Pairwise Comparison

1-nearest neighbor				
Models	*QBC*	*CoBC*	*QBC-then-CoBC*	*QBC-with-CoBC*
QBC-then-CoBC	8/2/0	5/5/0	-	3/7/0
QBC-with-CoBC	5/5/0	1/9/0	0/7/3	-
C4.5 pruned decision tree				
Models	*QBC*	*CoBC*	*QBC-then-CoBC*	*QBC-with-CoBC*
QBC-then-CoBC	3/7/0	3/7/0	-	1/8/1
QBC-with-CoBC	3/5/2	2/8/0	1/8/1	-

Table 10.2: Mean and standard deviations of test error rates where *EnsembleLearn* = *RSM* and *BaseLearn* = *1-nearest neighbor* applied to handwritten digits

(a) Passive SSL using CPE (Starting with 10% random sampling)

Data set		*mfeat-pix*	*mfeat-kar*	*mfeat-fac*	*mfeat-fou*	*ave.*
ST	*initial*	7.01(1.43)	13.18(2.40)	8.03(1.80)	30.46(2.37)	14.67
	final	5.66(1.65)	9.13(2.30) •	6.00(1.68) •	32.25(4.61)	13.26
	improv	19.26	30.73	25.28	-5.88	9.61
CoBC	*initial*	6.79(1.50)	12.20(2.38)	7.85(2.07)	29.61(2.62)	14.11
	final	4.47(1.51) •	8.00(2.10) •	5.59(1.55) •	26.25(3.20) •	11.08
	improv	34.17	34.43	28.79	11.35	21.47

(b) Active Learning (Starting with 5% random sampling, until 10% selective sampling)

Data set		*mfeat-pix*	*mfeat-kar*	*mfeat-fac*	*mfeat-fou*	*ave.*
UncertaintySamping	*initial*	11.43(2.08)	18.60(3.01)	11.24(2.31)	35.92(2.99)	19.30
	final	6.76(1.77) •	13.83(2.49) •	8.19(2.12) •	32.56(3.08) •	15.33
	improv	40.86	25.65	27.14	9.35	20.57
QBC	*initial*	10.98(2.10)	18.43(2.32)	11.20(2.48)	35.84(3.22)	19.11
	final	5.24(1.30) •	11.89(2.28) •	6.86(1.94) •	28.90(2.87) •	13.22
	improv	52.28	35.49	38.75	19.36	30.82

(c) Active SSL (Starting with 5% random sampling, until 10% selective sampling then until 60% SSL)

Data set		*mfeat-pix*	*mfeat-kar*	*mfeat-fac*	*mfeat-fou*	*ave.*
US-then-ST	*initial*	6.76(1.77)	13.83(2.49)	8.19(2.12)	32.56(3.08)	15.33
	final	5.05(1.89) •	10.11(2.29) •	5.46(1.71) •	34.73(4.72)	13.84
	improv	25.30	26.90	33.33	-6.66	9.72
US-then-CoBC	*initial*	6.76(1.77)	13.83(2.49)	8.19(2.12)	32.56(3.08)	15.33
	final	3.88(1.41) •	7.50(1.84) •	5.25(1.74) •	26.73(3.14) •	10.84
	improv	42.60	45.77	35.90	17.91	29.29
QBC-then-ST	*initial*	5.24(1.30)	11.89(2.28)	6.86(1.94)	28.90(2.87)•	13.22
	final	5.75(1.72)	13.33(3.55)	5.73(1.61)	34.66(4.78)	14.86
	improv	-9.73	-12.11	16.47	-19.93	-6.33
QBC-then-CoBC	*initial*	5.24(1.30)	11.89(2.28)	6.86(1.94)	28.90(2.87)	13.22
	final	3.38(1.16) •	7.20(2.00) •	4.88(1.48) •	25.05(2.77) •	10.13
	improv	35.50	39.44	28.86	13.32	23.37

(d) Interleaving Active and SSL (Starting with 5% random sampling and until 60%)

Data set		*mfeat-pix*	*mfeat-kar*	*mfeat-fac*	*mfeat-fou*	*ave.*
QBC-with-CoBC	*initial*	10.98(2.10)	18.43(2.32)	11.20(2.48)	35.84(3.22)	19.30
	final	3.40(1.17) •	6.69(1.56) •	4.78(1.31) •	25.48(2.92)	10.08
	improv	69.03	63.70	57.32	28.91	47.25

Table 10.3: Mean and standard deviations of test error rates where *EnsembleLearn = RSM* and *BaseLearn = 1-nearest neighbor*

(a) *CoBC* (Starting with 20% random sampling, until 70% SSL)

Data set		*initial*	*final*	*improv*
ionosphere		16.45(6.23)	13.31(6.38)	19.09
fruits	*colorhist3x3*	8.04(2.92)	6.17(2.61)	23.26
	sobel4x4	20.75(4.27)	16.91(4.28) •	18.51
COIL20	*colorhist3x3*	14.90(2.78)	12.07(2.46) •	18.99
	sobel4x4	8.27(2.72)	3.46(1.38) •	58.16
texture		10.03(2.46	8.03(2.43) •	19.94
ave.		13.07	9.99	23.57

(b) *QBC* (Starting with 10% random sampling, until 20% selective sampling)

Data set		*initial*	*final*	*improv*
ionosphere		20.28 (7.27)	19.30 (5.97)	4.83
fruits	*colorhist3x3*	13.75 (4.75)	4.26 (2.43)	69.02
	sobel4x4	28.40 (4.98)	16.79 (3.93)	40.88
COIL20	*colorhist3x3*	21.72 (2.76)	11.83 (2.03)	45.53
	sobel4x4	16.98 (3.50)	3.60 (1.32)	78.80
texture		13.80 (2.69)	7.92 (2.42)	42.61
ave.		19.16	10.62	46.95

(c) *QBC-then-CoBC* (Starting with 10% random sampling, until 20% selective sampling then until 70% SSL)

Data set		*initial*	*final*	*improv*
ionosphere		19.30 (5.97)	14.45(6.96)•	25.13
fruits	*colorhist3x3*	4.26(2.43)	3.76(2.33)	11.74
	sobel4x4	16.79(3.93)	14.68(3.89)	12.57
COIL20	*colorhist3x3*	11.83(2.03)	9.83(2.07) •	16.91
	sobel4x4	3.60(1.32)	1.53(1.03) •	57.50
texture		7.92(2.42)	6.26(2.14) •	20.96
ave.		10.62	8.42	24.14

(d) *QBC-with-CoBC* (Starting with 10% random sampling and Until 70%)

Data set		*initial*	*final*	*improv*
ionosphere		20.28(7.27)	14.31(6.77) •	29.44
fruits	*colorhist3x3*	13.75(4.75)	5.28(2.51) •	61.60
	sobel4x4	28.40(4.98)	16.94(4.14) •	40.35
COIL20	*colorhist3x3*	21.72(2.76)	11.55(2.26) •	46.82
	sobel4x4	16.98(3.50)	2.85(1.47) •	83.22
texture		13.80(2.69	6.87(2.01) •	50.22
ave.		19.16	9.63	51.94

Table 10.4: Mean and standard deviations of test error rates where *EnsembleLearn* = *RSM* and *BaseLearn* = *C4.5 pruned decision tree* applied to handwritten digits datasets

(a) Passive SSL using *Competence* (Starting with 10% random sampling)

Data set		*mfeat-pix*	*mfeat-kar*	*mfeat-fac*	*mfeat-fou*	*ave.*
ST	*initial*	25.48(3.45)	34.51(4.01)	25.03(2.79)	36.80(4.82)	30.45
	final	22.50(2.81)$^\bullet$	29.20(4.16)$^\bullet$	21.50(3.30)$^\bullet$	34.15(3.76)$^\bullet$	26.84
	improv	11.70	15.39	14.10	7.20	11.86
CoBC	*initial*	16.69(2.82)	21.30(2.96)	15.71(2.52)	29.29(3.50)	20.75
	final	13.18(3.58)$^\bullet$	15.10(2.20)$^\bullet$	14.24(3.33)	25.65(3.03)$^\bullet$	17.04
	improv	21.03	29.11	9.36	12.43	17.88

(b) Active Learning (Starting with 5% random sampling and Until 10%)

Data set		*mfeat-pix*	*mfeat-kar*	*mfeat-fac*	*mfeat-fou*	*ave.*
UncertaintySamping	*initial*	30.99(4.60)	42.04(3.62)	32.63(4.80)	41.74(4.74)	36.85
	final	25.08(3.73)$^\bullet$	34.53(4.36)$^\bullet$	24.53(4.06)$^\bullet$	36.10(4.22)$^\bullet$	30.06
	improv	19.07	17.86	24.82	13.51	18.43
QBC	*initial*	22.78(4.17)	31.53(4.41)	23.23(4.82)	35.10(3.95)	28.16
	final	13.48(2.61)$^\bullet$	19.13(2.77)$^\bullet$	12.26(3.41)$^\bullet$	28.30(2.59)$^\bullet$	18.29
	improv	40.83	39.33	47.22	19.37	35.05

(c) Active SSL (Starting with 5% random sampling plus 5% selective sampling)

Data set		*mfeat-pix*	*mfeat-kar*	*mfeat-fac*	*mfeat-fou*	*ave.*
US-then-ST	*initial*	30.99(4.60)	42.04(3.62)	32.63(4.80)	41.74(4.74)	36.85
	final	20.86(4.31)$^\bullet$	28.04(3.56)$^\bullet$	19.41(3.76)$^\bullet$	33.42(4.09)	25.43
	improv	32.69	33.30	40.51	19.93	30.99
US-then-CoBC	*initial*	30.99(4.60)	42.04(3.62)	32.63(4.80)	41.74(4.74)	36.85
	final	12.21(3.00)	14.49(2.76)	12.88(2.95)	25.99(3.42)	16.39
	improv	60.60	65.53	60.53	37.73	55.52
QBC-then-ST	*initial*	22.78(4.17)	31.53(4.41)	23.23(4.82)	35.10(3.95)	28.16
	final	19.84(3.38)	28.55(3.63)	18.66(3.48)	32.29(2.97)	24.83
	improv	12.91	9.45	19.67	8.01	11.83
QBC-then-CoBC	*initial*	22.78(4.17)	31.53(4.41)	23.23(4.82)	35.10(3.95)	28.16
	final	11.04(2.11)$^\bullet$	14.71(2.26)$^\bullet$	11.95(2.65)	25.10(2.69)$^\bullet$	15.70
	improv	51.54	53.35	48.56	28.49	44.25

(d) Interleaving AL and SSL (Starting with 5% random sampling and until 60%)

Data set		*mfeat-pix*	*mfeat-kar*	*mfeat-fac*	*mfeat-fou*	*ave.*
QBC-with-CoBC	*initial*	22.78(4.17)	31.53(4.41)	23.23(4.82)	35.10(3.95)	28.16
	final	9.11(1.93)$^\bullet$	13.45(2.47)$^\bullet$	10.11(2.45)$^\bullet$	24.38(2.46)$^\bullet$	14.26
	improv	60.01	57.34	56.48	30.54	49.36

Table 10.5: Mean and standard deviations of test error rates where *EnsembleLearn* = *RSM* and *BaseLearn* = *C4.5 pruned decision tree*

(a) *CoBC* (Starting with 20% random sampling, until 70% SSL)

Data set		*initial*	*final*	*improv*
ionosphere		11.69(5.28)	14.03(6.96)	-20.02
fruits	*colorhist3x3*	14.47(4.37)	16.02(3.61)	-10.71
	sobel4x4	19.17(5.03)	17.15(4.28)	10.54
COIL20	*colorhist3x3*	20.44(3.45)	18.84(2.98)	7.83
	sobel4x4	16.69(3.20)	15.11(3.21)	9.47
texture		18.55(3.76)	18.60(4.21)	-0.27
ave.		16.83	16.62	1.25

(b) *QBC* (Starting with 10% random sampling, until 20% selective sampling)

Data set		*initial*	*final*	*improv*
ionosphere		16.96(7.27)	8.69(4.69) •	48.76
fruits	*colorhist3x3*	23.31(5.05)	10.39(3.79) •	55.43
	sobel4x4	28.19(6.21)	16.49(4.07) •	41.50
COIL20	*colorhist3x3*	28.27(3.92	17.37(3.55) •	38.56
	sobel4x4	26.06(4.62)	11.10(3.19) •	57.41
texture		25.19(4.36)	15.96(3.74) •	36.64
ave.		24.66	13.33	45.94

(c) *QBC-then-CoBC* (Starting with 10% random sampling, until 20% selective sampling then until 70% SSL)

Data set		*initial*	*final*	*improv*
ionosphere		8.69(4.69)	11.75(6.78)	-35.21
fruits	*colorhist3x3*	10.39(3.79)	12.09(3.54)	-16.36
	sobel4x4	16.49(4.07)	13.99(4.44)	15.16
COIL20	*colorhist3x3*	17.37(3.55)	16.50(3.32)	5.01
	sobel4x4	11.10(3.19)	10.32(3.23)	7.03
texture		15.96(3.74)	15.48(4.06)	3.01
ave.		13.33	13.36	-0.23

(d) *QBC-with-CoBC* (Starting with 10% random sampling and until 70%)

Data set		*initial*	*final*	*improv*
ionosphere		16.96(7.27)	16.25(7.62)	4.19
fruits	*colorhist3x3*	23.31(5.05)	15.57(4.62) •	33.20
	sobel4x4	28.19(6.21)	17.53(4.92) •	37.81
COIL20	*colorhist3x3*	28.27(3.92)	19.76(3.25) •	30.10
	sobel4x4	26.06(4.62)	12.89(3.31) •	50.54
texture		25.19(4.36)	17.64(3.69) •	29.97
ave.		24.46	16.61	32.67

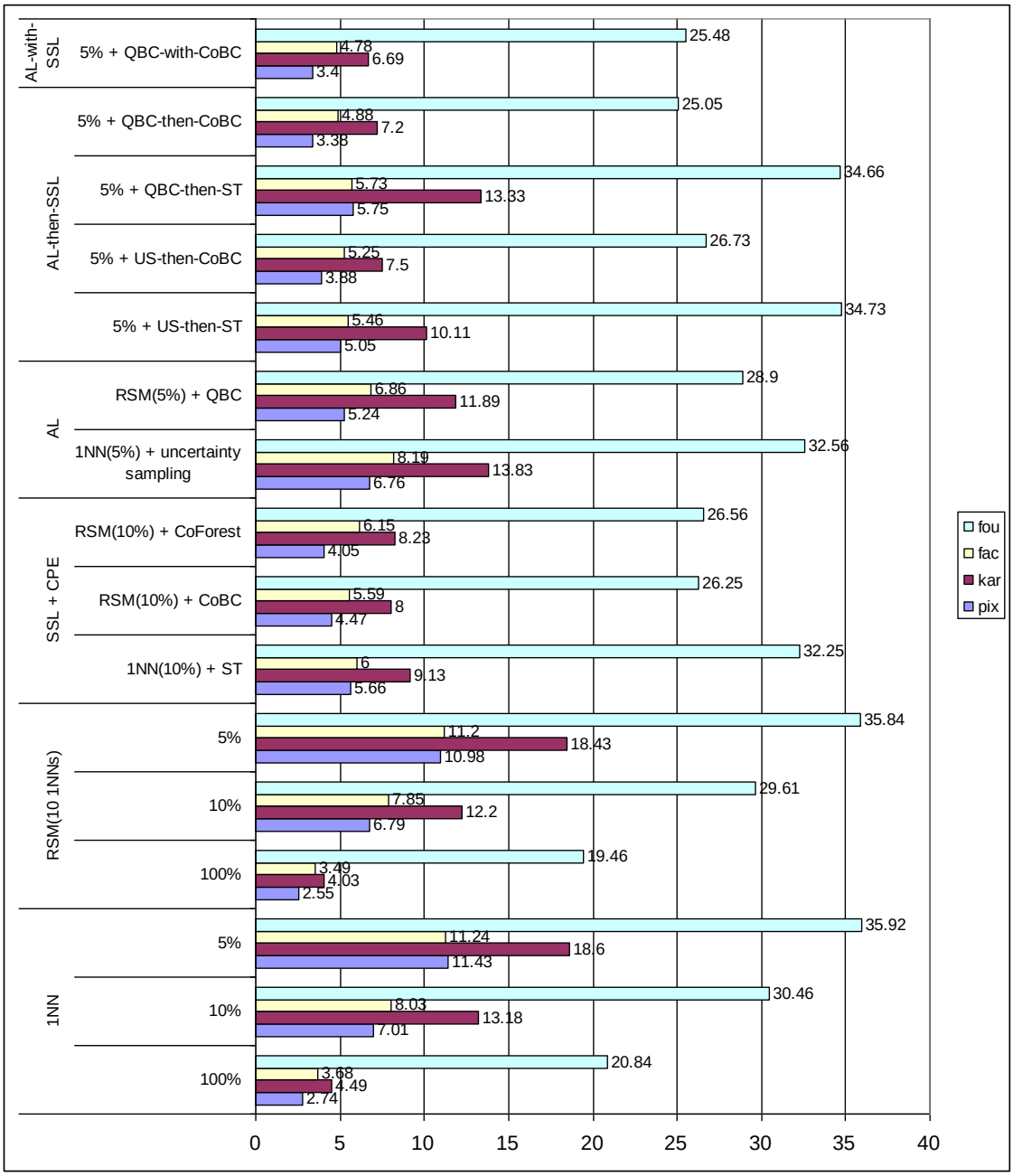


Figure 10.4: Average of test error rates using *1-nearest neighbor* classifier

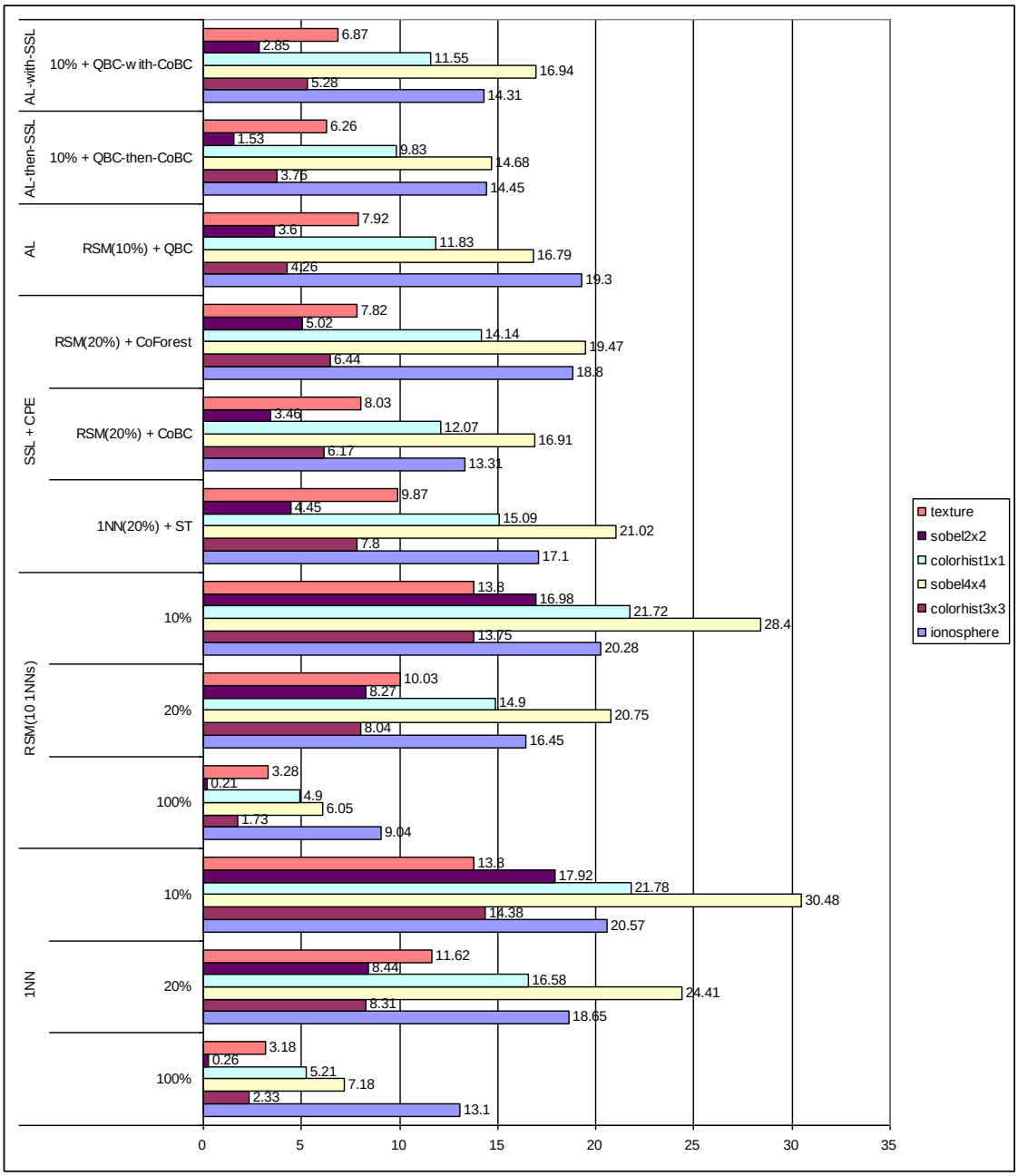


Figure 10.5: Average of test error rates using *1-nearest neighbor* classifier for handwritten digits datasets

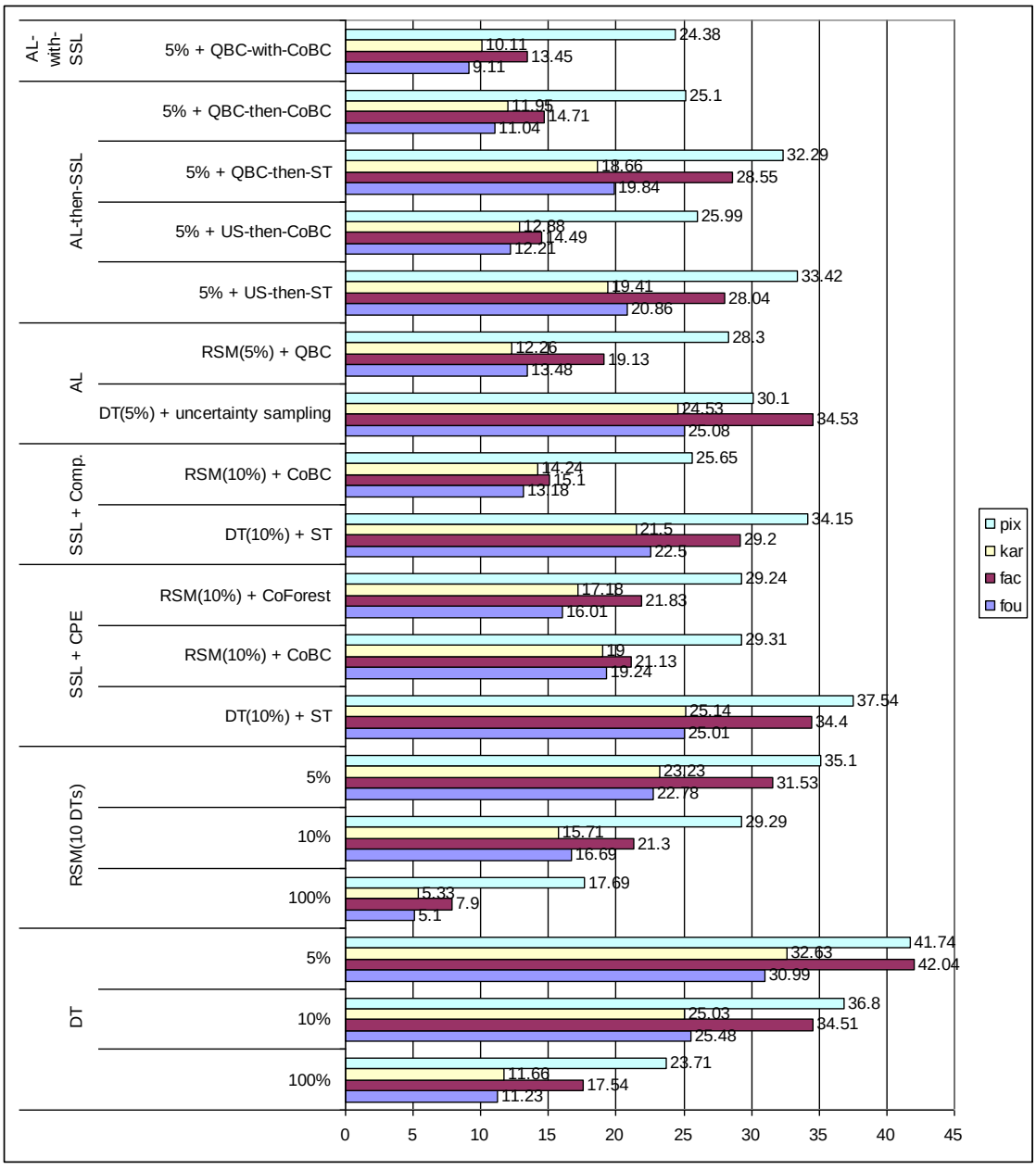


Figure 10.6: Average of test error rates using *C4.5 pruned decision tree* for handwritten digits datasets

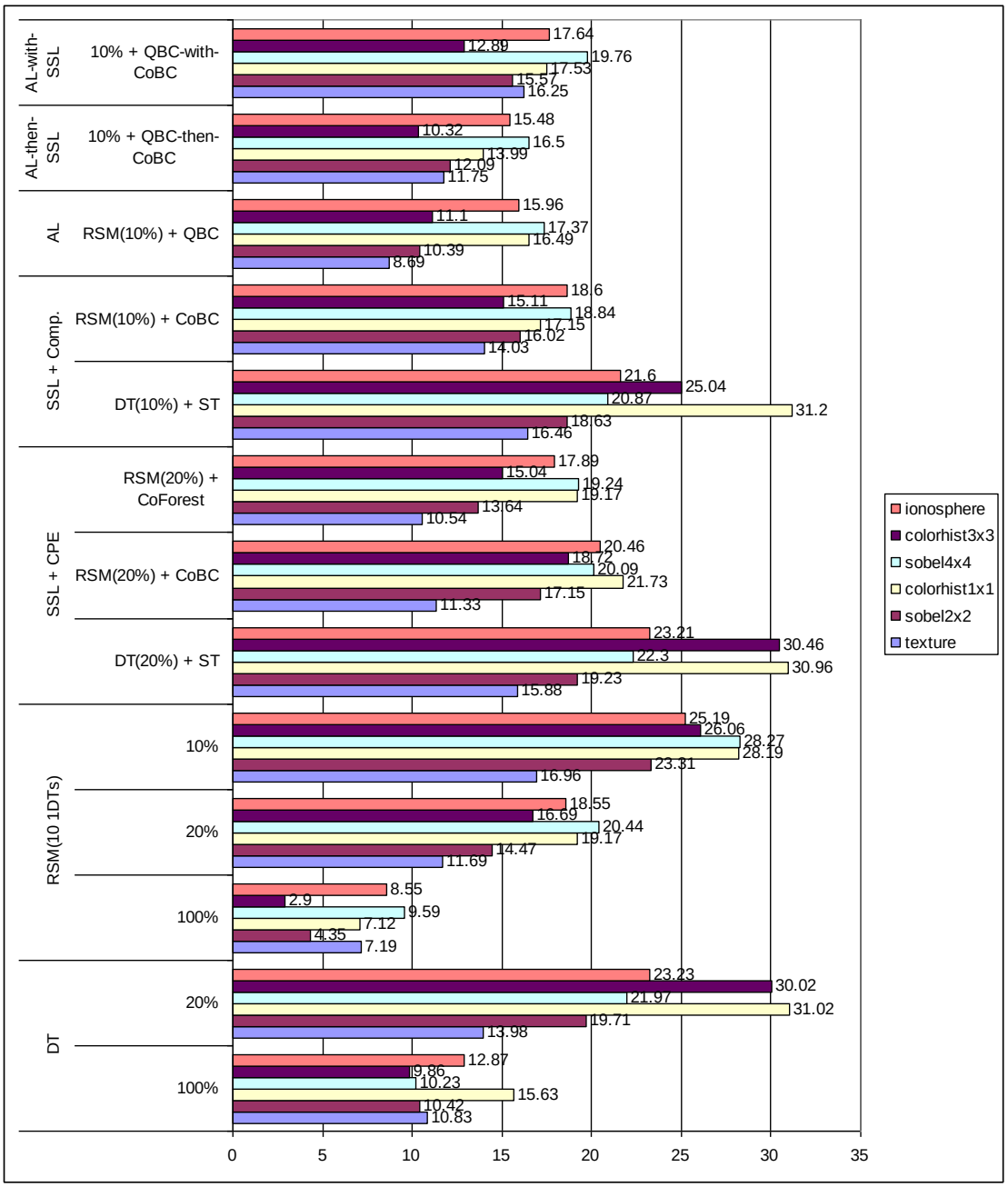


Figure 10.7: Average of test error rates using *C4.5 decision tree*

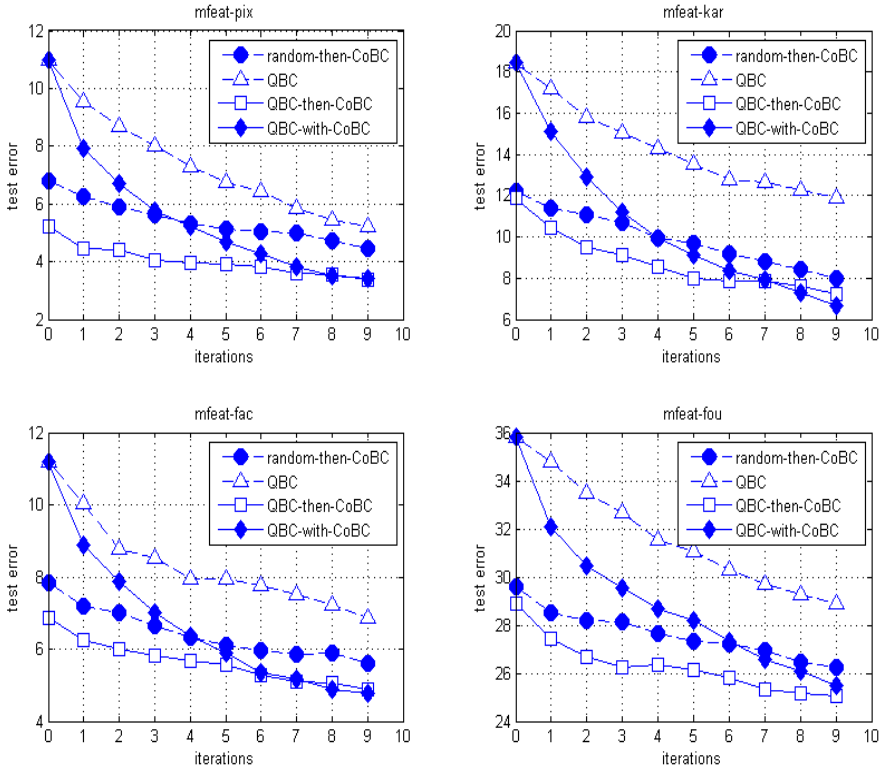


Figure 10.8: Learning curves using *1-nearest neighbor* classifier

10.5.2.2 *CoBC* against Self-Training

The results of comparing the performance of *CoBC* with *Self-Training* are shown in section 9.4.2.2. Using *1-nearest neighbor* classifier, the final ensemble after *CoBC* outperforms the final single classifier after *Self-Training* on all the data sets. Using *C4.5 decision tree* and *local competence* as confidence measure, the final ensemble after *CoBC* is significantly better than the final single classifier after *Self-Training* for all data sets except three where the difference is not statistically significant.

10.5.2.3 QBC against Uncertainty Sampling

The performance of *QBC* (Section 6.4.2) is compared with the single classifier active learning algorithm, i.e., *Uncertainty Sampling* (Section 6.4.1). For a fair comparison, both algorithms are given the same L and U and allowed to label the same amount of unlabeled data. That is, both are initialized with 10% of the training data (for digits data sets, 5% of the training data) that are randomly selected and work until the size of L becomes 20% of the training data (for

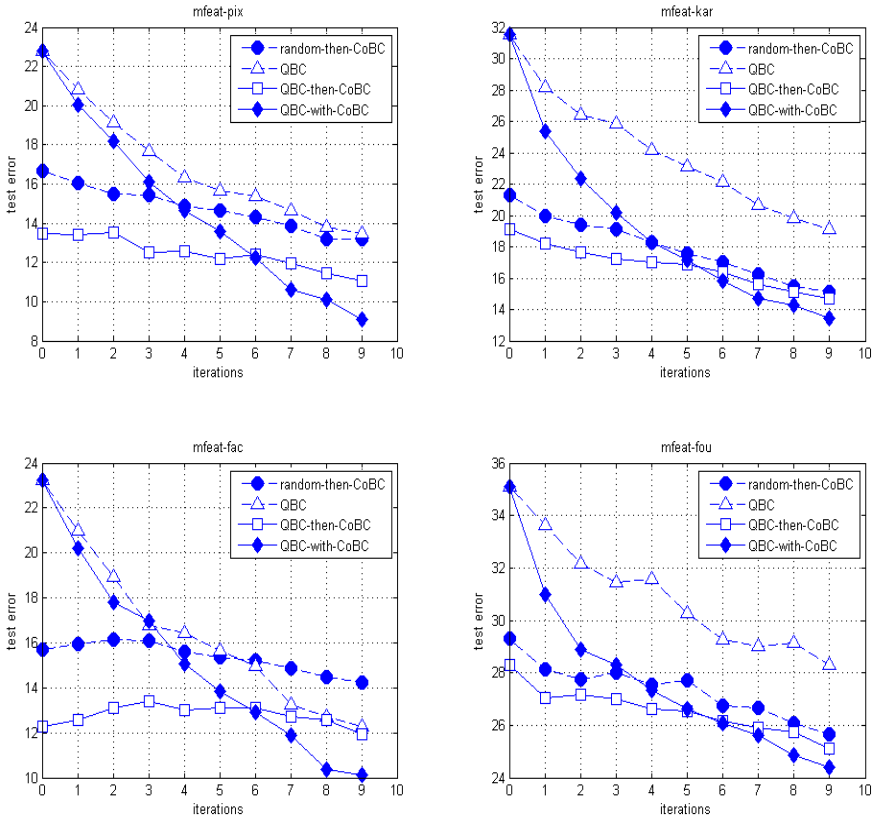


Figure 10.9: Learning curves using *C4.5 pruned decision tree*

digits data sets, 10%). Table 10.2(b) and Table 10.4(b) present the average test error rates on the initial iteration (*initial*), after selecting the most informative examples (*final*) and the relative improvement percentage ($improv = \frac{initial - final}{initial} \times 100$).

Using *1-nearest neighbor* classifier, the final test error rates after *QBC* (on average, 13.22%) are significantly better than the initial error rates on the four digits data sets (on average, 19.11%). The same observation for *Uncertainty Sampling* where the final test error rates (on average, 15.33%) are significantly better than its initial error rates on the four digits data sets (on average, 19.30%). But the final error rates after *QBC* are better than the final error rates after *Uncertainty Sampling* on the four data sets. The average percentage of relative improvement is 30.82% for *QBC* compared to only 20.57% for *Uncertainty Sampling*.

Using *C4.5 decision tree*, the final test error after *QBC* (on average, 18.29%) is significantly better than its initial error rates on the four digits data sets (on average, 28.16%). The same observation for *Uncertainty Sampling*, the final test error rates (on average, 30.06%) are significantly better than its initial error rates

on the digits data sets (on average, 36.85%). But the final test error rates after *QBC* are better than the final error rates after *Uncertainty Sampling* on all the data sets. The average percentage of relative improvement is 35.05% for *QBC* compared to only 18.43% for *Uncertainty Sampling*.

10.5.2.4 QBC-then-CoBC and QBC-with-CoBC

Tables 10.2(c), 10.2(d), 10.3(c) and 10.3(d) present the results using *1-nearest neighbor* base classifier. One can observe the following:

- *QBC-then-CoBC* outperforms *QBC* on all the ten data sets but the improvement is statistically significant on eight data sets.
- *QBC-then-CoBC* outperforms *CoBC* on nine data sets. The improvement is statistically significant on only five data sets.
- *QBC-with-CoBC* outperforms *QBC* on 8 out of the ten data sets. The improvement is statistically significant on five data sets.
- *QBC-with-CoBC* outperforms *CoBC* on 8 out of the ten data sets but the improvement is statistically significant on only one data set.
- *QBC-then-CoBC* performs better than *QBC-with-CoBC* on seven data sets but the improvement is significant for only three data set (*fruits sobel4x4*, *COIL20 colorhist1x1* and *COIL20 sobel2x2*).

Using *C4.5 decision tree* as base classifier, Tables 10.4(c), 10.4(d), 10.5(c) and 10.5(d) present the results. The following can be observed:

- *QBC-then-CoBC* outperforms *QBC* on 8 out of ten data sets. The improvement is statistically significant on only three data sets.
- *QBC-then-CoBC* performs better than *CoBC* on all the ten data sets but the improvement is statistically significant on only three data sets.
- *QBC-with-CoBC* outperforms *QBC* on only the 4 digits data sets. The improvement is statistically significant on only three data sets.
- *QBC-with-CoBC* outperforms *CoBC* on 7 out of the ten data sets but the improvement is statistically significant on only two data sets.
- *QBC-with-CoBC* performs better than *QBC-then-CoBC* on only the four digits data sets where the improvement is significant on only a single data set (*mfeat-pix*). For the other six data sets, *QBC-then-CoBC* outperforms *QBC-with-CoBC* but improvement is statistically significant on a single data set (*COIL20 colorhist1x1*).

10.5.2.5 Other AL and SSL combinations

To verify the advantages of ensemble learning, we implemented three alternative combinations of active and *semi-supervised learning* algorithms: *US-then-ST*, *US-then-CoBC* and *QBC-then-ST*.

- *US-then-ST* trains a single classifier and runs *Uncertainty Sampling*. Then the output informative examples and the original labeled examples are used together to run Self-Training.
- *US-then-CoBC* trains a single classifier and runs *Uncertainty Sampling*. Then it trains an RSM ensemble with both the informative examples resulting from *Uncertainty Sampling* and the original labeled data followed by performing *CoBC*.
- *QBC-then-ST* trains an RSM ensemble and runs *QBC*. Then it trains a single classifier using the informative examples resulting from *QBC* and the original labeled data followed by performing Self-Training.

We sort all the learning algorithms based on the average test error rates for the four digits data sets. For the *1-nearest neighbor* classifier, we get (1) *QBC-with-CoBC* (10.08%), (2) *QBC-then-CoBC* (10.13%), (3) *US-then-CoBC* (10.84%), (4) *CoBC* (11.08%), (5) *Co-Forest* (11.25%), (6) *QBC* (13.22%), (7) *ST* (13.26%), (8) *US-then-ST* (13.84%), (9) *QBC-then-ST* (14.86%) and (10) *US* (15.33%).

For *C4.5 decision tree*, we get (1) *QBC-with-CoBC* (14.26%), (2) *QBC-then-CoBC* (15.70%), (3) *US-then-CoBC* (16.39%), (4) *CoBC* (17.04%), (5) *QBC* (18.29%), (6) *Co-Forest* (21.06%), (7) *QBC-then-ST* (24.83%), (8) *US-then-ST* (25.43%), (9) *ST* (26.84%) and (10) *US* (30.06%). This confirm that combining committe-based active learning with committee-based SSL algorithm is superior to combining it with single-classifier SSL algorithm.

10.6 Conclusions and Future Work

In this chapter, the combination of committee-based *semi-supervised learning* and the state-of-the-art *active learning* algorithms is investigated. I introduced two new approaches, *QBC-then-CoBC* and *QBC-with-CoBC*, that combine the merits of *committee-based active learning* and *committee-based semi-supervised learning*. The first approach is the most straightforward way of combining *CoBC* and active learning where *CoBC* is run after active learning completes (denoted by *QBC-then-CoBC*). The second approach is to run *CoBC* on each *QBC* iteration (denoted by *QBC-with-CoBC*). Experiments were conducted on the ten image recognition tasks used in the previous chapter. The results have shown that:

1. *QBC-then-CoBC* and *QBC-with-CoBC* can enhance the performance of *CoBC* and also outperform other non committee-based combinations of

semi-supervised and active learning algorithms such that *US-then-ST*, *US-then-CoBC* and *QBC-then-ST*.

2. *QBC-then-CoBC* outperforms *random-then-CoBC* because *QBC* provides a better starting point for *CoBC* by selecting informative samples which improve the local competence estimation.

3. Whether *QBC-with-CoBC* outperforms *QBC-then-CoBC* or not depends on the accuracy of the initial ensemble members. It is clear that *CoBC* step starts in *QBC-with-CoBC* approach at earlier iteration than in *QBC-then-CoBC* approach. That is it depends on the number of mislabeled examples added at the early iterations.

In this study, the least confident example is concerned to be the most informative example. Further work should investigate the influence of using other informativeness measures such as vote entropy, margin or Jensen-Shanon divergence (Chapter 6) on the performance of the proposed frameworks. In addition, pool-based sampling is currently used, which assumes that a large amount of unlabeled data can be collected at once before active learning. Future investigations should study combining stream-based sampling with the pool-based one. That is, the training-data distribution can be approximated based on the given labeled training data and the examples newly-labeled by semi-supervised learning. Then an unlabeled example is randomly sampled from the approximated distribution. Then the underlying classifiers decide whether this example is informative or not. This hybrid approach can overcome the drawbacks of both pool-based (Section 6.3) and stream-based sampling (Section 6.2).

Chapter 11

Co-Training by Committee for Semi-supervised Regression

11.1 Introduction

Although the success of semi-supervised learning for classification, there is not much work on *SSL* for regression. Zhou et al. [214] proposed a *Co-Training* style semi-supervised regression algorithm called *CoReg*. This algorithm employs two diverse k-Nearest Neighbor (kNN) regressors that were instantiated using two different values of the Minkowski distance order. The *labeling confidence* is estimated such that the most confidently labeled example is the one which keeps the regressor most consistent with the existing labeled training set.

This chapter presents two major contributions: (1) A new single-view committee-based semi-supervised regression algorithm, called *CoBCReg* that extends the standard *Co-Training* algorithm. It is based on an ensemble of *RBF network* regressors constructed by *Bagging* [31]. (2) A new *Gaussian basis function* that is based on Minkowski distance instead of Euclidean distance, see Figure 11.1. For the effectiveness of *CoBCReg*, there must be some diversity among the committee members and *CoBCReg* should maintain this diversity during the *SSL* process. This is achieved not only by training regressors using different training subsets but also through using different distance measures and different random initialization of the regressors parameters. The applicability of the proposed algorithm is broader than standard *Co-Training* algorithm because it does not require multiple redundant and independent views. The work in this chapter has been previously published ([8]).

11.2 *CoBCReg* Algorithm

There are two potential problems that can prevent any *Co-Training* style algorithm from exploiting the unlabeled data to improve the performance and these

Figure 11.1: The unit circle using Minkowski distance with different distance orders

problems are the motivations for *CoBCReg*. Firstly the outputs of unlabeled examples may be incorrectly estimated by a regressor. This leads to adding noisy examples to the training set of the other regressor and therefore *SSL* will degrade the performance. Secondly there is no guarantee that the newly-predicted examples selected by a regressor as *most confident examples* will be *informative examples* for the other regressor. In order to mitigate the former problem, a committee of predictors is used in *CoBCReg* to predict the unlabeled examples instead of a single predictor. For the latter problem, each regressor selects the most informative examples for itself.

Let $L = \{(x_\mu, y_\mu)\}_{\mu=1}^{m}$ and $U = \{x_\mu\}_{\mu=1}^{n}$ represent the labeled and unlabeled training set respectively, which are drawn randomly from the same distribution where y_i is the target real-valued output for each instance x_i in L while the real-valued outputs of instances in U are unknown. The pseudo-code of *CoBCReg* is shown in Algorithm 17. *CoBCReg* works as follow: initially an ensemble consists of N regressors, which is denoted by H, is constructed from L using *Bagging*. Then the following steps will be repeated until the maximum number of iterations T is reached or U becomes empty. For each iteration t and for each ensemble member h_i, a set U' of u examples is drawn randomly from U without replacement. It is computationally more efficient to use a pool U' instead of using the whole set U. The *SelectRelevantExamples* method (Algorithm 18) is applied to estimate the relevance of each unlabeled example in U' given the *companion committee* H_i. H_i is the ensemble consisting of all member regressors except h_i. A set π_j is created that contains the gr most relevant examples. Then π_j is removed from U' and inserted into the training set of h_i (L_i) such that h_i is refined using the augmented training set L_i. In the prediction phase, the regression estimate for a given example is the weighted average of the outputs of the N regressors created at the final *CoBCReg* iteration.

11.2.1 Diversity Creation

The combination of an ensemble of regressors is only effective if they are diverse. Clearly, if they are identical, then for each regressor, the outputs estimated by the other regressors will be the same as these estimated by the regressor for itself. That is, there is no more knoweldge to be transfered among regressors. In

Algorithm 17 Pseudo Code of CoBC for Regression

Require: set of m labeled training examples (L), set of n unlabeled examples (U), maximum number of Co-Training iterations (T), *ensemble size* (N), pool size (u), growth rate (gr), number of RBF hidden nodes (k), RBF width parameter (α), distance order of the i^{th} regressor (p_i)

Training Phase

for $i = 1$ to N **do**
  $\{L_i, V_i\} \leftarrow BootstrapSample(L)$ {L_i is bag and V_i is out-of-bag}
  $h_i = RBFNN(L_i, k, \alpha, p_i)$
end for
for $t \in \{1 \dots T\}$ **do**
  if U is empty **then** $T = t$-1 and abort loop **end if**
  for $i \in \{1 \dots N\}$ **do**
    Create a pool U' of u examples by random sampling from U
    $\pi_i = SelectRelevantExamples(i, U', V_i, gr)$
    $U' = U' \setminus \pi_i$ and $U = U \cup U'$
  end for
  for $i \in \{1 \dots N\}$ **do**
    if π_i is not empty **then**
      $L_i = L_i \cup \pi_i$
      $h_i = RBFNN(L_i, k, \alpha, p_i)$
    end if
  end for
end for

Prediction Phase

return $H(x) = \sum_{i=1}^{N} w_i h_i(x)$ for a given sample x

regression, ensemble diversity (variance) on an instance x can be quantified by

$$\bar{A}(x) = \sum_{i=1}^{N} w_i (h_i(x) - H(x))^2. \tag{11.1}$$

Brown et al. presented in [36] an exhaustive survey of the various techniques used for creating diverse ensembles. Krogh and Vedelsby [103] introduced the *error-ambiguity decomposition* concept in which the ensemble error (E) is decomposed into two terms, the weighted average error of the ensemble members ($\bar{E}$) and the diversity among their outputs for a given instance ($\bar{A}$). That is, $E = \bar{E} - \bar{A}$. The importance of this decomposition is that it shows us that the average error of the ensemble members should be low while the diversity among them should be high, in order to achieve high ensemble error reduction.

In *CoBCReg*, there are three sources for diversity creation, the *RBF* network regressors are trained using: (1) different bootstrap samples, (2) different random

Algorithm 18 Pseudo Code of of the *SelectRelevantExamples* method

Require: the index of the regressor excluded from the committee (j), pool of u unlabeled examples (U'), validation set (V_j), growth rate (gr)

Calculate validation error of h_j using V_j, ϵ_j
for each $x_u \in U'$ **do**
  $H_j(x_u) = \frac{1}{N-1} \sum_{i=1, i \neq j}^{N} h_i(x_u)$
  $h'_j = RBFNN(L_j \cup \{(x_u, H_j(x_u))\}, k, \alpha, p_j)$
  Calculate validation error ϵ'_j of h'_j using V_j, then $\Delta_{x_u} = (\epsilon_j - \epsilon'_j)/\epsilon_j$
end for
$\pi_j \leftarrow \emptyset$
for gr times **do**
  if there exists $x_u \in U' \setminus \pi_j$ with $\Delta_{x_u} > 0$ **then**
    $\tilde{x}_j = arg\max_{x_u \in U' \setminus \pi_j} \Delta_{x_u}$
    $\pi_j = \pi_j \cup \{(\tilde{x}_j, H_j(\tilde{x}_j))\}$
  end if
end for
return π_j

initialization of *RBF* centers and (3) different distance measures. The Minkowski distance between two D-dimensional feature vectors x_1 and x_2, as defined in Eq. (11.2), is used with different distance order p to train different *RBF* network regressors. In general, the smaller the order, the more robust the resulting distance metric to data variations. Another benefit of this setting, is that, since it is difficult to find in advance the best p value for a given task, then regressors based on different p values might show complementary behavior.

$$\|x_1 - x_2\|_p = \left(\sum_{i=1}^{D} |x_{1i} - x_{2i}|^p \right)^{1/p} \tag{11.2}$$

Unlike *Co-Forest* [119], *CoBCReg* does not hurt the diversity among regressors because the examples selected by a regressor are removed from U. Thus, they can not be selected further by other regressors which keeps the training sets of regressors not similar. Even if the training sets become similar, the regressors could still be diverse because they are instantiated with different distance measures, for some data sets this acts like using different feature spaces.

11.2.2 Confidence Measure

One of the most important factors that affects the performance of any *Co-Training* style algorithm is how to measure the confidence of a given unlabeled example. The inaccurate confidence estimation can lead to selecting and adding mislabeled examples to the labeled training set and therefore might negatively affect the

performance of the *SSL* algorithm. For classification, it is a straightforward task because many classifiers can estimate class posterior probabilities such as Naive Bayes classifier or return real-valued outputs that can be transformed to class probability estimates such as neural networks and decision trees. Assuming that a classifier estimates the probability that an instance x_1 belongs to classes ω_1 and ω_2 is 0.9 and 0.1, respectively, while that for an instance x_2 is 0.6 and 0.4, respectively, then the classifier is more confident that x_1 belongs to classes ω_1 than x_2. Therefore, a *labeling confidence* can be assigned to each unlabeled example using its class probability distribution.

The main challenge for *CoBCReg* is the mechanism for estimating the confidence because the number of possible predictions in regression is unknown. For regression, in [103], variance is used as an effective selection criterion for active learning because a high variance between the estimates of the ensemble members leads to a high average error. Unfortunately, a low variance does not necessarily imply a low average error. That is, it can not be used as a selection criterion for *SSL* because agreement of committee members does not imply that the estimated output is close to the target output. In fact, we will not measure the *labeling confidence* but we will provide another confidence measure called *selection confidence* (See Algorithm 18). The most relevantly selected example should be the one which minimizes the regressor error on the validation set. Thus, for each regressor h_j, create a pool U' of u unlabeled examples. Then, the root mean squared error (*RMSE*) of h_j is evaluated first (ϵ_j). Then for each example x_u in U', h_j is refined with $(x_u, H_j(x_u))$ creating new regressor h'_j. So the *RMSE* of h'_j can be evaluated (ϵ'_j), where $H_j(x_u)$ is the real-valued output estimated by the *companion committee* of h_j (H_j denotes all other ensemble members in H except h_j). Finally, the unlabeled example $\tilde{x}_j$ which maximizes the relative improvement of the *RMSE* (Δ_{x_u}) is selected as the most relevant example labeled by *companion committee* H_j.

It is worth mentioning that the *RMSE*s ϵ_j and ϵ'_j should be estimated accurately. If the training data of h_j is used, this will under-estimate the *RMSE*. Fortunately, since the bootstrap sampling [31] is used to construct the committee, the *out-of-bootstrap* examples are considered for a more accurate estimate of ϵ'_j.

11.2.3 Two-Phase Learning for RBF Networks

The *RBF* network two-phase learning algorithm discussed in Section 2.1.2 is used for training a regressor h_i with *multivariate Gaussian radial basis function* (g) as activation function. At the first phase, the *RBF* centers are determined by performing k-means clustering using the Minkowski distance. The set of Gaussian centers are initialized with training examples randomly selected from L_i. The width of the j^{th} *RBF* neuron (σ_j) is set to the average Minkowski distance between the center c_j and the two nearest Gaussian centers multiplied by α to control the extent of overlap between them. Then, the radial basis function ϕ_j is defined as

follows

$$\phi_j(\|x - c_j\|_p) = exp(-\frac{\|x - c_j\|_p^2}{2\sigma_j^2}). \quad (11.3)$$

At the second phase, the output layer weights W are determined directly by calculating the pseudo-inverse of Φ which provides a least squares solution to the system of linear equations, $T = \Phi W$, where T is the target outputs of the m training examples and $\Phi = (\phi_{\mu j})$ is the activation matrix where

$$\phi_{\mu j} = \phi_j(\|x_\mu - c_j\|_p) \quad (11.4)$$

The gradient-descent error backpropagation learning method is not used, otherwise the computational load will be high. On the other hand, direct computation of W is easier and provides instantaneous training of the network. Therefore, the refinement of regressors with newly-labeled examples can be more efficient.

11.3 Experimental Evaluation

11.3.1 Methodology

An experimental study is conducted to evaluate *CoBCReg* framework on six data sets described in Table 15.1. *Friedman #1*, *#2*, and *#3* have been used by Breiman [31] for evaluating the performance of *Bagging*. *Gabor* and *Multi* have been used by Hansen [77] for comparing several ensemble methods. *Plane* has been used by Ridgeway et al. [155] for investigating the performance of boosted naive Bayesian regressors. All algorithms are implemented using WEKA library [201]. The input features and the real-valued outputs are scaled to [0, 1]. For each experiment, 5 runs of 4-fold cross-validation have been performed. That is, for each data set, 25% are used as test set, while the remaining 75% are used as training examples where 10% of the training examples are randomly selected as the initial labeled data set L while the remaining 90% of the 75% of data are

Table 11.1: Description of the simulated data sets

Data set	Size	Function	Features
Friedman#1	3,000	$y = 10sin(\pi x_1 x_2) + 20(x_3 - 0.5)^2 + 10x_4 + 5x_5$	$x_1, x_2, x_3, x_4, x_5 \sim U[0, 1]$
Friedman#2	5,000	$y = \sqrt{x_1^2 + (x_2 x_3 - (\frac{1}{x_2 x_4}))^2}$	$x_1 \sim U[0, 100]$ $x_2 \sim U[40\pi, 560\pi]$ $x_3 \sim U[0, 1]$ $x_4 \sim U[1, 11]$
Friedman#3	3,000	$y = tan^{-1}\frac{x_2 x_3 - (\frac{1}{x_2 x_4})}{x_1}$	$x_1 \sim U[0, 100]$ $x_2 \sim U[40\pi, 560\pi]$ $x_3 \sim U[0, 1]$ $x_4 \sim U[1, 11]$
Gabor	3,000	$y = \frac{\pi}{2} exp[-2(x_1^2 + x_2^2)]cos[2\pi(x_1 + x_2)]$	$x_1, x_2 \sim U[0, 1]$
Multi	4,000	$y = 0.79 + 1.27x_1 x_2 + 1.56x_1 x_4 + 3.42x_2 x_5 + 2.06x_3 x_4 x_5$	$x_1, x_2, x_3, x_4, x_5 \sim U[0, 1]$
Plane	1,000	$y = 0.6x_1 + 0.3x_2$	$x_1, x_2 \sim U[0, 1]$

Table 11.2: Mean and standard deviation of the test *RMSE* using noise-free functions

	RBFNNs			*CoBCReg*		
Data set	*initial*	*final*	*improv*	*initial*	*final*	*improv*
Friedman#1	0.0817 ± 0.0042	0.0670 ± 0.0032	17.99%	0.0687 ± 0.0035	0.0590 ± 0.0027	14.12%
Friedman#2	0.0417 ± 0.0033	0.0332 ± 0.0024	20.38%	0.0354 ± 0.0028	0.0294 ± 0.0028	16.95%
Friedman#3	0.1019 ± 0.0037	0.0939 ± 0.0038	7.85%	0.0921 ± 0.0049	0.0865 ± 0.0047	6.08%
Gabor	0.0575 ± 0.0108	0.0330 ± 0.0081	42.60%	0.0375 ± 0.0106	0.0202 ± 0.0062	46.13%
Multi	0.0449 ± 0.0037	0.0345 ± 0.0024	23.16 %	0.0373 ± 0.0038	0.0303 ± 0.0025	18.76%
Plane	0.0180 ± 0.0045	0.0093 ± 0.0032	48.33%	0.0136 ± 0.0045	0.0077 ± 0.0032	43.38%
ave.	0.0576	0.0452	26.72%	**0.0474**	**0.0389**	24.24%

used as unlabeled data set U. In the experiments, an initial ensemble of four *RBF network* regressors, $N = 4$, is constructed by *Bagging* where the distance order p_i used by the i^{th} regressor is set to i+1 ($i = 1, 2, 3, 4$). The weights of regressors were uniform, $w_i = 1/N$. We set the pool size u is 50, the growth rate gr is one, the maximum number of iterations T is 30, and for each RBF network the number of *RBF*s k is set to 20 and α is set to 2.0.

11.3.2 Results

Table 11.2 shows the average of the *RMSE*s of the four RBF Network regressors used in *CoBCReg* and the *RMSE* of *CoBCReg* on the test set at iteration 0 (*initial*) trained only on the 10% available labeled data L, after the 30^{th} *SSL* iteration of exploiting the unlabeled data set U (*final*) and the relative improvement percentage on *RMSE* (*improv* $= \frac{initial - final}{initial}$). Figure 11.2 shows the *RMSE* of *CoBCReg* (*CoBCReg*), and the average of the *RMSE*s of the four regressors used in *CoBCReg* (*RBFNNs*) at the different *SSL* iterations. The dash and solid horizontal lines show the average of the *RMSE*s of the four regressors and the *RMSE* of the ensemble trained using only the 10% labeled data, respectively, as a basline for the comparison. The dash-dot horizontal line represents the *RMSE* of the committee trained using all the training data 100% labeled as another baseline. Paired t-test with 0.05 significance level indicates that the final regression estimates of *CoBCReg* are significantly better than its initial estimates on all the data sets. In addition, for all data sets both the *initial* and *final RMSE* of *CoBCReg* (E) (on average, 0.0474 and 0.0389) is less than that of the average of *RMSEs* of its members ($\bar{E}$). Therefore, *CoBCReg* can exploit the unlabeled examples to improve the generalization error of the committee and it does not hurt the diversity among the committee members during the *SSL* process ($\bar{A} > 0$).

11.3.3 Influence of Output Noise

In order to study the robustness of *CoBCReg* to noise, we added Gaussian noise to the target functions of *Friedman #1, #2, #3*, *Gabor*, *Multi* and *Plane* that is distributed as $N(0, 1.0^2)$, $N(0, 125^2)$, $N(0, 0.1^2)$, $N(0, 0.15^2)$, $N(0, 0.35^2)$, and

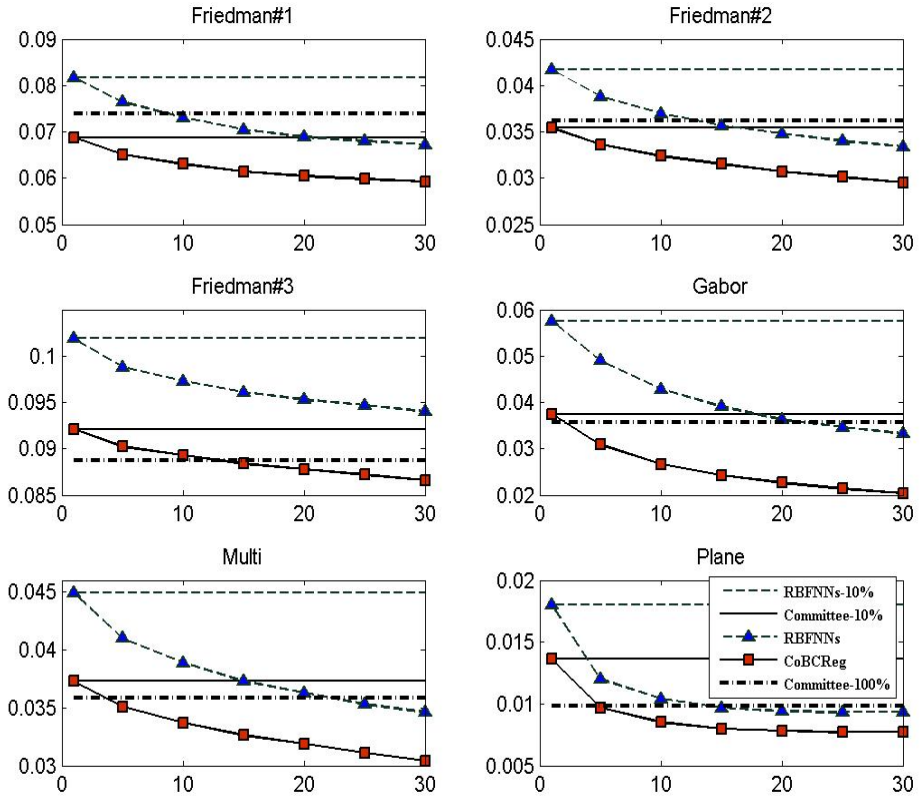


Figure 11.2: The average of test *RMSE* at different iterations using noise-free functions

$N(0, 0.05^2)$, respectively, where standard deviation is selected to give 3:1 *signal-to-noise* ratio (i.e., the ratio of the standard deviations). Thus, the variance of the function itself (without noise) accounts for 90% of the total variance. Table 11.3 present the *initial* and *final* average of the *RMSE*s of the four regressors used in *CoBCReg* ($\bar{E}$) and the *RMSE* of *CoBCReg* (E) for the noisy functions. Figure 11.4 shows the performance at the different *SSL* iterations. Again the final regression estimates of *CoBCReg* significantly outperform its initial estimates on all the data sets except on *Plane* where the improvement is not significant. In addition, both the *initial* and *final* E (on average, 0.0682 and 0.0646) is less than that $\bar{E}$. Although *highly noise* problems are used, *CoBCReg* can still exploit the unlabeled examples to improve the regression estimates on all data sets. It is worth noting that *CoReg*, proposed in [214], was applied on the same data sets and both the absolute *RMSE* and the relative improvement achieved by *CoBCReg* are better than that of *CoReg* on all data sets. Box-plots of the test committee *RMSE* for both the noisy and noise-free functions are given in Figure 11.3.

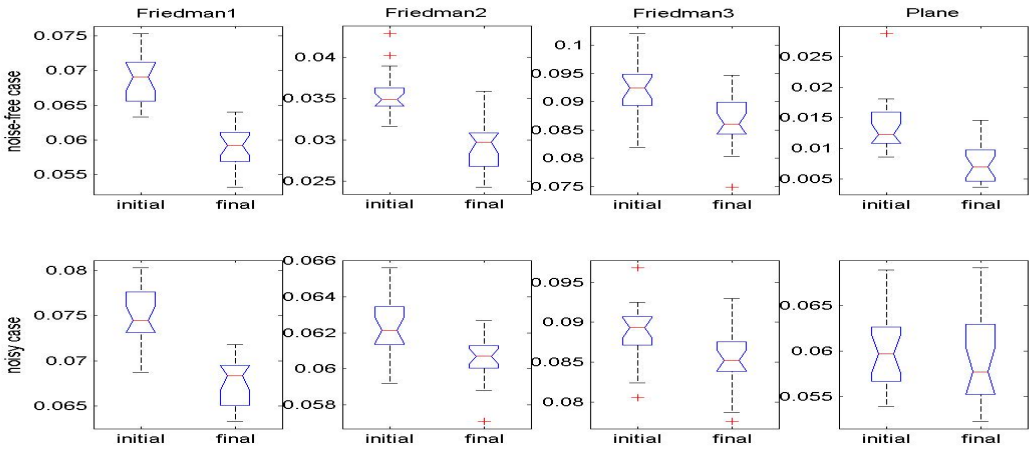


Figure 11.3: Box plots of the test *RMSE* before and after *CoBCReg* using noise-free functions and noisy functions. Notches indicates robust estimates of median.

Table 11.3: Mean and standard deviation of the test RMSE using noisy functions.

	RBFNNs			*CoBCReg*		
Data set	*initial*	*final*	*improv*	*initial*	*final*	*improv*
Friedman#1	0.0860 ± 0.0037	0.0745 ± 0.0026	13.37%	0.0748 ± 0.0035	0.0677 ± 0.0025	9.49%
Friedman#2	0.0669 ± 0.0022	0.0635 ± 0.0013	5.08%	0.0624 ± 0.0016	0.0607 ± 0.0013	2.72%
Friedman#3	0.0962 ± 0.0031	0.0904 ± 0.0029	6.03%	0.0887 ± 0.0036	0.0852 ± 0.0036	3.95%
Gabor	0.0739 ± 0.0073	0.0615 ± 0.0041	16.78%	0.0602 ± 0.0059	0.0541 ± 0.0025	10.13%
Multi	0.0690 ± 0.0029	0.0646 ± 0.0024	6.37%	0.0632 ± 0.0030	0.0607 ± 0.0024	3.96%
Plane	0.0685 ± 0.0055	0.0621 ± 0.0051	9.34%	0.0599 ± 0.0040	0.0592 ± 0.0049	1.34%
ave.	0.0905	0.0770	14.92%	**0.0682**	**0.0646**	5.28%

11.4 Conclusions and Future Work

For regression tasks, labeling the examples for training is a time consuming, tedious and expensive process. Such burden could be alleviated if the regression learning algorithm can exploit the unlabeled data during learning. In this chapter, a *Co-Training* style framework called *CoBCReg* is proposed. *CoBCReg* relax the hard requirements of standard *Co-Training* algorithm through using an ensemble of N diverse regressors instead of a set of redundant and independent views. At each iteration and for each regressor, the companion committee labels the unlabeled examples then the regressor select the most informative newly-labeled examples for itself, where the selection confidence is based on estimating the validation error. The final prediction is the average of the estimates of the N regressors.

CoBCReg is more applicable than the standard *Co-Training* because it does not require sufficient and independent views to construct diverse regressors. However, it depends on three mechanisms to create the diversity, initial regressors are

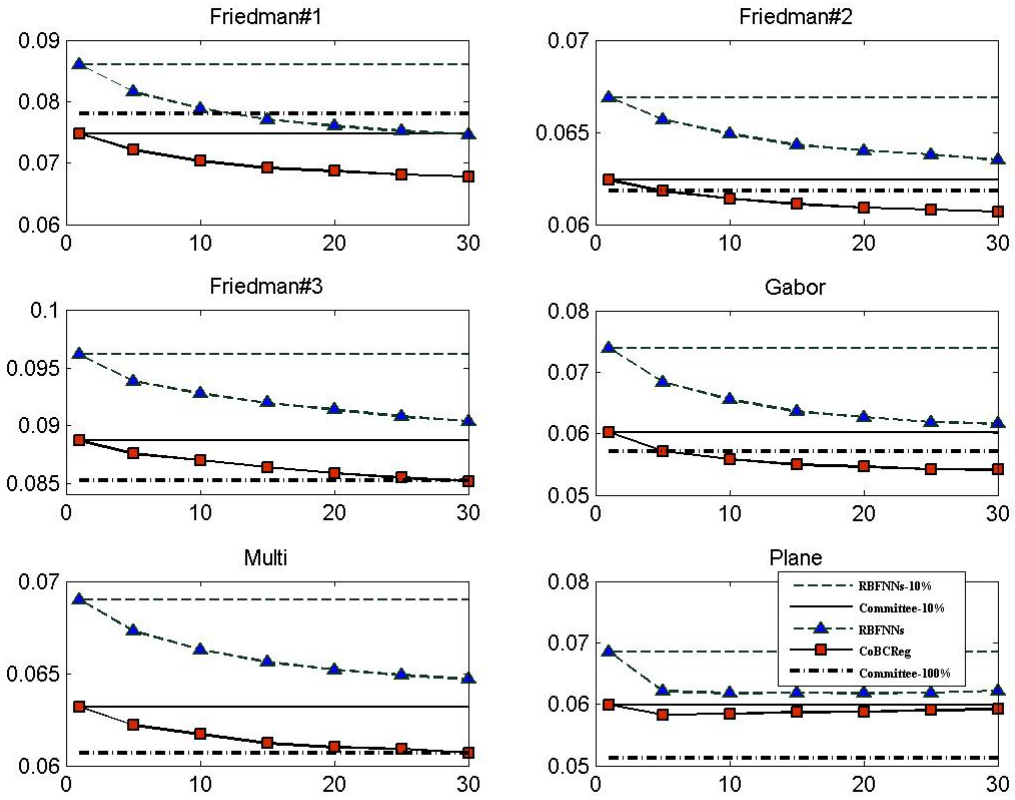


Figure 11.4: The average of test *RMSE* at different iterations using noisy functions

trained using different bootstrap samples with different random initialization of *RBF* centers and are using different Minkowski distance orders. Experimental results show that *CoBCReg* can effectively exploit the unlabeled examples to improve the generalization error and it is robust to output noise. There are many interesting directions for future work.

- Apply *CoBCReg* using other types of regressors such as decision tree and support vector machines.
- Investigate other diversity creation methods such as using *AdaBoost.RT* [177] ensemble method, an *AdaBoost* variant for regression, that will extend the idea of *CoBCReg*.
- Explore other confidence measures that are more efficient and effective.
- The theoretical analysis of *CoBCReg* is necessary because it will show when and why the algorithm works.

- The combination of committee-based active learning and semi-supervised learning works effectively for classification [4]. The current implementation of *CoBCReg* framework depends on a labeled training set that is a random sample of the unlabeled data. Further work should investigate the influence of providing *CoBCReg* with a better starting point through combining it with *Query by Committee* [103].

Chapter 12

One-against-One Co-Training with Tri-Class SVMs

12.1 Introduction

Multi-class decomposition (Chapter 4) is a supervised learning task that requires an appropriate amount of labeled data in order to achieve high classification accuracy. However, the data labeling process is often difficult, expensive, or time consuming, as it requires the efforts of human experts. In chapter 8, two learning frameworks are proposed to integrate the unlabeled data into the tree-structured approach where RBF networks are used as binary classifiers. In this chapter, a learning framework is introduced to exploit the unlabeled examples into the one-against-one approach (Section 4.3). This chapter and chapter 8 have the same objective which is to benefit from the unlabeled data in multi-class learning. First, multi-class problem is decomposed into a set of binary problems and then *Co-Training* is used to exploit unlabeled data in solving each binary problem. An important factor for an effective *Co-Training* algorithm is how to measure class prediction confidence. Thus, a new probabilistic interpretation of the outputs of Tri-Class Support Vector Machine (SVM) is introduced where the confidence is derived from the predicted class probabilities. The main advantage of *Tri-Class SVM* is that it can discriminate between uncertainty and ignorance so it can reject the examples that do not belong to its target classes. In addition, a variant of the *Sequential Minimal Optimization* (SMO) algorithm is introduced for faster learning of the *Tri-Class SVM*s since *Co-Training* is an iterative method.

The effectiveness of the proposed framework is evaluated on facial expressions recognition from image sequences. A task that involves a large number of classes and a small amount of labeled data because human annotation of facial expressions is difficult. The results have shown that *Co-Training* with an ensemble of three multi-view *Tri-Class SVM*s can automatically improve the recognition rate using a small amount of human-labeled videos which minimize the cost of data labeling. The Gaussian Mixture Model (GMM) approach is used to extract the

features, called *super vectors*, from facial expression videos. These *GMM super vectors* are the input of *Tri-Class SVM*s. The work in this chapter has been previously published ([2]).

12.2 One-against-One Co-Training

12.2.1 Motivation

Support Vector Machines are discriminative classifiers that model the decision boundary between classes. They use the notion of *classification margin* and attempt to maximize the margin of all (or most) training examples. The most confident unlabeled examples with respect to a single machine can not be informative because these examples have a large margin and therefore have little impact on its decision boundary. For discriminative classifiers one must find unlabeled examples which have a small margin because these examples are the informative ones that can change the separating hyperplane (Figure 12.1).

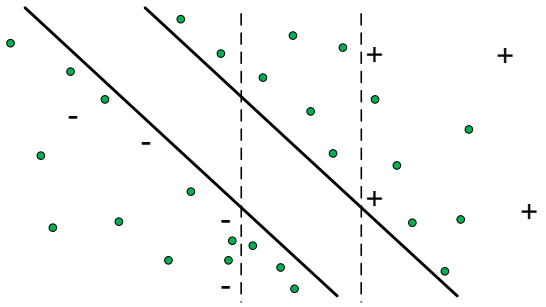


Figure 12.1: Graphical illustration of SVM: The unlabeled examples help to put the decision boundary in low density regions. Using labeled data only, the maximum margin separating hyperplane is plotted with the versicle dashed lines. Using both labeled and unlabeled data (dots), the maximum margin separating hyperplane is plotted with the oblique solid lines.

To address this problem, *Co-Training* (Section 5.7.1.1) is used that requires an ensemble of two or more diverse classifiers instead of a single classifier. Since the margins assigned by the co-trained classifiers are not directly related, there may exist a set of examples with high average margin with respect to the ensemble that have a small or negative margin with respect to an individual machine. That is some examples which would be confidently labeled by the ensemble that would be misclassified by one of the classifiers. Thus individual classifiers can exchange knowledge among each other in the form of additional informative labeled examples.

12.2.2 Co-Training with Tri-Class SVMs

The framework is formally defined in Algorithm 19. A given multi-class data set is decomposed into $K(K-1)/2$ binary-class data sets (L_{kh}), one for each pair of classes (ω_k, ω_h), for each $k, h = 1, \ldots, K$. An ensemble of support vector machines are *co-trained* to discriminate between the two classes using the examples in L_{kh} that belong to class ω_k labeled with $y = 3$, those belonging to ω_h labeled with $y = 1$ and the other examples labeled with $y = 2$, see Table 12.1. The pseudo-code of *Tri-Class Co-Training* is shown in Algorithm 20 and Figure 12.2 illustrates the data flow of the algorithm.

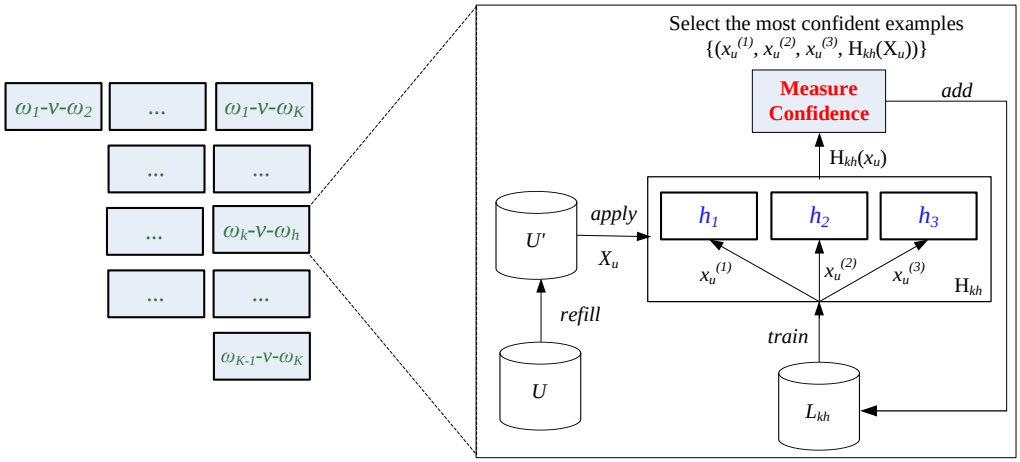


Figure 12.2: Tri-Class Co-Training

Algorithm 19 One-against-One Co-Training

Require: set of m labeled training examples (L), set of unlabeled examples (U), maximum number of *Co-Training* iterations (T), number of unlabeled examples in the pool (u), set of classes ($\Omega = \{\omega_1, \ldots, \omega_K\}$),combination method (*Combiner*)

Training Phase

1: **for** $k = 1$ to $K - 1$ **do**
2: **for** $h = k + 1$ to K **do**
3: Relabel the training examples L, $L_{kh} = \{(X, y)|(X, t) \in L$ and $y = 1$ if $t = \omega_h$, $y = 2$ if $t \in \Omega - \{\omega_k, \omega_h\}$ and $y = 3$ if $t = \omega_k\}$
4: Train binary classifier,
H_{kh}= *Co-Training*(L_{kh}, U, T, u) (see Algorithm 20)
5: **end for**
6: **end for**

Prediction Phase

7: **return** *Combiner*$(x, \{H_{kh}\}_{k=1,h=k+1}^{K})$ for a given x (see Eq. (12.58))

Given a labeled data set $L_{kh} = \{(X_\mu, y_\mu)|X_\mu = (x_\mu^{(1)}, x_\mu^{(2)}, x_\mu^{(3)})|x_\mu^{(i)} \in \mathbb{R}^{D_i}, y_\mu \in \{1,2,3\}, \mu = 1,\ldots,m\}$ and $U = \{X_u = (x_u^{(1)}, x_u^{(2)}, x_u^{(3)})|x_u^{(i)} \in \mathbb{R}^{D_i}, u = 1,\ldots,n\}$ the set of unlabeled data where each example is represented by three D_i-dimensional feature vectors. For each view i, an initial *Tri-Class SVM* $h_i^{(0)}$ is trained (Algorithm 21) using the available labeled data $V_i(L)$. Then the following steps are repeated for a given number of iterations T or until the U becomes empty. For each iteration t, a pool U' is created of u examples randomly drawn from U without replacement. It is computationally more efficient to use a pool U' instead of using the whole set U.

Algorithm 20 Tri-Class Co-Training

Require: set of m labeled training examples (L_{kh}), set of unlabeled examples (U), three sets of features (V_1,V_2,V_3), maximum number of co-training iterations (T), number of unlabeled examples in the pool (u), prior probability of each class ω_k ($\{Pr_k\}_{k=1}^K$)

Training Phase

1: Construct initial classifiers, $h_1^{(0)} = TriClassSMO(V_1(L_{kh}))$, $h_2^{(0)} = TriClassSMO(V_2(L_{kh}))$ and $h_3^{(0)} = TriClassSMO(V_3(L_{kh}))$ (see Algorithm 21)
2: **for** $t = 1$ to T **do**
3: **if** U is empty **then** $T = t$-1 and abort loop **end if**
4: Create a pool U' of u examples from U
5: **for** each $x_u \in U'$ **do**
6: Estimate the ensemble class probabilities as defined in Eq.(12.1)
7: Measure the prediction confidence as defined in Eq. (12.2)
8: **end for**
9: Rank the examples in U' by confidence
10: Create a subset π_t that contains the $n_y \propto Pr_y$ most confident examples assigned to each class $y \in \{1,2,3\}$
11: $U' = U' \setminus \pi_t$ % remove π_t from U'
12: $L_{kh} = L_{kh} \cup \pi_t$ % add π_t to L_{kh}
13: $U = U \cup U'$ % return the rest to U
14: Re-train the machines, $h_1^{(t)} = TriClassSMO(V_1(L_{kh}))$, $h_2^{(t)} = TriClassSMO(V_2(L_{kh}))$ and $h_3^{(t)} = TriClassSMO(V_3(L_{kh}))$
15: **end for**

Prediction Phase

16: **return** $P^{(T)}(y|x)$ as defined in Eq. (12.1), for a given example x

For each example $X_u = (x_u^{(1)}, x_u^{(2)}, x_u^{(3)}) \in U'$, each *Tri-Class SVM* $h_i^{(t-1)}$ is applied in order to predict the class membership probability $P_i^{(t-1)}$ of $x_u^{(i)}$ as defined in Eq. (12.55) to Eq. (12.57). Afterward the final output of the ensemble H_{kh}, denoted as $P_{kh}^{(t-1)}$, is the average of the probabilities estimated by the three

*Tri-Class SVM*s. That is, for each class $y \in \{1, 2, 3\}$,

$$P_{kh}^{(t-1)}(y|x_u) = \frac{1}{3}\sum_{i=1}^{3} P_i^{(t-1)}(y|x_u). \tag{12.1}$$

12.2.3 Confidence Measure

Thus, the confidence of the ensemble constructed at the $(t-1)^{th}$ iteration in predicting the class label of X_u is the maximum average class probability.

$$Confidence(X_u, H_{kh}^{(t-1)}) = \max_{1 \le y \le 3} P_{kh}^{(t-1)}(y|X_u) \tag{12.2}$$

Afterward, the unlabeled examples are ranked by the confidence in the class prediction. A set π_t is created that contains the n_y most confident examples assigned to each class y by ensemble of machines. Then π_t is moved from the pool U' to the training set L_{kh}. Then the three classifiers are retrained using the augmented training set. The aim is that the confident examples with respect to the ensemble can be informative examples with respect to any of the individual classifiers (it has a small margin). In the classification phase, the final output for a given example is the average of the probabilistic outputs of the three classifiers created at the final *Co-Training* iteration, $h_1^{(T)}$, $h_2^{(T)}$ and $h_3^{(T)}$.

Unfortunately, Binary-Class SVM for pair (ω_k, ω_h) can not discriminate the unlabeled examples in U' that belong neither to ω_k nor to ω_h and these examples may have a margin larger that those belong to ω_k nor to ω_h (see Figure 12.3). Thus π_t may contain newly labeled examples that belong neither to ω_k nor to ω_h that have no impact on the decision boundary. To avoid this problem, I used *Tri-Class SVM* (Section 12.3.2) instead of Binary-Class SVM because *Tri-Class SVM* can reject the undesired examples.

12.3 Support Vector Machines (SVM)

Given a labeled training set $L = \{(x_i, y_i)|x_i \in \mathbb{R}^d, y_i \in \Omega, i = 1, \ldots, m\}$ where $\Omega = \{\omega_1, \ldots, \omega_K\}$ is a predefined set of classes. Let $L_k = \{(x_i, y_i) \in L|y_i = \omega_k\}$ be the set of $n_k = |L_k|$ d-dimensional training examples belonging to class ω_k. Usually multi-class decomposition schemes such as one-against-one, one-against-others and ECOC have been applied. A multi-class decomposition scheme consists of three stages: (1) the multi-class problem is divided into a set of simpler binary-class problems, (2) an ensemble of binary classifiers such as SVMs is constructed where each machine takes only two classes in consideration. (3) the outputs of individual SVMs are combined to yield the final decision of the ensemble. In this study, the one-against-one scheme (Section 4.3) is considered where $\frac{K(K-1)}{2}$ binary classifiers are trained to generate hyperplanes f_{kh} that separate between

L_k and L_h. If f_{kh} is the optimal hyperplane, then $sign(f_{kh}(x_i))$=1, for $x_i \in L_k$ and $sign(f_{kh}(x_i))$= -1, for $x_i \in L_h$. Note that the remaining training examples $L - \{L_k \cup L_h\}$ are not considered in the optimization problem. If a hyperplane f_{kh} must classify an example x that belongs neither to ω_k nor to ω_h, the correct decision is $f_{kh}(x)$ =0 which means that the f_{kh} rejects the example x. In order to add this reject option, f_{kh} must be enforced to produce output $f_{kh}(x)$ =0 for all the training examples x belonging to different classes from ω_k and ω_h.

12.3.1 Binary-Class SVMs

Let $L_k \cup L_h$ be the set of training examples (x_i, y_i) that belong to Class ω_k or ω_h and the associated labels be $s_i = 1$ for y_i=ω_k and -1 for y_i=ω_h, see Figure 12.3. In a support vector machine (Section 2.4), the optimal hyperplane is required to minimize the generalization error. But the classifier may not have high generalization ability if the training data are not linearly separable. In order to improve linear separability, the training data are mapped from the original d-dimensional *input space* using nonlinear function ϕ into a higher D-dimensional space called the *dot product feature space*. Thus the decision function in the D-dimensional feature space would be

$$f_{kh}(x) = \langle w, \phi(x) \rangle - b, \text{ where margin} = \frac{2}{\|w\|^2} \tag{12.3}$$

where $w \in \mathbb{R}^D$ is a vector orthogonal to the hyperplane; $b \in \mathbb{R}$ is a bias term.

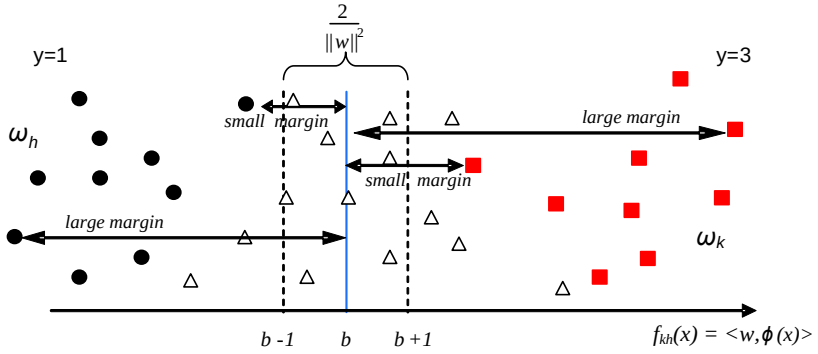


Figure 12.3: An illustration of the hyperplane that discriminates between ω_k and ω_h

The optimal separating hyperplane (w^*, b^*) with maximum margin can be obtained by minimizing

$$\frac{1}{2}\|w\|^2 + C \sum_{i=1}^{n_k+n_h} \epsilon_i \tag{12.4}$$

subject to the constraints

$$y_i(\langle w, \phi(x_i) \rangle - b) \geq 1 - \epsilon_i, \quad \epsilon_i \geq 0, \quad for \quad i = 1, \ldots, n_k + n_h \tag{12.5}$$

where ϵ_i are slack variables that permit margin failure and C is a regularization parameter which trades off between margin maximization and margin errors minimization. The training examples that satisfy the equalities are called *the support vectors*. Then the output of an SVM is explicitly computed from the Lagrange multipliers α_i that are obtained by solving the above optimization problem:

$$f_{kh}(x) = \sum_{i \in S} \alpha_i y_i \mathbb{K}(x, x_i) - b, \tag{12.6}$$

where S is the set of support vector indices and $\mathbb{K}$ is a kernel function that measures the similarity between the input vector x and the training vector x_j. The advantage of using kernels is that there is no need to deal with the high dimensional feature space explicitly. This technique is called *the kernel trick* as $\mathbb{K}(x, x_i) = \langle \phi(x), \phi(x_i) \rangle$ (Section 2.4).

12.3.2 One-against-One Tri-Class SVMs

Shashua and Levin [176] have recently developed an approach to deal with ordinal regression problems using SVMs. This approach considers all the classes at once and trains parallel hyperplanes that separate consecutive classes. The main drawback is that, all the hyperplanes considered must be parallel, hence the explanation power of the machine is reduced, and the use of the machine is restricted to ordinal regression. Angulo et al. [16] introduced the *One-against-One Tri-class SVMs* approach that extends the idea of the ordinal regression approach in [176]. At each step, it considers a pair of classes ω_k and ω_h and trains two parallel hyperplanes that separate L_h, $L - \{L_k \cup L_h\}$ and L_k, respectively where the training set L is divided into three groups, labeled 1, 2, 3 (see Figure 12.4 and Table 12.1). Figure 12.5 illustrates the ensemble of *1-v-1 Tri-class SVMs* applied to a linearly separable dataset with 45 examples [16]. This approach improves the explanation power of the machines because it does not enforce the hyperplanes to be parallel. The size of the optimization problem associated to the *Tri-Class SVM* has been drastically reduced. Hence, if each class has the same number of examples, (i.e. $\frac{m}{K}$ examples for classes ω_k and ω_h and $\frac{m(K-2)}{K}$ examples for the remaining classes labeled 2), the optimization problem has to fulfill a number of $O(m)$ constraints. When all the necessary One-against-One Tri-Class machines are considered, $\frac{K(K-1)}{2}$, then the total number of constraints is $O(K^2 m)$.

Table 12.1: Code matrix

	f_{12}	f_{13}	f_{14}	f_{23}	f_{24}	f_{34}
ω_1	3	3	3	2	2	2
ω_2	1	2	2	3	3	2
ω_3	2	1	2	1	2	3
ω_4	2	2	1	2	1	1

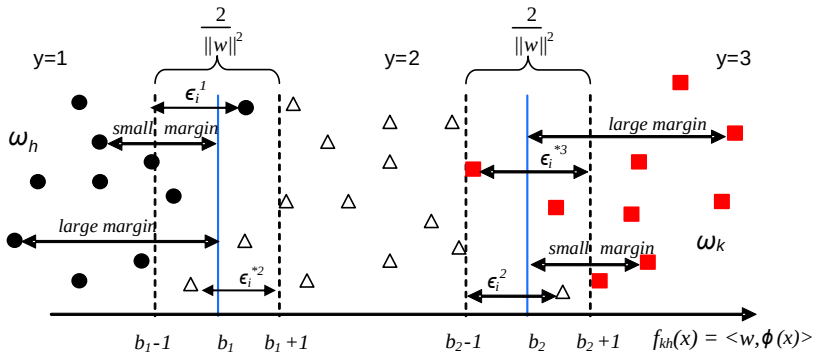
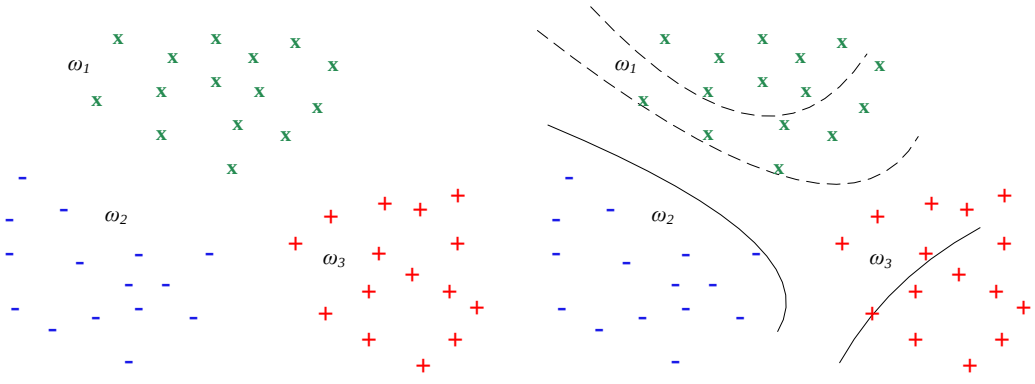
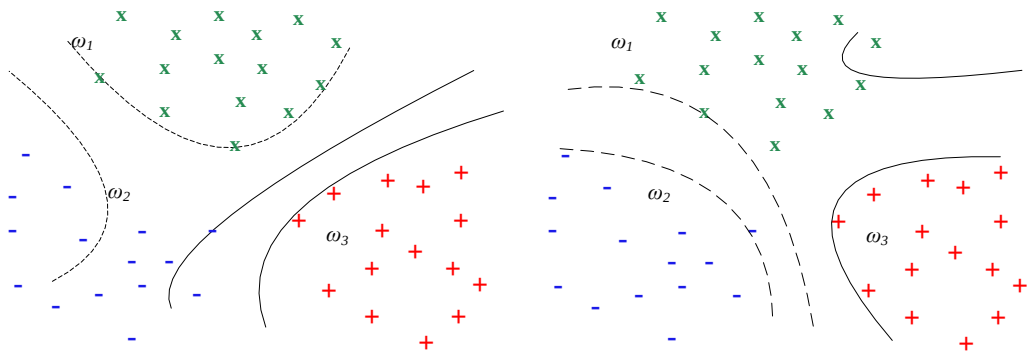


Figure 12.4: An illustration of the two hyperplanes that discriminate between ω_k and ω_h



(a) *input space* (b) Class ω_1 against ω_2



(c) Class ω_1 against ω_3 (d) Class ω_2 against ω_3

Figure 12.5: 1-v-1 Tri-class SVMs applied to a linearly separable dataset with 45 examples [16]. The solid lines represent the first hyperplane and the dashed lines represent the second hyperplane in the input space.

12.3.2.1 Primal problem

The objective of *Tri-Class SVM* is to find the direction w and the positions b_1 and b_2 of the first and second hyperplanes such that the margins between classes $y = 1$ and $y = 2$ and between classes $y = 2$ and $y = 3$ are maximized. Let w, b_1 and b_2 be scaled such that the distance of boundary examples from the hyperplanes is 1. Thus the margin between classes $y = 1$ and $y = 2$ and between classes $y = 2$ and $y = 3$ are $\frac{2}{\|w\|^2}$ (see Figure 12.4). Like conventional SVM (Section 2.4), in order to improve linear separability, the training data are mapped from the original d-dimensional input space using nonlinear function ϕ into a higher D-dimensional feature space. The primal formulation for the *Tri-Class SVM* is written as follows:

$$\min_{w,b_1,b_2,\epsilon,\epsilon^*} \Psi_P = \frac{1}{2}\|w\|^2 + C(\sum_{i=1}^{n_1}\epsilon_i^1 + \sum_{i=1}^{n_2}\epsilon_i^{*2} + \sum_{i=1}^{n_2}\epsilon_i^2 + \sum_{i=1}^{n_3}\epsilon_i^{*3}) \tag{12.7}$$

subject to

$$\begin{aligned}
\langle w, \phi(x_i^1)\rangle - b_1 \leq -1 + \epsilon_i^1, \;\; \epsilon_i^1 \geq 0 \;\; for \;\; i = 1, \ldots, n_1; \\
\langle w, \phi(x_i^2)\rangle - b_1 \geq 1 - \epsilon_i^{*2}, \;\; \epsilon_i^{*2} \geq 0 \;\; for \;\; i = 1, \ldots, n_2; \\
\langle w, \phi(x_i^2)\rangle - b_2 \leq -1 + \epsilon_i^2, \;\; \epsilon_i^2 \geq 0 \;\; for \;\; i = 1, \ldots, n_2; \\
\langle w, \phi(x_i^3)\rangle - b_2 \geq 1 - \epsilon_i^{*3}, \;\; \epsilon_i^{*3} \geq 0 \;\; for \;\; i = 1, \ldots, n_3; b_1 \leq b_2
\end{aligned} \tag{12.8}$$

where x_i^1,x_i^2,x_i^3 represent the training examples that belong to L_h, $L - L_h \cup L_k$ and L_k, respectively such that $n_1 = |L_h|$, $n_2 = |L - \{L_h \cup L_k\}|$ and $n_3 = |L_k|$. The inequality constraint $b_1 \leq b_2$ is added explicitly to make sure that the hyperplanes are correctly ordered. The primal problem can be solved by standard Lagrangian techniques where α_i^1, α_i^{*2}, α_i^2, α_i^{*3}, γ_i^1, γ_i^{*2}, γ_i^2, γ_i^{*3} and η are all non-negative Lagrangian multipliers.

$$\begin{aligned}
L_P(w, b_1, b_2, \epsilon, \epsilon^*) = \frac{1}{2}\|w\|^2 + C(\sum_{i=1}^{n_1}\epsilon_i^1 + \sum_{i=1}^{n_2}\epsilon_i^{*2} + \sum_{i=1}^{n_2}\epsilon_i^2 + \sum_{i=1}^{n_3}\epsilon_i^{*3}) \\
- \sum_{i=1}^{n_1}\alpha_i^1[-1 + \epsilon_i^1 - \langle w, \phi(x_i^1)\rangle + b_1] - \sum_{i=1}^{n_2}\alpha_i^{*2}[\langle w, \phi(x_i^2)\rangle - b_1 - 1 + \epsilon_i^{*2}] \\
- \sum_{i=1}^{n_2}\alpha_i^2[-1 + \epsilon_i^2 - \langle w, \phi(x_i^2)\rangle + b_2] - \sum_{i=1}^{n_3}\alpha_i^{*3}[\langle w, \phi(x_i^3)\rangle - b_2 - 1 + \epsilon_i^{*3}] \\
- \sum_{i=1}^{n_1}\gamma_i^1\epsilon_i^1 - \sum_{i=1}^{n_1}\gamma_i^{*2}\epsilon_i^{*2} - \sum_{i=1}^{n_1}\gamma_i^2\epsilon_i^2 - \sum_{i=1}^{n_1}\gamma_i^{*3}\epsilon_i^{*3} - \eta(b_2 - b_1)
\end{aligned} \tag{12.9}$$

The Karush-Kuhn-Tucker (KKT) optimality conditions for the primal problem are

$$\frac{\partial L_P}{\partial w} = 0 \Rightarrow w = -\sum_{i=1}^{n_1} \alpha_i^1 \phi(x_i^1) + \sum_{i=1}^{n_2} \alpha_i^{*2} \phi(x_i^2) - \sum_{i=1}^{n_2} \alpha_i^2 \phi(x_i^2) + \sum_{i=1}^{n_3} \alpha_i^{*3} \phi(x_i^3); \tag{12.10}$$

$$\frac{\partial L_P}{\partial b_1} = 0 \Rightarrow \sum_{i=1}^{n_1} \alpha_i^1 = \sum_{i=1}^{n_2} \alpha_i^{*2} + \eta; \tag{12.11}$$

$$\frac{\partial L_P}{\partial b_2} = 0 \Rightarrow \sum_{i=1}^{n_2} \alpha_i^2 + \eta = \sum_{i=1}^{n_3} \alpha_i^{*3}; \tag{12.12}$$

$$\frac{\partial L_P}{\partial \epsilon_i^1} = 0 \Rightarrow C - \alpha_i^1 = \gamma_i^1 \ \Rightarrow \alpha_i^1 \leq C \ for \ \ i = 1, \ldots, n_1; \tag{12.13}$$

$$\frac{\partial L_P}{\partial \epsilon_i^{*2}} = 0 \Rightarrow C - \alpha_i^{*2} = \gamma_i^{*2} \ \Rightarrow \alpha_i^{*2} \leq C \ for \ \ i = 1, \ldots, n_2; \tag{12.14}$$

$$\frac{\partial L_P}{\partial \epsilon_i^2} = 0 \Rightarrow C - \alpha_i^2 = \gamma_i^2 \ \Rightarrow \alpha_i^2 \leq C \ for \ \ i = 1, \ldots, n_2; \tag{12.15}$$

$$\frac{\partial L_P}{\partial \epsilon_i^{*3}} = 0 \Rightarrow C - \alpha_i^{*3} = \gamma_i^{*3} \ \Rightarrow \alpha_i^{*3} \leq C \ for \ \ i = 1, \ldots, n_3 \tag{12.16}$$

12.3.2.2 Dual problem

The unknwon variables of the convex optimization problem given by Eq. (12.7) and Eq. (12.8) are w, b_1 and b_2. Thus the number of variables is the number of input variables plus 2: D+2. When the number of input variables is small, one can solve Eq. (12.7) and Eq. (12.8) by the quadratic programming technique. But as one maps the input space into a high-dimensional feature space, in some cases, with infinite dimensions, one converts Eq. (12.7) and Eq. (12.8) into the equivalent dual problem whose number of variables depends on the number of training examples. In order to represent the dual problem in a compact form, some notations are introduced first. Let X^j be the $d \times n_j$ matrix whose columns are the training examples that belong to class j (x_i^j), where $i = 1, \ldots, n_j$ and $j = 1, 2, 3$:

$$X^j = [x_1^j, \ldots, x_{n_j}^j]_{d \times n_j}$$

Let $\mu = (\alpha^1, \alpha^{*2}, \alpha^2, \alpha^{*3})^T$ be the vector holding all the Lagrange multipliers where $\alpha^1 = (\alpha_1^1, \ldots, \alpha_{n_1}^1)^T, \alpha^{*2} = (\alpha_1^{*2}, \ldots, \alpha_{n_2}^{*2})^T, \alpha^2 = (\alpha_1^2, \ldots, \alpha_{n_2}^2)^T, \alpha^{*3} = (\alpha_1^{*3}, \ldots, \alpha_{n_3}^{*3})^T$. Let Q be the training data matrix:

$$Q = [X^1, X^2, X^2, X^3]_{d \times N}, \tag{12.17}$$

and s be the N-dimensional vector defined as follows:

$$s_i = \begin{cases} -1 & \text{if } 1 \leq i \leq n_1 \text{ or } n_1 + n_2 + 1 \leq i \leq n_1 + 2.n_2 \ , \\ 1 & \text{if } n_1 + 1 \leq i \leq n_1 + n_2 \text{ or } n_1 + 2.n_2 + 1 \leq i \leq N. \end{cases} \tag{12.18}$$

Then the kernel matrix $H = [s_i s_j \mathbb{K}(x_i, x_j)]_{N\times N}$ results from applying the kernel function on Q directly in the input space rather than the inner-products in the higher dimensional feature space ($\mathbb{K}(x_i, x_j) = \langle \phi(x_i), \phi(x_j) \rangle$ known as *kernel trick*), where $N = n_1 + 2.n_2 + n_3$. In case of linear kernel, $H = [s_i s_j \langle x_i, x_j \rangle]_{N\times N}$. Using the new notation, Eq. (12.10) becomes $w = Q\mu$. Let us now apply Wolfe duality theory to the primal problem. By substituting with the new expression of w and the other KKT conditions in Eq. (12.11) to Eq. (12.16) into the Lagrangian in Eq. (12.9), one obtains the dual objective function Ψ_D that should be maximized with respect to the Lagrange multipliers α,α^* and η alone while all the remaining multipliers γ and γ^* have been dropped out:

$$\max_{\alpha,\alpha^*,\eta} \Psi_D = \max_{\alpha,\alpha^*,\eta} \sum_{i=1}^{N} \mu_i - \frac{1}{2}\mu^T H \mu \tag{12.19}$$

Subject to the constraints

$$\begin{aligned} \sum_{i=1}^{n_1} \alpha_i^1 &= \sum_{i=1}^{n_2} \alpha_i^{*2} + \eta; \\ \sum_{i=1}^{n_2} \alpha_i^2 + \eta &= \sum_{i=1}^{n_3} \alpha_i^{*3}; \\ 0 \le \alpha_i^1 \le C \;\; for \;\; i &= 1, \ldots, n_1; \\ 0 \le \alpha_i^{*2} \le C \;\; for \;\; i &= 1, \ldots, n_2; \\ 0 \le \alpha_i^2 \le C \;\; for \;\; i &= 1, \ldots, n_2; \\ 0 \le \alpha_i^{*3} \le C \;\; for \;\; i &= 1, \ldots, n_3; \eta \ge 0 \end{aligned} \tag{12.20}$$

Note that the size of the optimization problem (number of unknown variables) is N. The parallel hyperplanes direction vector w can be obtained from Eq. (12.10) once the optimal values of α and α^* are obtained from solving Eq. (12.19) and Eq. (12.20). The calculation of the hyperplanes positions b_1 and b_2 is addressed in the following section. Then the decision function for a given example x is

$$f_{kh}(x) = -\sum_{i=1}^{n_1} \alpha_i^1 \mathbb{K}(x_i^1, x) + \sum_{i=1}^{n_2} (\alpha_i^{*2} - \alpha_i^2) \mathbb{K}(x_i^2, x) + \sum_{i=1}^{n_3} \alpha_i^{*3} \mathbb{K}(x_i^3, x) \tag{12.21}$$

Therefore, an unknown example x is classified as follows (see Figure 12.4):

$$\begin{cases} \text{Class 1 (assigned to } \omega_h) & \text{if } f_{kh}(x) \le b_1, \\ \text{Class 2 (unclassified and rejected)} & \text{if } b_1 < f_{kh}(x) \le b_2. \\ \text{Class 3 (assigned to } \omega_k) & otherwise. \end{cases} \tag{12.22}$$

12.3.3 SMO for Tri-Class SVM

Angulo et al. [16] have solved the optimization problem using the exact quadratic program-solver provided by Matlab Optimization Toolbox. Unfortunately, traditional quadratic programming algorithms are not suitable to solve this large size problem because they require the large kernel matrix H be computed and stored in memory. Platt [147] introduced a fast learning algorithm for SVM classifiers design, called *Sequential Minimal Optimization* (SMO). Keerthi et al. [94] proposed an improved version of Platt's SMO algorithm then Chu and Keerthi [42] extended this improvement to ordinal regression. In this section, Chu and Keerthi's SMO algorithm is extended for the design of *Tri-Class SVM* classifiers, that is, to solve the dual problem defined in Eq. (12.19) and Eq. (12.20). The main idea of SMO is to jointly optimize only a pair of selected Lagrange multipliers at each step while keeping the other multipliers fixed. The main advantage of SMO that the numerical QP optimization is avoided entirely because two Lagrange multipliers can be solved analytically.

It is important to write down the Lagrangian of the dual problem where $\beta_1 \in \mathbb{R}$, $\beta_2 \in \mathbb{R}$, $\delta_i^1 \geq 0$, $\delta_i^{*2} \geq 0$, $\delta_i^2 \geq 0$, $\delta_i^{*3} \geq 0$, $\mu_i^1 \geq 0$, $\mu_i^{*2} \geq 0$, $\mu_i^2 \geq 0$, $\mu_i^{*3} \geq 0$ and $\lambda \geq 0$ are the Lagrangian multipliers.

$$\begin{aligned} L_D(\alpha, \alpha^*, \mu) = \Psi_D &+ \beta_1(\sum_{i=1}^{n_1} \alpha_i^1 - \sum_{i=1}^{n_2} \alpha_i^{*2} - \eta) + \beta_2(\sum_{i=1}^{n_2} \alpha_i^2 + \eta - \sum_{i=1}^{n_3} \alpha_i^{*3}) \\ &- \sum_{i=1}^{n_1} \delta_i^1 \alpha_i^1 - \sum_{i=1}^{n_2} \delta_i^{*2} \alpha_i^{*2} - \sum_{i=1}^{n_2} \delta_i^2 \alpha_i^2 - \sum_{i=1}^{n_3} \delta_i^{*3} \alpha_i^{*3} - \sum_{i=1}^{n_1} \mu_i^1 (C - \alpha_i^1) \\ &- \sum_{i=1}^{n_2} \mu_i^{*2} (C - \alpha_i^{*2}) - \sum_{i=1}^{n_2} \mu_i^2 (C - \alpha_i^2) - \sum_{i=1}^{n_3} \mu_i^{*3} (C - \alpha_i^{*3}) - \lambda\eta \end{aligned} \quad (12.23)$$

For the optimal solution, the following KKT conditions are satisfied for the dual problem.

$$\frac{\partial L_D}{\partial \alpha_i^1} = 0 \Rightarrow -f_{kh}(x_i^1) - 1 - \delta_i^1 + \mu_i^1 + \beta_1 = 0 \quad for \quad i = 1, \ldots, n_1; \quad (12.24)$$

$$\frac{\partial L_D}{\partial \alpha_i^{*2}} = 0 \Rightarrow f_{kh}(x_i^2) - 1 - \delta_i^{*2} + \mu_i^{*2} - \beta_1 = 0 \quad for \quad i = 1, \ldots, n_2; \quad (12.25)$$

$$\frac{\partial L_D}{\partial \alpha_i^2} = 0 \Rightarrow -f_{kh}(x_i^2) - 1 - \delta_i^2 + \mu_i^2 + \beta_2 = 0 \quad for \quad i = 1, \ldots, n_2; \quad (12.26)$$

$$\frac{\partial L_D}{\partial \alpha_i^{*3}} = 0 \Rightarrow f_{kh}(x_i^3) - 1 - \delta_i^{*3} + \mu_i^{*3} - \beta_2 = 0 \quad for \quad i = 1, \ldots, n_3; \quad (12.27)$$

$$\frac{\partial L_D}{\partial \eta} = 0 \Rightarrow \beta_2 - \beta_1 = \lambda \Rightarrow \beta_1 \leq \beta_2; \quad (12.28)$$

in addition to the following *KKT complementarity conditions*.

$$\delta_i^1\alpha_i^1 = 0,\ \ \mu_i^1(C-\alpha_i^1) = 0, \delta_i^1 \geq 0, \mu_i^1 \geq 0\ \ for\ \ i = 1,\ldots,n_1; \tag{12.29}$$
$$\delta_i^{*2}\alpha_i^{*2} = 0,\ \ \mu_i^{*2}(C-\alpha_i^{*2}) = 0, \delta_i^{*2} \geq 0, \mu_i^{*2} \geq 0\ \ for\ \ i = 1,\ldots,n_2; \tag{12.30}$$
$$\delta_i^2\alpha_i^2 = 0,\ \ \mu_i^2(C-\alpha_i^2) = 0, \delta_i^2 \geq 0, \mu_i^2 \geq 0\ \ for\ \ i = 1,\ldots,n_2; \tag{12.31}$$
$$\delta_i^{*3}\alpha_i^{*3} = 0,\ \ \mu_i^{*3}(C-\alpha_i^{*3}) = 0, \delta_i^{*3} \geq 0, \mu_i^{*3} \geq 0\ \ for\ \ i = 1,\ldots,n_3; \tag{12.32}$$
$$\lambda\eta = 0 \tag{12.33}$$

These conditions can be simplified by considering the following 6 cases for j=1,2.
case 1: $\alpha_i^j = 0$

$$\mu_i^j = 0, \delta_i^j \geq 0 \Rightarrow f_{kh}(x_i^j) + 1 \leq \beta_j \tag{12.34}$$

case 2: $0 < \alpha_i^j < C$

$$\mu_i^j = 0, \delta_i^j = 0 \Rightarrow f_{kh}(x_i^j) + 1 = \beta_j \tag{12.35}$$

case 3: $\alpha_i^j = C$

$$\mu_i^j \geq 0, \delta_i^j = 0 \Rightarrow f_{kh}(x_i^j) + 1 \geq \beta_j \tag{12.36}$$

case 4: $\alpha_i^{*j+1} = 0$

$$\mu_i^{*j+1} = 0, \delta_i^{*j+1} \geq 0 \Rightarrow f_{kh}(x_i^{j+1}) - 1 \geq \beta_j \tag{12.37}$$

case 5: $0 < \alpha_i^{*j+1} < C$

$$\mu_i^{*j+1} = 0, \delta_i^{*j+1} = 0 \Rightarrow f_{kh}(x_i^{j+1}) - 1 = \beta_j \tag{12.38}$$

case 6: $\alpha_i^{*j+1} = C$

$$\mu_i^{*j+1} \geq 0, \delta_i^{*j+1} = 0 \Rightarrow f_{kh}(x_i^{j+1}) - 1 \leq \beta_j \tag{12.39}$$

According to the cases in Eq. (12.34) to Eq. (12.39), we define the following index sets for j=1,2:

$$I_{0a}^j = \{i \in \{1,\ldots,n_j\} : 0 < \alpha_i^j < C\};$$
$$I_{0b}^j = \{i \in \{1,\ldots,n_{j+1}\} : 0 < \alpha_i^{*j+1} < C\}; I_0^j = I_{0a}^j \cup I_{0b}^j;$$
$$I_1^j = \{i \in \{1,\ldots,n_{j+1}\} : \alpha_i^{*j+1} = 0\}; I_2^j = \{i \in \{1,\ldots,n_j\} : \alpha_i^1 = C\};$$
$$I_3^j = \{i \in \{1,\ldots,n_{j+1}\} : \alpha_i^{*j+1} = C\}; I_4^j = \{i \in \{1,\ldots,n_j\} : \alpha_i^j = 0\}$$

12.3.3.1 Computing the Thresholds

Let us define F_i as

$$F_i = \begin{cases} f_{kh}(x_i^j) + 1 & \text{if } i \in I_{0a}^j \cup I_2^j \cup I_4^j, \\ f_{kh}(x_i^{j+1}) - 1 & \text{if } i \in I_{0b}^j \cup I_1^j \cup I_3^j. \end{cases}$$

The necessary conditions in Eq. (12.34) to Eq. (12.39) can be summarized as

$$\beta_j \leq F_i \ \ \forall i \in I_0^j \cup I_1^j \cup I_2^j; \tag{12.40}$$

$$\beta_j \geq F_i \ \ \forall i \in I_0^j \cup I_3^j \cup I_4^j; \tag{12.41}$$

which can be written as:

$$b_{low}^j \leq b_{up}^j \ \text{ for } \ j = 1, 2 \tag{12.42}$$

where

$$\begin{aligned} &b_{low}^j = max\{F_i : i \in I_0^j \cup I_3^j \cup I_4^j\}; \ \ b_{up}^j = min\{F_i : i \in I_0^j \cup I_1^j \cup I_2^j\}; \\ &i_{low}^j = argmax\{F_i : i \in I_0^j \cup I_3^j \cup I_4^j\} \ \text{ and } \ i_{up}^j = argmin\{F_i : i \in I_0^j \cup I_1^j \cup I_2^j\} \end{aligned} \tag{12.43}$$

In order to merge conditions in Eq. (12.28) and Eq. (12.42), we define

$$\begin{aligned} B_{up}^1 = min\{b_{up}^1, b_{up}^2\} \ \text{ and } \ B_{low}^1 = \begin{cases} max\{b_{low}^1, b_{low}^2\} & \text{if } \beta_1 = \beta_2, \\ b_{low}^1 & \text{otherwise.} \end{cases} \\ B_{up}^2 = \begin{cases} min\{b_{up}^1, b_{up}^2\} & \text{if } \beta_1 = \beta_2, \\ b_{up}^2 & \text{otherwise.} \end{cases} \ \text{ and } \ B_{low}^2 = max\{b_{low}^1, b_{low}^2\} \end{aligned} \tag{12.44}$$

It is usually not possible to achieve exact optimality, approximate optimality conditions are defined.

$$B_{low}^1 \leq B_{up}^1 + \tau \ \text{ and } \ B_{low}^2 \leq B_{up}^2 + \tau \tag{12.45}$$

where τ is a positive tolerance parameter. Note that at the optimal solution, β_j and b_j are identical. Thus b_j can be placed at the middle between B_{low}^j and B_{up}^j. The following points will help to easy understand the SMO algorithm presented in Algorithm 21:

1. First check optimality for each training example i as in Algorithm 22 and Algorithm 23. Note that if $y_i < 3$, we check the y_i^{th} threshold (its upper threshold) and if $y_i > 1$, we check the $(y_i - 1)^{th}$ threshold (its lower threshold). That is if $i \in I_1^J \cup I_2^J$ and $F_i < B_{low}^J - \tau$ then there is a violation, and in this case SMO's *takeStep* method in Algorithm 24 is applied to the pair (i, i_{low}^J). Similarly, if $i \in I_3^J \cup I_4^J$ and $F_i > B_{up}^J + \tau$ then there is a violation, and *takeStep* is applied to the pair (i_{up}^J, i) where $J = y_i - 1$ or y_i.

2. After checking for optimality of all indices, get the index of the most *violating threshold* that is $J = arg\min\{\tau, B_{up}^1 - B_{low}^1, B_{up}^2 - B_{low}^2\}$. If $J = 0$, that means that both 1^{st} and 2^{nd} thresholds satisfy optimality condition in Eq. (12.45). Otherwise, *takeStep* is applied to the most violating pair (i_{up}^J, i_{low}^J). This step is repeated until $J = 0$.

3. An additional loop is applied to check optimality on all training examples ($examineAll = TRUE$). Since (i^j_{low}, b^j_{low}) and (i^j_{up}, b^j_{up}) have been partially computed in *takeStep* using only I^j_0, at this loop, these quantities are modified by each example i even if there is no violation if $i \in I^J_1 \cup I^J_2$ and $F_i < b^J_{up}$ or if $i \in I^J_3 \cup I^J_4$ and $F_i > b^J_{low}$ as shown in Algorithm 22.

4. After checking for optimality of all indices, if there is a violation for any index i(that is, $numChanged > 0$), go to Step 2. Otherwise, we conclude that all Lagrange multipliers α and α^* satisfy optimality and that the correct values of b^j_{up} and b^j_{low} have been computed.

Algorithm 21 The pseudo code of SMO for *Tri-Class SVM* ($TriClassSMO$)

Require: $L = \{(x_i, y_i) : x_i \in \mathbb{R}^d$ and $y_i \in \{1, 2, 3\}\}$ set of training examples
initialize *alpha* array to zero
for each training example $x_i \in L$ **do**
  calculate $f_{kh}(x_i)$ (as defined in Eq. (12.21)) and set $f\_cache[i] = f_{kh}(x_i)$
  if $y_i < 3$ **then**
    add i into $I^{y_i}_4$
  end if
  if $y_i > 1$ **then**
    add i into $I^{y_i-1}_1$
  end if
end for
set (i^j_{low}, B^j_{low}) and (i^j_{up}, B^j_{up}) for $j = 1, 2$ as defined in Eq. (12.43) and Eq. (12.44)
set examineAll = TRUE and numChanged = 0
while $numChanged > 0$ or $examineAll$ **do**
  if examineAll **then**
    numChanged = 0
    for each training example $x_i \in L$ **do**
      numChanged += $examineExample(i)$
    end for
  else
    while $J = activeThreshold() > 0$ and $numChanged > 0$ **do**
      if $takeStep(i^J_{up}, i^J_{low})$ **then**
        numChanged += 1
      end if
    end while
    numChanged = 0 ;
  end if
  if examineAll = TRUE **then**
    examineAll = FALSE
  else if numChanged =0 **then**
    examineAll = TRUE
  end if
end while
return α,α^*, b_1 and b_2

12.3.3.2 Solving for Two Lagrange Multipliers (*takeStep*)

In order to jointly optimize the two Lagrange multipliers α_u and α_o, SMO first computes the bound constraints and the linear equality constraint on these mul-

Algorithm 22 examineExample(i_2)

if $i_2 \in I_0^1 \cup I_0^2$ **then**
  set $f_2 = f\_cache[i_2]$
else
  calculate $f_{kh}(x_2)$ (as defined in Eq. (12.21)) and set $f\_cache[i2] = f_2 = f_{kh}(x_2)$ {Update i_{low}^j, i_{up}^j, b_{low}^j and b_{up}^j for $j = 1, 2$}
  if $y_i < 3$ **then**
    if $i_2 \in I_{0a}^{y_i} \cup I_2^{y_i}$ and $f_2 + 1 < b_{up}^{y_i}$ **then**
      set $b_{up}^{y_i} = f_2 + 1$ and $i_{up}^{y_i} = i_2$
    end if
    if $i_2 \in I_{0a}^{y_i} \cup I_4^{y_i}$ and $f_2 + 1 > b_{low}^{y_i}$ **then**
      set $b_{low}^{y_i} = f_2 + 1$ and $i_{low}^{y_i} = i_2$
    end if
  end if
  if $y_i > 1$ **then**
    if $i_2 \in I_{0b}^{y_i-1} \cup I_1^{y_i-1}$ and $f_2 - 1 < b_{up}^{y_i-1}$ **then**
      set $b_{up}^{y_i-1} = f_2 - 1$ and $i_{up}^{y_i-1} = i_2$
    end if
    if $i_2 \in I_{0b}^{y_i-1} \cup I_3^{y_i-1}$ and $f_2 - 1 > b_{low}^{y_i-1}$ **then**
      set $b_{up}^{y_i-1} = f_2 - 1$ and $i_{low}^{y_i-1} = i_2$
    end if
  end if
end if
update B_{low}^1, B_{up}^1, B_{low}^2, and B_{up}^2
[*optimal*, i_u, i_o] = *checkOptimality*(B_{low}^1, B_{up}^1, B_{low}^2, B_{up}^2)
if !*optimal* and *takestep*(i_u, i_o) **then** return 1 **else** return 0 **end**

tipliers as follows:

$$0 \leq \alpha_u \leq C \text{ and } 0 \leq \alpha_o \leq C; \tag{12.46}$$

$$\alpha_u + s_o s_u \alpha_o = \alpha_u^{new} + s_o s_u \alpha_o^{new} = \rho \tag{12.47}$$

where

$$s_u = \begin{cases} -1 & \text{if } i_u \in I_{0a}^{th} \cup I_2^{th}, \\ +1 & \text{if } i_u \in I_{0b}^{th} \cup I_1^{th} \end{cases}; \tag{12.48}$$

$$s_o = \begin{cases} -1 & \text{if } i_o \in I_{0a}^{th} \cup I_4^{th}, \\ +1 & \text{if } i_o \in I_{0b}^{th} \cup I_3^{th} \end{cases} \tag{12.49}$$

Then the sub-optimization problem is solved for α_u and α_o. For simplicity, all quantities that refer to the first multiplier will have a subscript u, while all quantities that refer to the second multiplier will have a subscript o. Because there are only two multipliers, the constraints can be displayed in two dimensions (see Figure 12.6). The bound inequality constraints in Eq. (12.46) enforce the Lagrange multipliers to lie within a box, while the linear equality constraint in Eq. (12.47) enforces them to lie on a diagonal line. Thus, the constrained minimum of the objective function must lie on a diagonal line segment (as shown in Figure 12.6). This constraint explains why two is the minimum number of Lagrange multipliers that can be optimized at every step: if SMO optimized only one multiplier,

Algorithm 23 checkOptimality(i_2)

optimal = TRUE
if $y_i < 3$ **then**
  if $i_2 \in I_{0a}^{y_i} \cup I_2^{y_i}$ and $f_2 + 1 < B_{low}^{y_i} - \tau$ **then**
    set *optimal* = FALSE, $i_u = i_2$ and $i_o = i_{low}^{y_i}$
  end if
  if $i_2 \in I_{0a}^{y_i} \cup I_4^{y_i}$ and $f_2 + 1 > B_{up}^{y_i} + \tau$ **then**
    set *optimal* = FALSE, $i_u = i_{up}^{y_i}$ and $i_o = i_2$
  end if
  if *optimal*= FALSE and $i_2 \in I_{0a}^{y_i}$ **then**
    if $B_{low}^{y_i} - (f_2 + 1) > (f_2 + 1) - B_{up}^{y_i}$ **then**
      set $i_u = i_2$ and $i_o = i_{low}^{y_i}$
    else
      set $i_u = i_{up}^{y_i}$ and $i_o = i_2$
    end if
  end if
end if
if *optimal* and $y_i > 1$ **then**
  if $i_2 \in I_{0b}^{y_i-1} \cup I_1^{y_i-1}$ and $f_2 - 1 < B_{low}^{y_i-1} - \tau$ **then**
    set *optimal* = FALSE, $i_u = i_2$ and $i_o = i_{low}^{y_i-1}$
  end if
  if $i_2 \in I_{0b}^{y_i-1} \cup I_3^{y_i-1}$ and $f_2 - 1 > B_{up}^{y_i-1} + \tau$ **then**
    set *optimal* = FALSE, $i_u = i_{up}^{y_i-1}$ and $i_o = i_2$
  end if
  if *optimal* = FALSE and $i_2 \in I_{0b}^{y_i}$ **then**
    if $B_{low}^{y_i-1} - (f_2 - 1) > (f_2 - 1) - B_{up}^{y_i-1}$ **then**
      set $i_u = i_2$ and $i_o = i_{low}^{y_i-1}$
    else
      set $i_u = i_{up}^{y_i-1}$ and $i_o = i_2$
    end if
  end if
end if
return *optimal*, i_u, i_o

it could not fulfill the linear equality constraint. The ends of the diagonal line segment can be expressed quite simply. Without loss of generality, the algorithm first computes the second Lagrange multiplier α_o and computes the ends of the diagonal line segment in terms of α_o. If s_u does not equal s_o, then the following bounds apply to α_o:

$$L = \max(0, \alpha_o - \alpha_u), H = \min(C, C + \alpha_o - \alpha_u). \tag{12.50}$$

If s_u equals s_o, then the following bounds apply to α_o:

$$L = \max(0, \alpha_o + \alpha_u - C), H = \min(C, \alpha_o + \alpha_u) \tag{12.51}$$

Due to the linear constraint between α_o and α_u, the unbounded solution to the restricted problem can be exactly determined as

$$\alpha_o^{new} = \alpha_o + s_o \Delta\eta \tag{12.52}$$

$$\text{where} \Delta\eta = \frac{(f_{kh}(x_u) - s_u - f_{kh}(x_o) + s_o)}{\mathbb{K}(x_u, x_u) + \mathbb{K}(x_o, x_o) - 2\mathbb{K}(x_u, x_o)} \tag{12.53}$$

Algorithm 24 takeStep(i_u, i_o)

if $i_u = i_o$ **then** return 0 **end**
compute L and H using Eq. (12.50) and Eq. (12.51)
if $L = H$ **then** return 0 **end**
set $k_{11} = \mathbb{K}(x_u, x_u)$, $k_{22} = \mathbb{K}(x_o, x_o)$ and $k_{12} = \mathbb{K}(x_u, x_o)$
set $\gamma = k_{11} + k_{22} - 2k_{12}$;
if $\gamma < 0$ **then**
    return 0
else
    set $\Delta\eta = (f\_cache[i_u] - s_u - f\_cache[i_o] + s_o)/\gamma$
    set $\alpha_o^{new} = \alpha_o + s_o \Delta\eta$
    if $\alpha_o^{new} < L$ **then** $\alpha_o^{new} = L$
    else if $\alpha_o^{new} > H$ **then** $\alpha_o^{new} = H$ **end**
    if $|\alpha_o^{new} - \alpha_o| < eps(\alpha_o^{new} + \alpha_o + eps)$ **then** return 0 **end**
    set $\alpha_u^{new} = \alpha_u + s_u s_o(\alpha_o - \alpha_o^{new})$
    set $\eta = \eta - \Delta\eta$
    update $f\_cache[i]$ for $i \in I_0^1 \cup I_0^2$ using α_o^{new} and α_u^{new}
    update the index sets using α_o^{new} and α_u^{new}
    update f_cache of i_u and i_o
    $f\_cache[i_u] += s_u(\alpha_u^{new} - \alpha_u)k_{11} + s_o(\alpha_o^{new} - \alpha_o)k_{12}$;
    $f\_cache[i_o] += s_u(\alpha_u^{new} - \alpha_u)k_{12} + s_o(\alpha_o^{new} - \alpha_o)k_{22}$
    partially update (i_{low}^j, b_{low}^j) and (i_{up}^j, b_{up}^j) for $j = 1, 2$ as defined in Eq. (12.43) and using only i_u, i_o and indices in $I_0^1 \cup I_0^2$
    return 1
end if

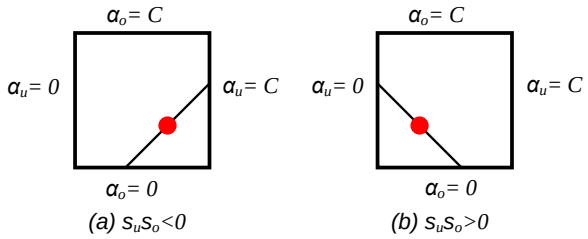


Figure 12.6: An illustration of the bound inequality constraints and linear equality constraints for α_u and α_o

As a next step, we check whether α_o^{new} satisfies the box constraint and if not, we clip it to the ends of the line segment as shown in Algorithm 24. Using the final value of α_o^{new}, α_u can be updated.

$$\alpha_u^{new} = \alpha_u + s_u s_o(\alpha_o - \alpha_o^{new}) \tag{12.54}$$

12.3.4 Probabilistic Output for *Tri-Class SVM*

In classification phase, *Tri-Class SVM* produces for an input example x an uncalibrated output $f_{kh}(x)$ that is not a probability as shown in Eq. (12.21). This output represents the distance (unbiased margin) in the kernel space between $\phi(x)$ and the separating hyperplane. Although a crisp class label can be directly predicted from the SVM output as done in Eq. (12.22), a probabilistic outputs

Table 12.2: One-against-One Decision Profile of example x

	ω_1	ω_2	ω_3	ω_4
ω_1	-	$P_{12}(y=3\|x)$	$P_{13}(y=3\|x)$	$P_{14}(y=3\|x)$
ω_2	$P_{12}(y=1\|x)$	-	$P_{23}(y=3\|x)$	$P_{24}(y=3\|x)$
ω_3	$P_{13}(y=1\|x)$	$P_{23}(y=1\|x)$	-	$P_{34}(y=3\|x)$
ω_4	$P_{14}(y=1\|x)$	$P_{24}(y=1\|x)$	$P_{34}(y=1\|x)$	-

are required in many cases. For example, if this machine is a member of an ensemble and the individual outputs must be combined to provide the final decision of the ensemble. Another important utilization is the confidence-based semi-supervised, preference and active learning algorithms that depend on the class probability estimates (CPE) to measure confidence such as *Co-Training*. We derive a probabilistic interpretation for the *Tri-Class SVM* output f_{kh} inspired by the method in [196] that fits a sigmoid function on the SVM output where Eq. (12.56) represents the doubt that input example x belongs to ω_k or ω_h.

$$P_{kh}(y=1|x) = \left(1 - \frac{1}{1+exp(-(f_{kh}(x)-b_1))}\right); \tag{12.55}$$

$$P_{kh}(y=2|x) = \left(\frac{1}{1+exp(-(f_{kh}(x)-b_1))}\right)\left(1 - \frac{1}{1+exp(-(f_{kh}(x)-b_2))}\right); \tag{12.56}$$

$$P_{kh}(y=3|x) = \left(\frac{1}{1+exp(-(f_{kh}(x)-b_1))}\right)\left(\frac{1}{1+exp(-(f_{kh}(x)-b_2))}\right) \tag{12.57}$$

12.3.5 Decision Fusion for Ensemble of Probabilistic Tri-Class SVMs

Crisp *Tri-Class SVM*s are combined by a majority voting scheme to produce the final multi-class decision for a given example x where only positive outputs are considered in the voting scheme, so ties between classes are considered as errors. Decision Profile of example x in Table 12.2 stores the probabilistic outputs of $\frac{K(K-1)}{2}$ *Tri-Class SVM*s, derived in Eq. (12.55) to Eq. (12.57). Thus the final probabilistic output of One-against-One ensemble of *Tri-Class SVM*s is defined as follows, for each $k = 1, \ldots, K$:

$$P(y=\omega_k|x) = \frac{\sum_{h=1}^{k-1} P_{hk}(y=1|x) + \sum_{h=k+1}^{K} P_{kh}(y=3|x)}{\sum_{k'=1}^{K}\sum_{h=1}^{k'-1} P_{hk'}(y=1|x) + \sum_{h=k'+1}^{K} P_{k'h}(y=3|x)} \tag{12.58}$$

Note that in case of using *Co-Training* as shown in Section 12.2, P_{hk} is not the output of a single *Tri-Class SVM* but it represents the output of an ensemble of SVMs as defined in Eq. (12.1).

12.4 Facial Expressions Recognition

The Cohn-Kanade dataset is a collection of image sequences with emotional content [93], which is available for research purposes. It was described in Section 7.1.5 (see Figure 7.7).

12.4.1 Feature Extraction

In all automatic facial expression recognition systems first some relevant features are extracted from the facial image and these feature vectors then utilized to train some type of classifier to recognize the facial expression. The details of the feature extraction step are given in Section 7.1.5.

12.4.2 GMM Supervectors

The feature vectors extracted from a video (image sequence) should be passed to classifiers in order to recognize the different emotions. Usually different videos have different durations leading to feature vectors of different lengths. Since almost all classifiers require fixed-length feature vectors as input, the stream of feature vectors must be transformed into a fixed-length input vector. This is the idea of GMM supervectors [37] (see Figure 12.7) which are calculated as follows:

1. Collect a general large sequence database for the universal background model (UBM). In this study the available data is divided into training and testing sets according to 8-fold cross validation test and the database was built by gathering all the sequences from the current training data set.

2. Calculate the UBM. This UBM is a GMM calculated using the general database [154]. The UBM acts as a basic model that is independent of any emotion or individual. It is used in order to guarantee uniform representation for all possible samples. Basically, this data base should be a large set of samples covering different person, in the state of different emotions.

3. For each image sequence, the pre-trained UBM is adapted in order to represent the information carried by the current sequence. Adaptation is performed through the maximum-a-posteriori (MAP) algorithm.

4. Construct the supervector by concatenating the means of the Gaussian mixture components of the adapted model according:

$$\boldsymbol{\mu} = [\boldsymbol{\mu}_1^T \dots \boldsymbol{\mu}_M^T]. \tag{12.59}$$

   These vectors $\boldsymbol{\mu}$ - the GMM supervectors - are then the input vectors to the *Tri-Class SVM* classifiers.

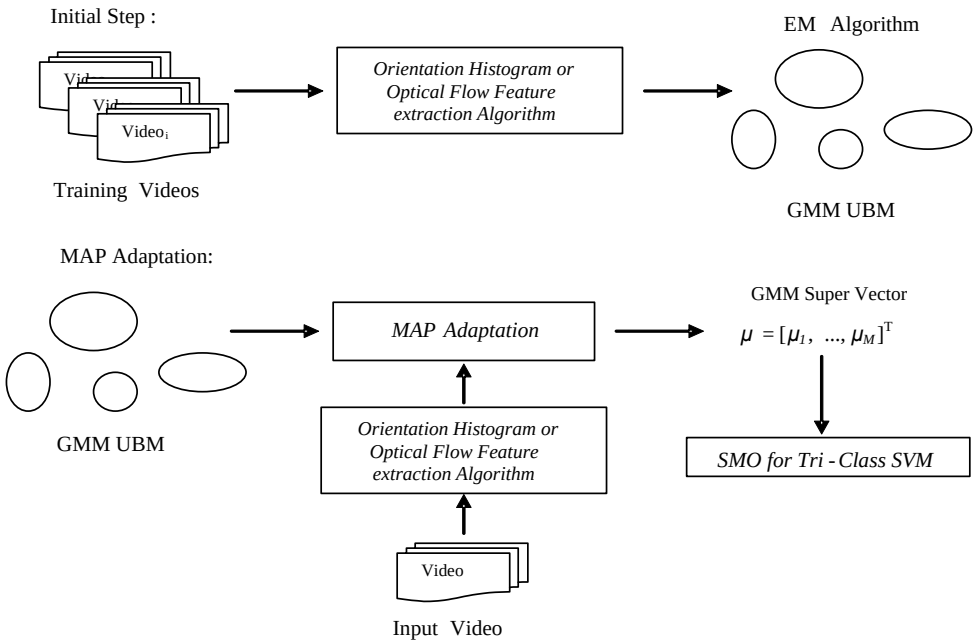


Figure 12.7: Calculation of GMM Super Vectors that is performed for each feature type

MAP adaptation is a popular technique for adapting the UBM [154]. The basic steps of MAP are as follows:

1. Calculate the posterior probability of each component of the model given the current sequence. This corresponds to the probability that this component has contributed to generating the sequence:

$$n_m = \sum_{t=1}^{T} P(m|\mathbf{x}_t). \tag{12.60}$$

Where m is a Gaussian component and $\mathbf{x}_t$ is an image of the sequence, that is currently used to adapt the UBM, assuming that the sequence is divided into T images. The factor $P(m|\mathbf{x}_t)$ is calculated through

$$P(m|\mathbf{x}_t) = \frac{P(m)p(\mathbf{x}_t|m)}{\sum_{k=1}^{M} P(k)p(\mathbf{x}_t|k)}. \tag{12.61}$$

here $p(\mathbf{x}_t|m)$ is the value of the Gaussian function representing component m at point $\mathbf{x}_t$ defined in Eq. (12.70).

2. Calculate the new estimate of the mean and covariance parameters, accord-

ing to the equations

$$\mathbf{E}_m \quad (\mathbf{x}) = \frac{1}{n_m} \sum_{t=1}^{T} P(m|\mathbf{x}_t)\mathbf{x}_t \tag{12.62}$$

$$\mathbf{E}_m \quad (\mathbf{x}\mathbf{x}^T) = \frac{1}{n_m} \sum_{t=1}^{T} P(m|\mathbf{x}_t)\mathbf{x}_t\mathbf{x}_t^T. \tag{12.63}$$

3. Adapt the UBM parameters to represent the features of the current sequence using the equations

$$\hat{P}(m) = [\alpha_m n_m/T + (1-\alpha_m)P(m)]\gamma \tag{12.64}$$

$$\hat{\boldsymbol{\mu}}_m = \alpha_m \mathbf{E}_m(\mathbf{x}) + (1-\alpha_m)\boldsymbol{\mu}_m \tag{12.65}$$

$$\hat{\boldsymbol{\Sigma}}_m = \alpha_m \mathbf{E}_m(\mathbf{x}^2) + (1-\alpha_m)(\boldsymbol{\Sigma}_m + \boldsymbol{\mu}_m^2) - \boldsymbol{\mu}_m^2 \tag{12.66}$$

Where $\hat{P}(m)$, $\hat{\boldsymbol{\mu}}_m$ and $\hat{\boldsymbol{\Sigma}}_m$ are the adapted weight, mean and covariance matrix of component m, respectively.

The factor γ in Eq. (12.64) ensures that the weights sum up to one, so that the constraints in Eq. (12.69) are satisfied. The adaptation coefficient α_m is a factor balancing the adapted values between old estimates (from the UBM) and calculated parameters (by using MAP adaptation). The adaptation coefficient is calculated by the following equation using a relevance factor r

$$\alpha_m = \frac{n_m}{n_m + r}. \tag{12.67}$$

Note that GMM supervectors calculation is an approach to provide more abstraction in data by clustering the images of the video around certain means, but of course GMM supervectors do not keep temporal information.

12.4.2.1 Gaussian Mixture Models

A GMM is a semi-parametric estimation technique used to estimate a probability density function (PDF) from a set of data points drawn from this function. The PDF is regarded as a linear combination of M Gaussian functions (components of the model). The value of the PDF at a point $\mathbf{x}_i$ in the d-dimensional space is given by

$$p(\mathbf{x}_i) = \sum_{m=1}^{M} P(m) \cdot p(\mathbf{x}_i|m) \tag{12.68}$$

under the constraints:

$$0 \leq P(m) \leq 1 \qquad \sum_{m=1}^{M} P(m) = 1 \qquad \int_{\mathbb{R}^d} p(\mathbf{x}|m)d\mathbf{x} = 1 \tag{12.69}$$

$P(m)$ is the weight (prior probability) of component m and $p(\mathbf{x}_i|m)$ is the value of the Gaussian function described by component m. Thus, $p(\mathbf{x}_i|m)$ can be calculated as follows

$$p(\mathbf{x}_i|m) = \frac{1}{(2\pi)^{d/2}} \cdot |\mathbf{\Sigma}_m|^{-1/2} \cdot e^{-\frac{1}{2}\cdot(\mathbf{x}_i-\boldsymbol{\mu}_m)^T\cdot\mathbf{\Sigma}_m^{-1}\cdot(\mathbf{x}_i-\boldsymbol{\mu}_m)} \tag{12.70}$$

A GMM is completely defined by the means of its components, their covariance matrices and their weights. In order to estimate these parameters, the well known EM-algorithm is used [26].

12.5 Experimental Evaluation

12.5.1 Methodology

To evaluate the performance, the set of videos has been divided into training set and test set according to 8-fold cross validation that has been conducted 5 times; consequently, each test set has 44 videos (13, 11, 10 and 10 per class, respectively) while each training set consists of 314 videos. I selected three feature vectors (views) to be used for *Co-Training*: the orientation histogram from the mouth region (V_1) and the optical flow features extracted from the full facial region (V_2) and from the mouth region (V_3). For each fold, a UBM was created using the training videos. For each view and for each video, the pre-trained UBM is adapted in order to reflect the individual information of this video and the mean vectors of GMM components are concatenated to form the supervector that represent this video. The supervectors are normalized to have zero mean and unit variance, in order to avoid problems with outliers. The one-against-one decomposition scheme and Tri-Class support vector machines (with the Gaussian kernel function in Eq. (12.71)) are trained from the collected supervectors where the regularization term $C = 32$, the width of the kernel $\kappa = 0.3$ and tolerance parameter of approximate optimality condition $\tau = 0.001$.

$$\mathbb{K}(x, x_j) = exp(-\frac{\kappa}{2}\sum_{i=1}^{d}(x_i - x_{ji})^2) \tag{12.71}$$

For the GMM, various tests have been conducted to select the number of Gaussian components and the type of covariance matrix to be used by the GMM kernel function. As a result, I set the number of GMM components to two and diagonal

covariance matrix has been selected since it is a compromise between the full covariance matrix and the spherical model with a single scalar width parameter. In addition, it is computationally efficient and leads to more robust GMM estimates than full covariance matrix especially for small-sample size problems. Then, the training set of supervectors is split randomly into two sets L and U: 10% of the training examples of each class are used in L (9, 8, 7 and 7, respectively), while the remaining are in U. For each pair of classes, *Co-Training* has been performed until 3/4 the maximum number of iterations is reached.

12.5.2 Results and Discussion

Table 12.3 shows the average recognition rates on test sets that are summarize graphically in Figure 12.8. The baseline results with *Tri-Class SVM* and one-against-one approach using 20% labeled and no unlabeled data are given, as well as those when the whole training set is 100% labeled, to provide an upper bound to evaluate our framework. Not that $mvEns$ represents the average of the probabilistic outputs of the three *Tri-Class SVM*s trained on the different views for each pair of classes as defined in Eq. (12.1).

Table 12.3: The performance of single *Tri-Class SVM*s, multi-view ensembles ($mvEns$) and one-against-one ensembles ($1v1Ens$) on the facial expression recognitions task

		1-v-2	1-v-3	1-v-4	2-v-3	2-v-4	3-v-4	$1v1Ens$
20% Labeled	$SVM(V_1)$	77.37%	58.05%	70.97%	61.20%	75.19%	64.16%	
	$SVM(V_2)$	78.14%	67.06%	64.55%	70.49%	67.89%	58.42%	
	$SVM(V_3)$	74.18%	65.86%	66.15%	70.03%	70.38%	59.24%	
	$mvEns$	86.56%	73.26%	79.47%	77.33%	82.76%	70.34%	
20%+ Unlabeled	$SVM(V_1)$	81.08%	61.35%	78.77%	63.25%	81.79%	65.14%	
	$SVM(V_2)$	81.25%	68.91%	70.05%	73.54%	73.17%	61.10%	
	$SVM(V_3)$	81.72%	70.79%	73.29%	72.34%	75.86%	62.04%	
	$mvEns$	89.53%	73.99%	83.40%	78.07%	85.77%	71.66%	86.95%
improvement	$SVM(V_1)$	4.79%	5.69%	10.99%	3.35%	8.77%	1.52%	
	$SVM(V_2)$	3.98%	2.77%	8.52%	4.32%	7.78%	4.58%	
	$SVM(V_3)$	10.17%	7.49%	10.80%	3.29%	7.79%	4.71%	
	$mvEns$	3.43%	0.99%	4.95%	0.96%	3.64%	1.87%	
100% Labeled	$SVM(V_1)$	86.21%	73.71%	84.99%	75.13%	87.15%	75.25%	
	$SVM(V_2)$	84.14%	77.56%	75.17%	81.36%	78.76%	70.74%	
	$SVM(V_3)$	83.31%	75.67%	78.49%	76.19%	81.76%	69.62%	
	$mvEns$	93.15%	84.31%	89.52%	87.22%	91.37%	81.57%	91.45%

For all pairs of classes using 20% or 100% labeling rate, the ensemble $mvEns$ outperforms its member machines. For instance, the ensemble responsible to discriminate between the first and the second class achieves 86.56% recognition rate although its best member machine, $SVM(V_2)$, has only 78.14% accuracy. It means that the multi-view machines are not correlated (diverse). This means that these views are suitable to perform *Co-Training* since they satisfy the independence assumption of *Co-Training*.

The results have shown that the performance of the individual machines are

improved after using the unlabeled image sequences and the improvement ranges between 10.99% and 1.52%. For instance, the accuracy of $SVM(V_1)$ responsible to discriminate between the first and the second class increases from 77.37% to 81.08% which is 4.79% relative improvement.

Not only the base machines ($SVM(V_1)$, $SVM(V_2)$ and $SVM(V_3)$) but also their ensembles $mvEns$ are improved after *Co-Training* where the improvement ranges between 4.95% and 0.96%. For instance, the ensemble responsible to discriminate between the first and the second class is relatively improved by 3.43%. The ensemble resulting from combining the one-against-one multi-view ensembles ($1v1Ens$) achieves an accuracy 86.95% after using the unlabeled image sequences compared to 91.45% using the full training set. Hence, further investigation is required to minimize this gap.

There are two different architectures to combine the one-against-one scheme and *Co-Training*. The first architecture, proposed in this chapter, is to decompose the given multi-class problem into a set of binary problems using the one-against-one scheme then learning each of these binary problems through using *Co-Training*. The second architecture is to train a one-against-one ensemble on each view (feature sets) separately and to combine them using *Co-Training*. It would be beneficial to study the second architecture in future work.

12.6 Conclusion and Future Work

The main objective of this chapter is to show that there is an improvement from using unlabeled data when training one-against-one ensembles. Thus a learning framework is introduced that integrates *multi-view Co-Training* in the one-against-one output-space decomposition process where Tri-Class support vector machines are used as binary classifiers. The experiments have shown that *Co-Training* improves facial expression recognition system using unlabeled videos where the visual recognizers are initially trained with a small quantity of labeled videos. Since Tri-Class support vector machines are retrained several times during *Co-Training* iterations in order to benefit from the newly-labeled videos, a modified version of SMO algorithm is introduced for fast learning of *Tri-Class SVM*s because it is computationally expensive to use traditional quadratic programming algorithms to solve *Tri-Class SVM* optimization problems. In this experiment, *GMM supervectors* approach was applied to extract features from image sequences that are used further as input for *Tri-Class SVM*s. The *GMM supervectors* approach provides a flexible processing scheme for the classification of any type of sequential data.

An important factor that influence the performance of any *Co-Training* style algorithm is how to measure the confidence on predicting the label of an unlabeled example which determine its probability of being selected. The results shows that the proposed probabilistic *Tri-Class SVM* can provide effective estimates of class

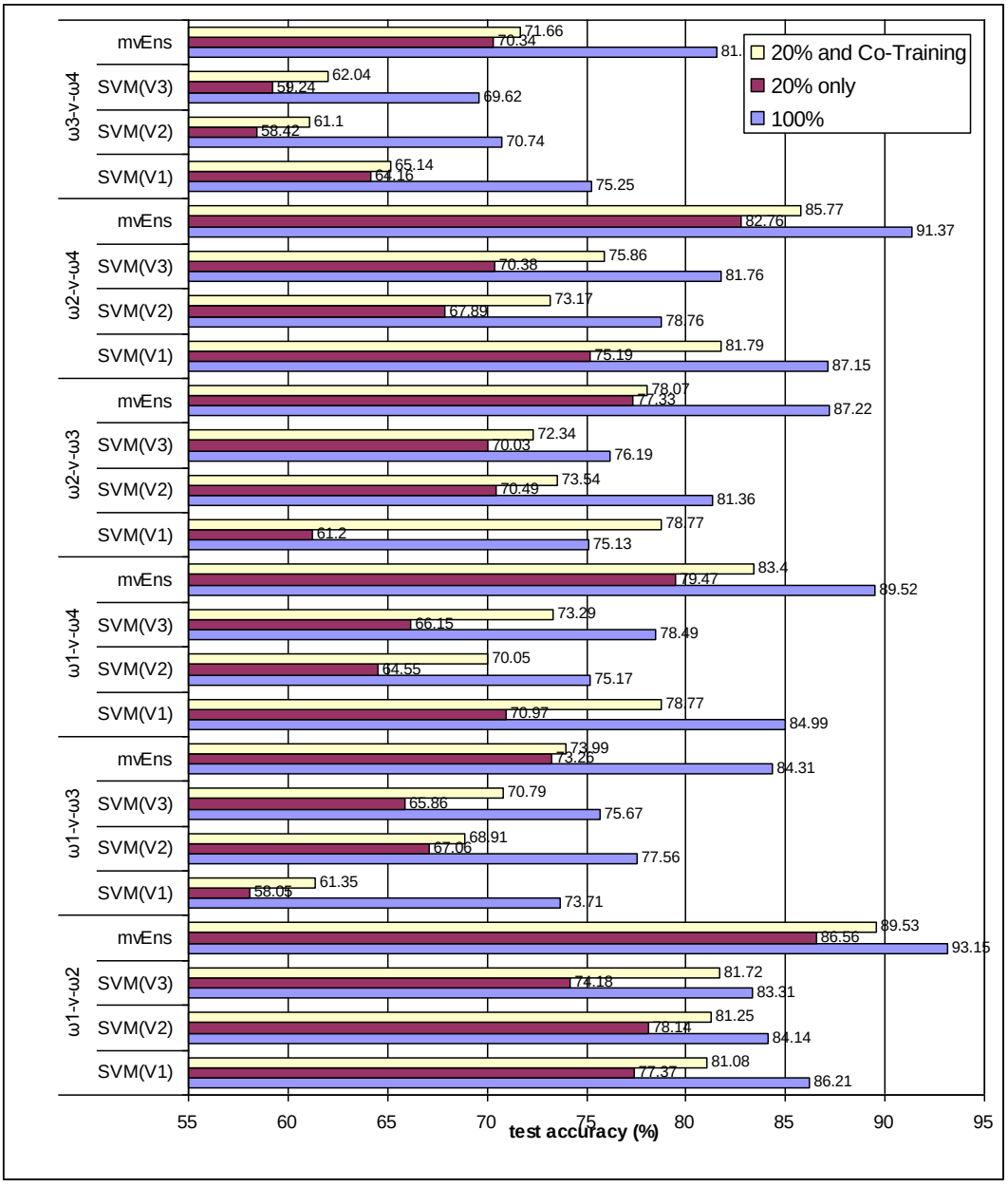


Figure 12.8: Average test accuracy percentage of *Tri-Class SVM*s and multi-view ensembles (*mvEns*) before and after *Co-Training*

probabilities that are used by *Co-Training* to measure confidence. There are many interesting directions for future work.

1. The reported experimental results are preliminary, the proposed framework will be evaluated on many real-world applications where there exist redundant and independent views.

2. *Co-Training* is sensitive to the initial videos that are provided as initially labeled examples. *Co-Testing* [134] is a *multi-view* active learning method that is inspired by *Co-Training*. Combining our framework with this method to select the initial videos should provide a better starting point for *Co-Training* than the random sampling currently used. This is an open issue that deserve investigation.

3. The tree-structured approach for multi-class decomposition performs comparable to the one-against-one approach with less number of classifiers. Schwenker et al. [168] applied successfully the tree-structured approach to binary-class SVMs. Future work should study the construction of *tree-structured Tri-Class SVMs* ensembles and compare them with the current *one-against-one Tri-Class SVMs* implementation. The evidence-theoretic hierarchical combination method should benefit from the fact that *Tri-Class SVM*, unlike conventional SVM, can discriminate between uncertainty and ignorance.

Chapter 13

Hierarchical Decision Templates based RBF Network Combiner

13.1 Introduction

Any multi-class decomposition approach consists of three stages: (1) decomposition of the multi-class problem into a set of simpler two-class problems, (2) solving these two-class problems and (3) combination of the intermediate solutions to yield the final decision. Ensemble methods can be divided into: *Flat* and *Hierarchical*. Flat architectures are the most popular ones where the members work independently disregarding the hierarchical structure of the classes. The *tree-structured* approaches construct an ensemble of K-1 binary classifiers where the objective is to improve the classification performance by taking into account the relationship and similarity among classes encoded into the class hierarchy.

The main motivation of this study are the following: (1) A key factor for the design of an effective ensemble is how to combine its member outputs to give the final decision. Although there are various methods to build the class hierarchy (first stage) and to solve the underlying binary-class problems (second stage), there is not much work to develop new combination methods that can best combine the intermediate results of the binary classifiers within the hierarchy (third stage). (2) The simple aggregation rules used for flat multiple classifier systems such as *minimum*, *maximum*, *average*, *product* and *majority vote* can not be applied to *hierarchical decision profiles*.

This chapter presents three main contributions: (1) A new trainable fusion method for a tree ensemble that integrates statistical information about its individual outputs, in the form of decision templates, into the training of an Radial Basis Function (RBF) network (Section 2.1). It is based on the assumption that the combined classifiers have real-valued outputs (soft classifiers) and is inspired by *Stacked Generalization* technique for combining multiple classifiers to improve generalization accuracy introduced by Wolpert [202]. (2) A new similarity measure based on *multivariate Gaussian function* to match a decision profile with

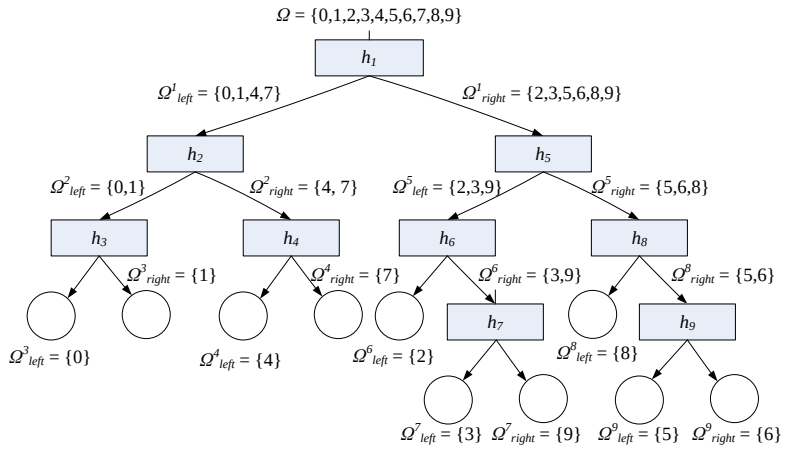


Figure 13.1: Class hierarchy constructed for the handwritten digits data set

decision templates. (3) The application of the *decision templates* combiner proposed by Kuncheva [109] for hierarchical ensembles. The work in this chapter has been previously published ([3]).

The tree-structured ensemble learning algorithm that is used in this chapter as well as the existing non-trainable decision fusion method for hierarchical ensembles, classical decision tree-like approach, product of the unique path and Dempster-Shafer evidence theory based method, are explained in Section 4.6. The remainder of this chapter is organized as follows: hierarchical decision profiles, the standard decision templates combiner and the proposed neural combiner is presented in Section 13.2. Section 15.6 contains the results of performance evaluation on nine multi-class visual object recognition tasks. Finally, the conclusion of the chapter is in Section 13.5.

13.2 Proposed Tree Combination Method

13.2.1 Hierarchical Decision Profile

For a given class hierachy as illustrated in Figure 13.1, the binary outputs of the K-1 internal node classifiers for each training example x can be stored in a decision profile $DP(x)$ as the matrix in Figure 13.2. Based on the way of using $DP(x)$ to find the overall support for each class k, the fusion methods are divided by Kuncheva [107] into **class-conscious** and **class-indifferent**. The **class-conscious** methods use only the k^{th} column of $DP(x)$ such as *minimum*, *maximum*, *average* and *product* rules. This type of methods uses the context of the profile but loses part of the information because it does not take into account the columns of the other classes. On the other hand, the **class-indifferent**

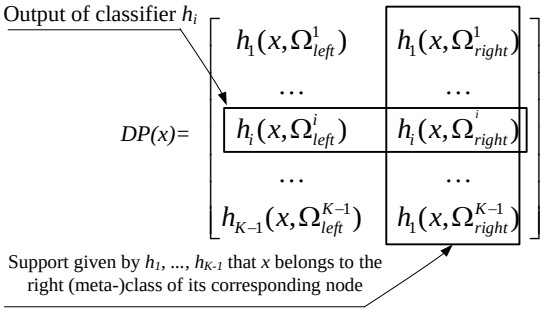


Figure 13.2: Decision profile for example x using tree-structured ensemble members

methods ignore the context of the profile and use all of $DP(x)$ as features in a new feature space, which is called the *intermediate feature space* and depicted in Figure 13.3. From Figure 13.2, one can observe that the **class-conscious** fusion methods can not be used with the *hierarchical decision profile* because the meta-classes are not the same at different rows. Hence, a **class-indifferent** fusion method is required where the final decision of the tree ensemble is made by another classifier that takes the intermediate feature space as input.

13.2.2 Standard Decision Templates Combiner

This trainable combiner was proposed by Kuncheva [109]. At the training phase, a decision template (DT_k) is calculated for each class k as the mean of the decision profiles of the training examples belonging to class k.

$$DT_k = \frac{1}{N_k} \sum_{y_i=k} DP(x_i) \tag{13.1}$$

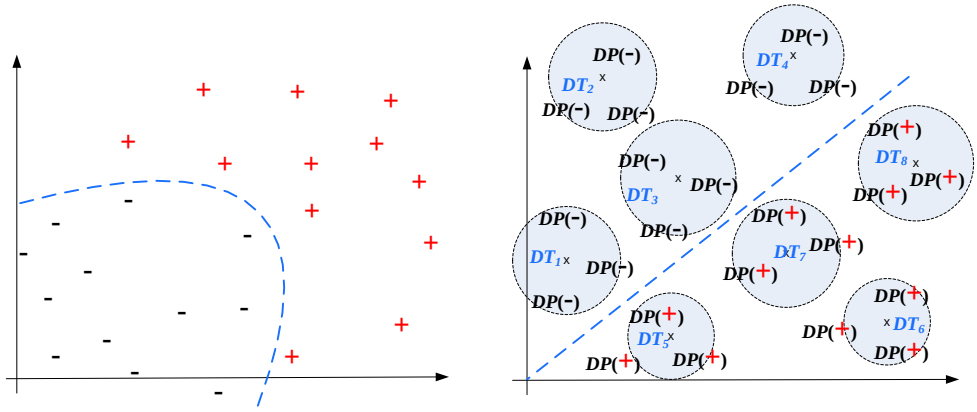
At the classification phase, the decision profile for an instance x is matched to the K decision templates using a similarity measure. The class label with the closest decision template will be assigned to x. In [109], Kuncheva discussed 11 different similarity measures and compared them with 14 other techniques. The most popular similarity measures are S_1 measure,

$$\mu_k(x) = S_1(DP(x), DT_k) = \frac{\sum_{i=1}^{K-1} \sum_{j=1}^{2} min(dp(i,j), dt_k(i,j))}{\sum_{i=1}^{K-1} \sum_{j=1}^{2} max(dp(i,j), dt_k(i,j))} \tag{13.2}$$

and the normalized Euclidean distance,

$$\mu_k(x) = N(DP(x), DT_k) = 1 - \frac{1}{(K-1) \times 2} \sum_{i=1}^{K-1} \sum_{j=1}^{2} (dp(i,j) - dt_k(i,j))^2 \tag{13.3}$$

This combination rule is equivalent to applying the *nearest mean classifier* (Section 2.2.3) in the profile space.



(a) D-dimensional input space

(b) 2×(K-1)-dimensional intermediate feature space (Profile Space)

Figure 13.3: An illustrative example for data transformation

13.2.3 RBF Network Combiner using Decision Templates

An *RBF network classifier* (Section 2.1) is applied in the intermediate feature space instead of the *nearest mean classifier* applied by the above *Decision Templates* combiner. *Multivariate Gaussian function* ϕ_j is used as an RBF at hidden nodes. Since the hidden layer applies a nonlinear transformation to the input data, class separation should be much easier in the profile space (see Figure 13.3). The output vector f for a given instance x is produced at the final output layer from the weighted summation of the activations of the Gaussian kernels ϕ_j's.

$$f_k(x) = \sum_{j=1}^{K \times c} w_{jk} \phi_j(\|DP(x) - DT_j\|) \;\; where \;\; k = 1, \ldots, K \tag{13.4}$$

The two-phase learning procedure discussed in Section 2.1.2 is used for training RBF network combiner using the same training set that is used to construct the ensemble members. In the first phase, for each class k, c decision templates are calculated by applying c-means clustering algorithm (Section 2.1.4.1) on the decision profiles of all training examples that belong to class k. After clustering, the $K \times c$ clustered decision templates are used as the RBF centers. Then the width of the j^{th} RBF (σ_j) is set to the distance between the decision template DT_j and the nearest template of different class multiplied by α as in Eq. (13.5) where α should control the degree of overlap between adjacent Gaussian nodes (in our experiments, α=1).

$$\sigma_j = \alpha \min_{i=1,\ldots,K \times c} \left\{ \|DT_j - DT_i\|_2 : i \neq j, class(DT_i) \neq class(DT_j) \right\} \tag{13.5}$$

Then, the radial basis function ϕ_j is defined as follows,

$$\phi_j(\|DP(x) - DT_j\|) = exp(-\frac{\|DP(x) - DT_j\|_2^2}{2\sigma_j^2}) \tag{13.6}$$

Then, in the second learning phase the output layer weights W are determined by minimizing the MSE at the network output by a matrix pseudo-inverse technique using singular value decomposition, $W = \Phi^+ T$, where T is the matrix of target outputs of the m training examples where the *1-out-of-K* coding scheme is used and Φ is the activation matrix,

$$\Phi_{ij} = \phi_j(\|DP(x_i) - DT_j\|)_{j=1,\ldots,K\times c}^{i=1,\ldots,m} \tag{13.7}$$

Therefore, calculating the pseudo-inverse of Φ provides a least squares solution to the system of linear equations $T = \Phi W$. This direct computation is faster than the gradient descent optimization and yields good classification results. After this step, all parameters of the RBF network have been determined and it can be used as a combination method for the tree-structured ensemble.

13.3 Experimental Results

13.3.1 Methodology

An experimental study is conducted to compare the proposed tree combiner (*RBFN*) with classical decision tree-like approach (*Hard*), product of the unique path combiner (*Product*), Dempster-Shafer evidence theory based combiner (*DS*) and standard *Decision Templates* combiner using S_1 measure (*DT:*S_1) and normalized Euclidean distance (*DT:NM*). The two-phase learning algorithm used to train the *RBFN* tree combiner is used also to learn the binary RBF network classifier at each node. Except that meta-class specific c-means clustering algorithm (with $c = 10$) is applied independently to the training examples that belong to each meta-class. The nine real-world data sets used in this study are described in Table 7.1 in Chapter 7. For simplicity, *Fruits1* denotes the *colorhist3x3* feature type and *Fruits2* denotes the *sobel4x4* feature type for the *fruits* recognition task. For the *COIL20* recognition task, *COIL1* denotes the *colorhist1x1* feature type and *COIL2* represents the *orienthist2x2* feature type. For the *digits* recognition task, *Digits1* refers to the *pca-40* feature type and *Digits2* denotes the *image-vector* feature type. I intentionally select data sets with variance in number of features, number of classes and number of examples. All implementation was carried out using the WEKA library [201].

For each data set and tree combiner, 5 runs of 10-fold cross-validation have been performed. The *(Win/Tie/Loss)* record presents three values, the number of data sets for which algorithm A is significantly better, equal, or worse than

algorithm B with respect to classification accuracy, using *corrected paired t-test*, see Section 7.2.3, implemented in WEKA at 0.05 significance level. The accuracy in our experiments is less than the results on the same data sets reported elsewhere because we use a random subset of the available data to save computation time. Our main concern is the relative accuracy between different combiners.

13.3.2 Results

Table 13.1 shows the average test accuracies and standard deviations. For each data set, the highest accuracy achieved is bold faced. The result with bullet(•)/open circle(°) mark indicates that the *RBFN* combiner is significantly better/worse than the respective combiner for the respective data set. We conclude that the *RBFN* combiner significantly outperforms *Hard*, *Product* and *DS* combiners in seven of the nine domains and its behavior is statistically indistinguishable in the remaining two domains. In addition, the *RBFN* combiner is significantly superior to the *DT:*S_1 and *DT:NM* in eight and seven of the nine domains, respectively.

Table 13.1: *RBF Network* against the other tree combiners, using 100% of the data

Dataset	*RBFN*(c=3)	*Hard*	*Product*	*DS*	*DT:*S_1	*DT:NM*
Fruits1	**97.05 ± 1.82**	95.95 ± 2.15	96.26 ± 1.94•	96.21 ± 1.99	95.76 ± 2.48	96.52 ± 1.95
Fruits2	**94.90 ± 2.44**	92.21 ± 3.68•	92.79 ± 3.25•	92.86 ± 3.35•	92.64 ± 3.06•	93.67 ± 2.76•
COIL1	**93.89 ± 2.07**	89.08 ± 2.14•	90.81 ± 1.74•	90.54 ± 2.13•	88.60 ± 2.32•	91.33 ± 2.10•
COIL2	**98.75 ± 0.86**	95.94 ± 1.82•	97.72 ± 1.24•	97.21 ± 1.25•	94.15 ± 1.70•	96.54 ± 1.32•
Digits1	**93.58 ± 1.84**	84.19 ± 2.72•	88.88 ± 2.37•	87.68 ± 2.56•	91.82 ± 2.06•	92.13 ± 2.16•
Digits2	**94.46 ± 1.61**	92.11 ± 2.03•	92.61 ± 1.50•	92.90 ± 1.64•	92.38 ± 1.80•	93.24 ± 1.53•
Letters	**80.37 ± 2.74**	68.74 ± 4.34•	73.29 ± 3.93•	72.11 ± 4.07•	71.15 ± 3.22•	73.24 ± 3.12•
Texture	**96.27 ± 1.74**	94.45 ± 1.92•	95.73 ± 1.88	95.05 ± 1.87•	93.65 ± 2.07•	94.45 ± 1.68•
Satimage	**87.92 ± 2.53**	87.49 ± 2.39	87.68 ± 2.61	87.67 ± 2.56	86.17 ± 2.96•	86.59 ± 2.83
ave.	**93.02**	88.91	90.64	90.25	89.59	90.86
(Win/Tie/Loss)		(0/2/7)	(0/2/7)	(0/2/7)	(0/1/8)	(0/2/7)

13.3.3 Influence of the Training Set Size

One might expect that the performance of *RBFN* combiner would be very poor with small training sets because it is a trainable combiner. To study the influence of the training set size, we evaluate the different tree combiners using only 40% of the available training set (see Table 13.2). The significance is again indicated with bullets and open circles. From the results, we conclude that sample size has no apparent influence on the benefits of *RBFN* combiner because it still works very well with small samples. That is, *RBFN* combiner significantly outperforms *Hard*, *Product*, *DS*, *DT:*S_1 and *DT:NM* combiners in eight, six, seven, nine and seven of the nine domains, respectively.

In addition, we observe the behavior of the different tree combiners when the labeled training set size is increased to 60% and 80% of the available data (see Table 13.3 and Table 13.4). For the case of 60%, one can conclude that *RBFN*

Table 13.2: *RBF Network* against the other tree combiners, using 40% of the data

Dataset	*RBFN*(c=3)	*Hard*	*Product*	*DS*	*DT:*S_1	*DT:NM*
Fruits1	**96.10 ± 1.88**	94.74 ± 2.26$^\bullet$	94.95 ± 2.33	95.21 ± 2.31	94.31 ± 2.03$^\bullet$	95.24 ± 2.11
Fruits2	**92.83 ± 3.11**	90.62 ± 3.09$^\bullet$	89.86 ± 3.79$^\bullet$	90.31 ± 3.69$^\bullet$	90.05 ± 3.56$^\bullet$	90.69 ± 3.55$^\bullet$
COIL1	**93.15 ± 2.27**	86.83 ± 2.53$^\bullet$	88.68 ± 2.42$^\bullet$	88.54 ± 2.27$^\bullet$	86.96 ± 2.67$^\bullet$	89.35 ± 2.76$^\bullet$
COIL2	**97.94 ± 1.14**	93.93 ± 2.00$^\bullet$	95.82 ± 1.58$^\bullet$	95.46 ± 1.74$^\bullet$	91.88 ± 2.09$^\bullet$	94.61 ± 1.83$^\bullet$
Digits1	**92.01 ± 1.81**	82.09 ± 2.97$^\bullet$	86.58 ± 2.72$^\bullet$	85.10 ± 2.74$^\bullet$	89.95 ± 1.99$^\bullet$	90.30 ± 1.85$^\bullet$
Digits2	**93.78 ± 1.64**	91.27 ± 2.03$^\bullet$	91.68 ± 2.05$^\bullet$	91.90 ± 1.87$^\bullet$	91.49 ± 1.97$^\bullet$	92.72 ± 1.73$^\bullet$
Letters	**78.21 ± 3.05**	65.06 ± 4.06$^\bullet$	69.70 ± 3.43$^\bullet$	68.66 ± 3.66$^\bullet$	68.66 ± 3.49$^\bullet$	70.66 ± 3.79$^\bullet$
Texture	**95.33 ± 1.89**	93.53 ± 1.94$^\bullet$	94.33 ± 2.18	94.02 ± 2.01$^\bullet$	92.51 ± 2.41$^\bullet$	93.38 ± 2.15$^\bullet$
Satimage	**87.32 ± 3.12**	86.19 ± 2.93	86.58 ± 2.95	86.44 ± 2.98	85.70 ± 3.13$^\bullet$	86.06 ± 3.40
ave.	**91.85**	87.14	88.69	88.40	87.94	89.22
(Win/Tie/Loss)		(0/1/8)	(0/3/6)	(0/2/7)	(0/0/9)	(0/2/7)

combiner significantly outperforms *Hard*, *Product*, *DS* and *DT:*S_1 combiners in seven, six, seven and nine of the nine domains, respectively. For the case of 80%, one concludes that *RBFN* combiner significantly outperforms *Hard*, *Product*, *DS* and *DT:*S_1 combiners in eight, six, seven and eight data sets, respectively.

Table 13.3: *RBF Network* against the other tree combiners, using 60% of the data

Dataset	*RBF Network*	*Hard*	*Product*	*DS*	*DT:S1*
Fruits1	**96.57(2.13)**	95.57(2.14)	95.88(1.90)	95.83(2.04)	95.31(2.02)$^\bullet$
Fruits2	**93.88(2.19)**	91.24(3.48)$^\bullet$	91.14(3.19)$^\bullet$	91.33(3.32)$^\bullet$	91.40(2.70)$^\bullet$
COIL1	**93.38(2.31)**	87.72(3.03)$^\bullet$	89.43(2.23)$^\bullet$	89.21(2.52)$^\bullet$	87.74(2.33)$^\bullet$
COIL2	**98.21(1.10)**	94.85(2.11)$^\bullet$	96.69(1.51)$^\bullet$	96.19(1.58)$^\bullet$	92.69(2.30)$^\bullet$
Digits1	**92.88(1.88)**	82.98(2.56)$^\bullet$	87.75(2.62)$^\bullet$	86.39(2.64)$^\bullet$	91.09(2.10)$^\bullet$
Digits2	**94.36(1.49)**	91.74(1.78)$^\bullet$	92.07(1.41)$^\bullet$	92.44(1.67)$^\bullet$	92.02(1.92)$^\bullet$
Letters	**79.34(2.99)**	66.54(3.76)$^\bullet$	71.17(3.35)$^\bullet$	69.71(3.48)$^\bullet$	70.01(3.28)$^\bullet$
Texture	**95.69(1.65)**	93.62(2.14)$^\bullet$	94.78(1.99)	94.27(2.10)$^\bullet$	92.40(2.41)$^\bullet$
Satimage	**87.53(2.90)**	86.56(3.02)	86.86(2.93)	86.58(3.10)	85.70(3.48)$^\bullet$
ave.	**92.43**	87.87	89.53	89.11	88.71
(Win/Tie/Loss)		(0/2/7)	(0/3/6)	(0/2/7)	(0/0/9)

Table 13.4: *RBF Network* against the other tree combiners, using 80% of the data

Dataset	*RBF Network*	*Hard*	*Product*	*DS*	*DT:S1*
Fruits1	**97.31(1.80)**	96.02(2.11)$^\bullet$	96.26(2.33)	96.14(2.08)	95.86(2.31)$^\bullet$
Fruits2	**94.90(2.84)**	92.43(3.35)$^\bullet$	92.26(3.12)$^\bullet$	92.45(3.35)$^\bullet$	92.31(2.63)$^\bullet$
COIL1	**93.90(2.13)**	88.11(2.27)$^\bullet$	90.31(2.20)$^\bullet$	89.67(2.07)$^\bullet$	88.31(2.59)$^\bullet$
COIL2	**98.49(1.20)**	95.38(2.08)$^\bullet$	97.35(1.45)$^\bullet$	96.86(1.57)$^\bullet$	93.44(1.86)$^\bullet$
Digits1	**93.27(2.03)**	83.69(2.72)$^\bullet$	88.43(2.54)$^\bullet$	87.13(2.57)$^\bullet$	91.31(2.12)$^\bullet$
Digits2	**94.23(1.73)**	91.77(1.73)$^\bullet$	92.42(1.65)$^\bullet$	92.57(1.57)$^\bullet$	92.22(1.84)$^\bullet$
Letters	**80.01(3.07)**	67.96(4.35)$^\bullet$	72.49(3.67)$^\bullet$	71.46(3.97)$^\bullet$	70.84(3.11)$^\bullet$
Texture	**95.91(1.88)**	94.24(2.06)$^\bullet$	95.29(2.15)	94.78(1.95)$^\bullet$	93.35(2.11)$^\bullet$
Satimage	**87.48(3.02)**	87.20(2.73)	87.26(2.80)	87.17(2.88)	86.23(2.94)
ave.	**92.83**	88.53	90.23	89.80	89.32
(Win/Tie/Loss)		(0/1/8)	(0/3/6)	(0/2/7)	(0/1/8)

13.3.4 Influence of Number of Decision Templates per Class

In all the previous experiments, *RBFN* combiner was trained with 3 clustered decision templates per class (c=3). To study the influence of the number of decision templates per class, we measured test accuracies of the *RBFN* combiner using one, 7, 10, 15 and 20 decision templates per class (see Table 13.5). The significance is again indicated with bullets and open circles. From the results, we can conclude that *RBFN* combiner with c >3 significantly outperforms *RBFN* with c=3 in only three data sets and the improvement is insignificant in the remaining domains. In addition, *RBFN* combiner with c =3 significantly outperforms *RBFN* with c=1 in seven of the nine tasks. Although the *RBFN* combiner with c=1 and both *Decision Templates* combiners are trained with one decision template per class, *RBFN* combiner outperforms both *Decision Templates* combiners (*DT:*S_1 and *DT:NM*) due to its trainable output layer and nonlinear behavior.

Table 13.5: Average test accuracy for *RBF Network* combiner with different number of clustered decision templates per class (c), using 100% of data

Dataset	*RBFN*(c=3)	*RBFN*(c=1)	*RBFN*(c=7)	*RBFN*(c=10)	*RBFN*(c=15)	*RBFN*(c=20)
Fruits1	97.05 ± 1.82	96.83 ± 1.87	97.31 ± 1.78	97.38 ± 1.61	**97.52 ± 1.81**	97.36 ± 1.76
Fruits2	94.90 ± 2.44	93.67 ± 2.81$^\bullet$	95.14 ± 2.42	95.24 ± 2.49	95.26 ± 2.58	95.55 ± 2.41
COIL1	93.89 ± 2.07	92.25 ± 2.17$^\bullet$	95.33 ± 1.90°	96.15 ± 1.83°	96.68 ± 1.71°	**96.74 ± 1.61**$^\circ$
COIL2	98.75 ± 0.86	97.12 ± 1.32$^\bullet$	99.57 ± 0.52°	99.76 ± 0.41°	99.88 ± 0.30°	**99.92 ± 0.27**$^\circ$
Digits1	93.58 ± 1.84	92.67 ± 1.98$^\bullet$	94.10 ± 1.71	94.20 ± 1.72	94.13 ± 1.66	**94.26 ± 1.74**
Digits2	94.46 ± 1.61	93.76 ± 1.67$^\bullet$	94.70 ± 1.50	94.78 ± 1.64	94.86 ± 1.63	**94.96 ± 1.51**
Letters	80.37 ± 2.74	75.14 ± 3.42$^\bullet$	83.93 ± 2.65°	85.06 ± 2.93°	86.36 ± 2.42°	**87.27 ± 2.75**$^\circ$
Texture	96.27 ± 1.74	95.13 ± 1.91$^\bullet$	96.93 ± 1.62	97.05 ± 1.50	97.25 ± 1.37	**97.27 ± 1.54**
Satimage	87.92 ± 2.53	87.59 ± 2.69	**88.27 ± 2.35**	88.20 ± 2.35	88.13 ± 2.26	87.92 ± 2.35
ave.	93.02	91.57	93.92	94.20	94.45	**94.58**
(Win/Tie/Loss)		(0/2/7)	(3/6/0)	(3/6/0)	(3/6/0)	(3/6/0)

13.4 Related Work

In [53], Dietrich et al. introduced the concept of multiple decision templates per class in the context of time series classification. They described two types of decision template: temporal decision templates and clustered decision templates. For each time series, a temporal decision template for its associated class is computed through the average of the decision profiles defined over all time windows. Thus, the number of decision templates per class is the number of training time series belonging to this class. For each class, the clustered decision templates are determined by clustering the decision profiles for each time series through a clustering algorithm such as k-means, fuzzy k-means or Kohonens self organised feature map.

13.5 Conclusion and Future Directions

In this chapter, a new soft trainable fusion method for tree-structured multiple classifier systems, used in multi-class problems, is introduced. The proposed model integrates statistical information about the individual binary classifier outputs (in the form of *clustered decision templates*) into an RBF network combiner. Multivariate Gaussian function was used as similarity measure to match a *hierarchical decision profile* with decision templates. Not only RBF network was used as combiner but also it was used to construct the ensemble classifiers. The experiments were conducted on nine real-world multi-class object recognition tasks including digits, letters, fruits, 3d objects and textures. The experiments have shown that the *RBF Network* tree combiner significantly outperforms the three existing non-trainable tree combiners and the *decision templates* based combiner proposed by Kuncheva. The results also demonstrate that this neural combiner is robust to changes in the training set size and the number of decision templates per class. As a future work, further study should investigates the exploitation of the available abundant unlabeled data to improve the performance of the neural combiner. To the best of my knowledge, although there is a lot of work that study the benefits of unlabeled data on base classifiers, there is no work devoted to study incorporating unlabeled data in the training of combination rules.

Dietrich et al. [53] suggest that the second training set R, on which the combiner should be trained, may be partly overlapping with the first training set used for the individual classifiers L. In this current work, the same the training set is used to train the neural combiner and the individual classifiers ($R = L$). Further work should study the influence of training the neural combiner using a different training set. In addition, it should investigate the influence of overlapping both R and L with different degrees on the performance of the neural combiner.

Chapter 14

Multi-View Forest

This chapter proposes a new ensemble method that constructs an ensemble of tree-structured classifiers using multi-view learning. An ensemble can outperform its individual classifiers if these classifiers are diverse and accurate. In order to construct diverse individual classifiers, in this chapter it is assumed that the object to be classified is described by multiple feature sets (views). The aim is to construct different tree classifiers using different combinations of views to improve the accuracy of the multi-class learning (Chapter 4). For the decision fusion of the binary classifiers within each tree classifier, Dempster's unnormalized rule of combination is applied and an evidence-theoretic decision profile is proposed to combine the decisions of different trees. Experiments have been performed on two real-world data sets: a data set of handwritten digits, and another data set of 3D visual objects. The results indicate that the proposed forest efficiently integrates multi-view data and outperforms the individual tree classifiers. The work in this chapter has been previously published ([6, 1, 11]).

14.1 Introduction

The growing interest in combining classifiers is due to the fact that finding the best individual classifier for a classification task is difficult from a statistical, computational and representational perspective (Chapter 3). The use of multiple classifiers allows the exploitation of complementary discriminating information that the group of classifiers may provide. Therefore, the objective of combining such a group of classifiers is to produce a more accurate classifier decision than a single classifier. The main motivation for proposing the new ensemble method is the fact that error diversity is an essential requirement to build an effective classifier ensemble. Diversity among classifiers means that they have independent (uncorrelated) errors. Many approaches for constructing ensembles of diverse individual classifiers have been developed. One approach is based on combining classifiers trained on different training sets, i.e. bagging [31] and boosting

[65]. Another approach to promote the diversity is based on combining classifiers trained on different feature subsets, such as Random Subspace Method [82] and Random Forests [32]. In general, feature subset selection is time-consuming and sometimes deleting some of the features degrades the performance of the individual classifiers due to the potential loss of information. Hence this approach is efficient only if the features are redundant.

In this chapter, a new ensemble method, denoted as *Multi-view Forest*, is proposed, a set of tree-structured ensembles (Chapter 4.6) are chosen as the base classifiers. Tree classifiers are used to decompose multi-class recognition problems into less complex binary classification subtasks. At the classification phase, a soft combination rule based on Dempster-Shafer evidence theory is applied to fuse the intermediate results of the node classifiers and to provide the final class probability estimates (*CPE*) of each tree t. After that the *CPE*s provided by different trees H_t are combined to provide the final decision of the forest. Due to the diversity among trees, we expect that the forest will outperform its member trees.

14.2 Multi-View Forest

A *multi-view forest* is an ensemble of tree-structured classifiers, and a tree classifier can be seen as a hierarchical ensemble of binary classifiers which solves a multi-class classification task using a single feature set (called *Single-View Tree*) or even a group of feature sets, then it is called *Multi-View Tree*. Let $L = \{(X_\mu, y_\mu)|X_\mu = (x_\mu^{(1)}, \ldots, x_\mu^{(n)}), y_\mu \in \Omega, \mu = 1, \ldots, m\}$ be the training data set where X_μ is an example described by n D_i-dimensional feature vectors $x_\mu^{(i)} \in \mathbb{R}^{D_i}$, y_μ denotes the class label of X_μ and $\Omega = \{\omega_1, \ldots, \omega_K\}$ is the set of classes. Let $H_1, \ldots, H_N$ denote the N tree classifiers that constitutes a *multi-view forest* (see Figure 14.1). The proposed method as any multi-view learning algorithm is applied only in real-world domains where each example is represented by two or more sufficient and independent sets of features.

14.2.1 Multi-View Learning

Multi-view learning is based on the assumption that each pattern is represented by multiple types of features obtained through different physical sources and sensors or derived by different feature extraction procedures. For example, a web page can be represented by different views, e.g. a distribution of words used in the web page, hyperlinks that point to this page, and any other statistical information. Multi-view learning was introduced for semi-supervised learning by Blum and Mitchell in the context of Co-training [30], where they proved that having multiple representations of a particular object can improve the classifier performance using unlabeled data.

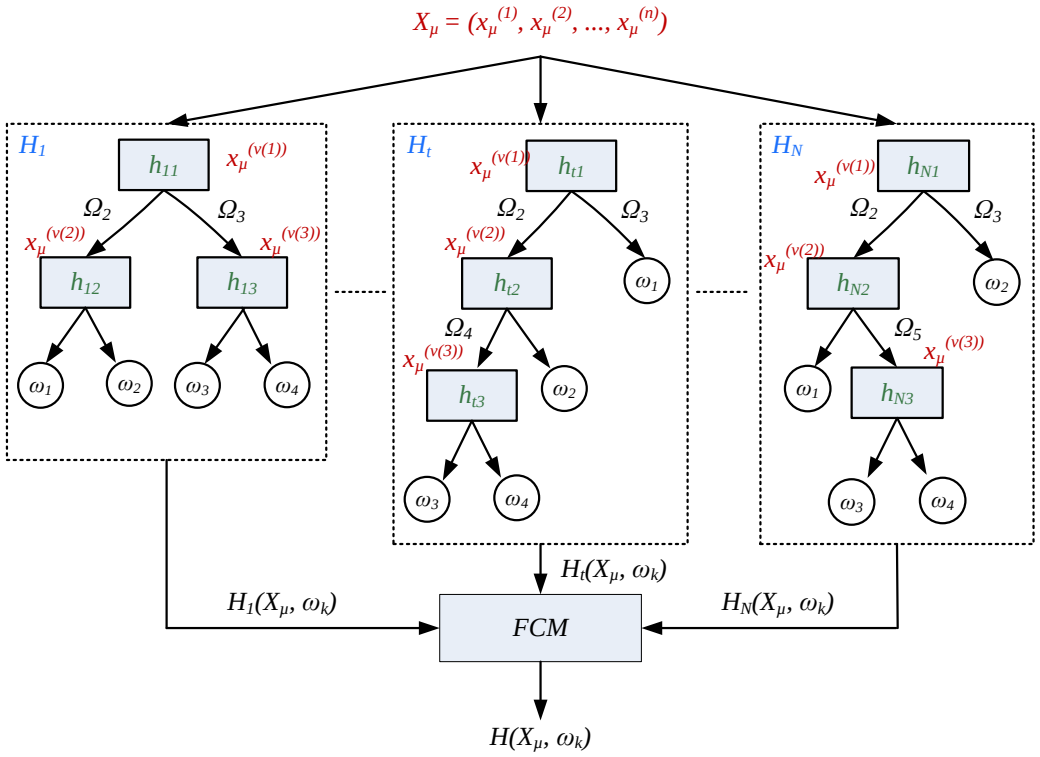


Figure 14.1: *An illustration of a Multi-View Forest*

Multi-view learning has been applied in clustering as well, for instance Gupta and Dasgupta [75] proposed a multi-view hierarchical clustering algorithm (see Algorithm 25 and Figure 14.2). It depends on the assumption that different views may have different distance measures leading to different clusterings. This method seems to work better than taking a linear combination of the distance measures, or appending the different feature sets together into a single feature vector. A tree structure is constructed through a bottom-up approach where the best feature set (view) is selected for each node in the tree. To select the best feature set, agglomerative clustering is applied at each feature space to produce a pair of clusters. Intuitively, the required best feature set is the one that provides the most well-separated clusters. The *SD* validity index [76] is used to measure the quality of each pair of clusters. Its definition is based on the concepts of total separation between clusters (inter-cluster distance) and average compactness of clusters (intra-cluster distance).

14.2.2 Tree-Structured Multi-class Decomposition

The task of the tree-structured approach as shown in Algorithm 4 is to decompose a given K-class problem into a set of simpler K-1 binary problems and to train classifiers to solve the binary problems at the internal nodes within the tree

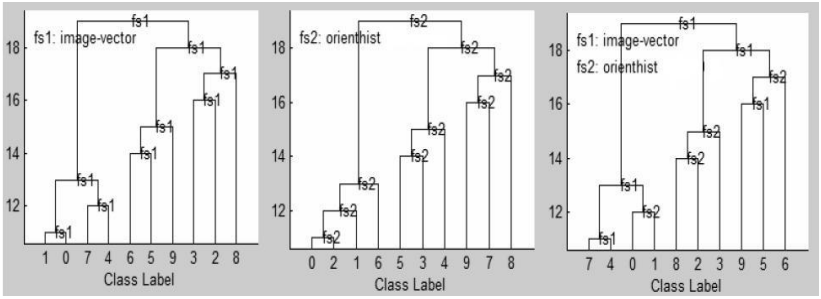


Figure 14.2: Dendrograms constructed for digits data represented by: *image-vector* (fs_1), and orientation histograms (fs_2).

through a base learning algorithm ($BaseLearn$). In the classification phase, for a given instance x, the intermediate results of the internal classifiers are combined through a given combination method ($TreeCombiner$) to produce the final decision of the ensemble.

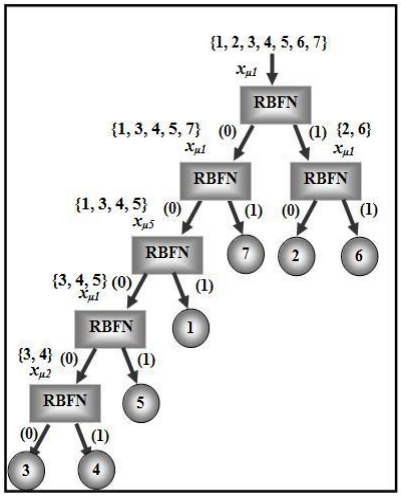


Figure 14.3: *Multi-View Tree* constructed using *Bottom-Up approach* for Fruits data represented by: Color histograms (x_1), orientation histograms utilizing canny edge detector (x_2) and orientation histograms using opponent colors red and green (x_5).

14.2.2.1 Generate Class Hierarchy

There are various ways to build the tree structure, e.g. user-defined and class-similarity based approaches. In the handwritten digits recognition problem for instance, the user might construct two meta-classes by separating the digits $\{0, 1, 2, 3, 4\}$ in one meta-class and the rest in the other meta-class. If the class hierarchy takes into account the relationships among classes, it provides important domain knowledge that might lead to improve the classification accuracy

Algorithm 25 *BuildNode* - (Bottom-Up Approach)

Require: set of classes assigned to tree node j (Ω_j), set of centroids of classes in meta-class Ω_j (C_j)

if $|\Omega_j| = 1$ **then**
  Add a leaf node j to *hierarchy* that represents class Ω_j
else
  Add an internal node j to *hierarchy* that represents meta-class Ω_j
  for $i = 1$ to n **do**
    Initially, put each class in Ω_j in a separate cluster
    repeat
      Get the two most close clusters in Ω_j
      Merge these two clusters into a new cluster
    until the number of remaining clusters is two
    Denote the remaining clusters, $\Omega_{2j}^{(i)}$ and $\Omega_{2j+1}^{(i)}$
    Calculate the SD validity index,
    $$SD_i = \frac{distance\ between\ clusters\ \Omega_{2j}^{(i)}\ and\ \Omega_{2j+1}^{(i)}}{average\ compactness\ of\ clusters\ \Omega_{2j}^{(i)}\ and\ \Omega_{2j+1}^{(i)}}$$
  end for
  Get the best view split, $i^* = \arg\max_{1 \le i \le n} SD_i$
  $C_{2j} \leftarrow$ set of centroids of classes in $\Omega_{2j}^{(i^*)}$
  $BuildNode(\Omega_{2j}^{(i^*)}, C_{2j})$
  $C_{2j+1} \leftarrow$ set of centroids of classes in $\Omega_{2j+1}^{(i^*)}$
  $BuildNode(\Omega_{2j+1}^{(i^*)}, C_{2j+1})$
end if
return *hierarchy*

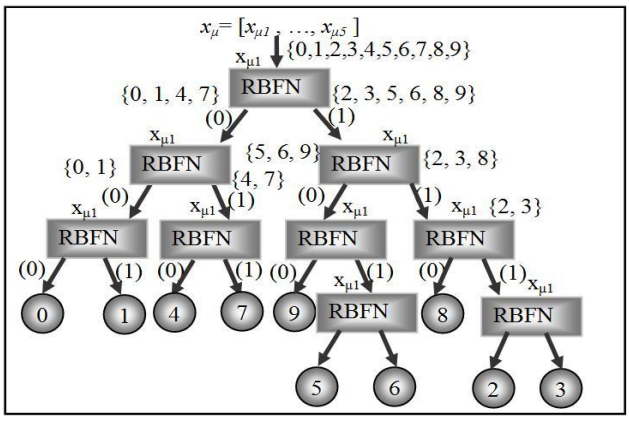


Figure 14.4: *Single-View Tree* constructed using *Top-Down* approach for digits

Algorithm 26 *BuildNode* - (Top-Down Approach)

Require: set of classes assigned to tree node j (Ω_j), set of centroids of classes in meta-class Ω_j (C_j)

if $|\Omega_j| = 1$ **then**
  Add a leaf node j to *hierarchy* that represents class Ω_j
else
  create an internal node j that represents meta-class Ω_j
  Add $node_j$ to *hierarchy*
  for $i = 1$ to n **do**
    Get the two most distant classes in Ω_j: $(c_{j1}, \omega_{j1}), (c_{j2}, \omega_{j2})$
    $\{\Omega_{2j}^{(i)}, \Omega_{2j+1}^{(i)}\} = seeded\text{-}k\text{-}means(C_j, c_{j1}, c_{j2})$
    Calculate the SD validity index,
    $SD_i = \frac{distance\ between\ clusters\ \Omega_{2j}^{(i)}\ and\ \Omega_{2j+1}^{(i)}}{average\ compactness\ of\ clusters\ \Omega_{2j}^{(i)}\ and\ \Omega_{2j+1}^{(i)}}$
  end for
  Get the winner view, $i^* = \arg\max_{1 \leq i \leq n} SD_i$
  $C_{2j} \leftarrow$ set of centroids of classes in $\Omega_{2j}^{(i^*)}$
  $BuildNode(\Omega_{2j}^{(i^*)}, C_{2j})$
  $C_{2j+1} \leftarrow$ set of centroids of classes in $\Omega_{2j+1}^{(i^*)}$
  $BuildNode(\Omega_{2j+1}^{(i^*)}, C_{2j+1})$
end if
return *hierarchy*

[105]. That is, the class hierarchy should satisfy the well-known cluster assumption: similar classes should belong to the same meta-class while dissimilar classes should be apart. Therefore, in this study we adapted two approaches that exploit the similarity among classes: the bottom-up approach defined in Algorithm 25 and the top-down approach defined in Algorithm 26. The resultant binary tree has K leaf nodes, one for each original class and K-1 internal nodes, each associated with two (meta-)classes and a binary classifier.

In the bottom-up approach, the multi-view hierarchical clustering algorithm proposed by Gupta and Dasgupta [75] is followed. In the top-down approach, the tree structure is generated by recursively applying *k-means* clustering algorithm [123] at each node j to split its associated set of classes Ω_j into two disjoint subsets Ω_{2j} and Ω_{2j+1}, until every subset contains exactly one class. In this study, the distance between classes ω_i and ω_k is the Euclidean distance between the centroid of the training examples that belong to class ω_i and that of the examples belonging to class ω_k. To find the best view to split the set of classes, different splits are evaluated using an evaluation measure such as the SD validity index or impurity measures such as the Entropy or Gini index.

14.2.2.2 Train Binary Classifiers

In the second step, a binary classifier h_j is trained for each internal node j using the corresponding training instances L_j such as a support vector machine or a radial basis functions network.

14.3 Forest Classification Phase

Two different strategies to combine the decisions of tree-structured binary classifiers, defined in Section 4.6.2, have been used throughout this study : decision-tree-like combination and a combination scheme which is derived from the Dempster's rule of combination. These schemes are rather different in terms of complexity and output type (crisp vs. soft).

14.3.1 Evidence-theoretic Soft Combiner

Let x_u be a given example to be classified by a multi-view forest. Classifying x_u means assigning on of the classes in Ω to it. Using the vocabulary of D-S theory, Ω can be called the frame of discernment of the task where hypothesis θ_k means that "the given instance x_u belongs to class ω_k". In addition, each internal node classifier h_j is considered as a source of evidence providing that it is soft classifier ($h_j : \mathbb{R}^d \times \{\Omega_{2j}, \Omega_{2j+1}\} \rightarrow [0, 1]$). The final decision is a combination of knowledge extracted from different sources: (i) binary classifier, (ii) tree ensemble of K-1 binary classifiers and (iii) the forest ensemble of trees.

14.3.1.1 Evidence from an individual node classifier

Consider an internal node j within a tree, let us define a local frame of discernment Θ_j:

$$\Theta_j = \{\Theta_{2j}, \Theta_{2j+1}\} \tag{14.1}$$

where hypothesis Θ_{2j} means that "the given instance x_u belongs to meta-class Ω_{2j} and Θ_{2j+1} means that "it belongs to meta-class Ω_{2j+1}".

Since h_j is a source of evidence, it can be represented by a *BBA* m_j. Usually, not all classifiers produce outputs that satisfy the conditions of *BBA*:

$$m_j(\emptyset) = 0 \quad \text{and} \quad \sum_{A \subseteq \Theta} m_j(A) = 1. \tag{14.2}$$

In this case, the outputs of classifier h_j are transformed into *BBA* as follows: (1) all negative values are set to zero, (2) if the sum of a classifier outputs is greater than one, it is normalized to sum up to one. if $h_j(x_u, \Omega_{2j})$ ($h_j(x_u, \Omega_{2j+1})$) is high, a high belief is assigned to hypothesis Θ_{2j} (Θ_{2j+1}).

Discounting Technique is used to propagate the outputs of high-level classifiers to the classifiers at the lower levels. That is, the output of each internal node classifier h_j is multiplied by the *BBA* of its parent node classifier $m_{par(j)}$ where the root node classifier output is not discounted. The motivation for discounting is the fact that a number of classifiers will be enforced to classify examples that actually belong to classes that are unknown to them. For instance, a classifier h_j that discriminates between $\Omega_{2j} = \{\omega_1, \omega_5\}$ and $\Omega_{2j+1} = \{\omega_2, \omega_6\}$ has to classify an example x_u belonging to class ω_3. In this case, it is desirable that $h_j(x_u, \Omega_{2j})$ and $h_j(x_u, \Omega_{2j+1})$ tends to zero but at the real situation, for instance if h_j is a support vector machine (Section 2.4), either of them may tend to one. If at least one classifier within a certain path gives a low response to instance x_u, this leads to weaken any undesirable high responses. Therefore, *BBA* m_j is defined as follows:

$$m_j(\Theta_{2j}) = m_{par(j)}(A).h_j(x_u, \Omega_{2j}) \tag{14.3}$$

$$m_j(\Theta_{2j+1}) = m_{par(j)}(A).h_j(x_u, \Omega_{2j+1}) \tag{14.4}$$

$$m_j(\Theta) = 1 - m_j(\Theta_{2j}) - m_j(\Theta_{2j+1}) \tag{14.5}$$

$$m_j(B) = 0 \;\; \forall B \in 2^{\Theta} - \{\Theta, \Theta_{2j}, \Theta_{2j+1}\} \tag{14.6}$$

where $A = \Theta_{2.par(j)}$ if $j = 2.par(j)$ (node j lies at the left subtree of its parent node) and similarly $A = \Theta_{2.par(j)+1})$ if $j = 2.par(j) + 1$. Note that $m_j(\Theta)$ represents the doubt in h_j.

14.3.1.2 Evidence from all K-1 node classifiers within tree

Following *Dempster's unnormalized rule of combination*, the *BBA*s from the K-1 internal node classifiers within a class hierarchy t are conjunctively combined in order to calculate the evidence about a hypothesis θ_k (degree of belief provided by H_t that an example x_u belongs to ω_k).

$$\mu_k^{(t)}(x_u) = m^{(t)}(\theta_k) = \sum_{\cap A_j = \theta_k} \prod_{1 \leq j \leq K-1} m_j(A_j) \;\text{ where }\; A_j = \Theta_{2j},\; \Theta_{2j+1},\; or\; \Theta \tag{14.7}$$

and

$$m^{(t)}(\Theta) = \prod_{1 \leq j \leq K-1} m_j(\Theta) \tag{14.8}$$

where $m^{(t)}(\Theta)$ represent the conflict among the internal classifiers $h_1, \ldots, h_{K-1}$.

14.3.1.3 Evidence from all trees within a forest

Each single-view tree or multi-view tree H_t provides a mass distribution $m^{(t)}$ describing the beliefs in the membership of an example x_u to the K classes. These

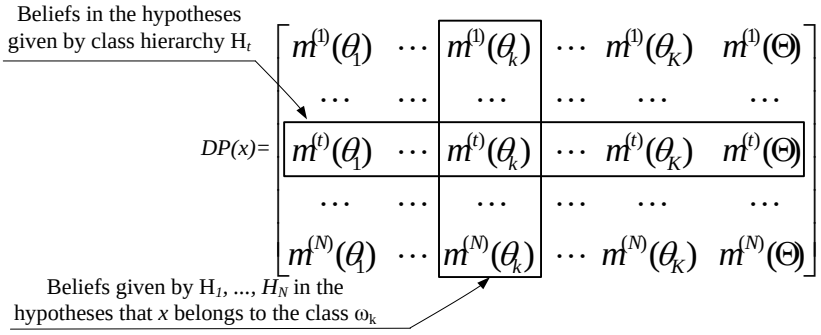


Figure 14.5: An illustration of evidence-theoretic decision profile

beliefs are stored in a matrix $DP(x)$ that is called *evidence-theoretic decision profile* (see Figure 14.5). Based on this profile, the overall support for each class ω_k can be obtained using either class-conscious or class-indifferent combination methods [107]. The class-conscious methods use only the k^{th} column of $DP(x)$ such as average, minimum, maximum and product rules. The class-indifferent methods ignore the context of $DP(x)$ and use all of $DP(x)$ as features in a new feature space, which is called the intermediate feature space. In the experiments, the class-conscious methods are used while the class-indifferent methods will be investigated in a future work.

14.4 Application to Visual Object Recognition

14.4.1 Results on the Fruits Data Set

The fruits data set defined in Section 7.1.1 (see Figure 7.2) was used for performance evaluation. Five different feature sets (views) were extracted: color histograms (fs_1), orientation histograms utilizing canny edge detection (fs_2), utilizing sobel edge detection (fs_3), utilizing opponent colors black and white (fs_4) and utilizing opponent colors red and green (fs_5) (see [60] for more details). The results are the average of 10 runs of 10-fold cross-validation (CV). First,we construct a tree classifier for each possible combination of views, leading to 31 classifiers for 5 feature sets. For the representation of feature sets, the binary string representation is chosen where each view is represented by N bits (N: number of features in the full set). Each bit represents the presence (1) or absence (0) of that feature set in the view set. For instance, if N=4, then string 1001 means that only fs_1, fs_4 are used to select best view at each node of the tree. The Bottom-Up approach defined in Algorithm 25 is used to build the class hierarchies. The radial basis function (RBF) networks with 16 RBF neurons were used as binary classifiers (see Algorithm 12 in Chapter 8).

Table 14.1 shows the results of using all the possible combinations of views

Table 14.1: Mean accuracy and standard deviation of the tree classifiers

Rank	View	DT	DS	Rank	View	DT	DS
1	[1 1 0 0 1]	96.86 ± 1.66	97.21 ± 1.55	17	[0 0 1 1 1]	96.14 ± 2.31	96.27 ± 2.32
2	[1 1 1 0 1]	96.56 ± 1.82	96.96 ± 1.72	18	[1 0 0 0 1]	96.04 ± 1.94	96.40 ± 1.80
3	[1 1 0 0 0]	96.50 ± 1.92	96.90 ± 1.71	19	[0 1 0 1 1]	95.36 ± 2.61	95.27 ± 2.66
4	[1 0 1 0 1]	96.46 ± 1.82	96.89 ± 1.77	20	[0 0 1 0 1]	95.26 ± 2.36	95.67 ± 2.31
5	[1 0 0 1 0]	96.46 ± 1.96	96.48 ± 2.02	21	[0 0 0 1 1]	95.24 ± 2.67	95.13 ± 2.69
6	[1 0 1 1 0]	96.46 ± 1.96	96.48 ± 2.02	22	[0 1 0 0 1]	94.60 ± 2.23	95.19 ± 2.20
7	[1 1 0 1 1]	96.45 ± 1.99	96.40 ± 2.08	23	[0 1 1 0 1]	94.35 ± 2.55	94.76 ± 2.48
8	[1 1 1 1 1]	96.45 ± 1.99	96.40 ± 2.08	24	[0 1 0 1 0]	94.23 ± 2.52	95.25 ± 2.51
9	[1 0 0 1 1]	96.45 ± 1.98	96.40 ± 2.07	25	[0 1 1 1 0]	94.19 ± 2.57	95.20 ± 2.55
10	[1 0 1 1 1]	96.45 ± 1.98	96.40 ± 2.07	26	[0 1 1 0 0]	92.21 ± 2.99	92.83 ± 3.05
11	**[1 0 0 0 0]**	**96.44 ± 2.03**	**96.69 ± 1.86**	**27**	**[0 1 0 0 0]**	**91.65 ± 2.70**	**92.21 ± 2.74**
12	[1 1 0 1 0]	96.44 ± 1.97	96.46 ± 2.03	28	[0 0 1 1 0]	90.2 ± 3.26	90.77 ± 3.00
13	[1 1 1 1 0]	96.44 ± 1.97	96.46 ± 2.03	**29**	**[0 0 1 0 0]**	**89.75 ± 4.08**	**90.39 ± 4.12**
14	[1 1 1 0 0]	96.25 ± 2.04	96.71 ± 1.82	**30**	**[0 0 0 0 1]**	**89.70 ± 3.24**	**89.82 ± 3.28**
15	[1 0 1 0 0]	96.18 ± 1.99	96.64 ± 1.80	**31**	**[0 0 0 1 0]**	**88.55 ± 3.38**	**88.87 ± 3.29**
16	[0 1 1 1 1]	96.15 ± 2.29	96.29 ± 2.30				

Table 14.2: Mean and Standard Deviation of CV Test Set Accuracy of *Multi-View Forest* consisting of the five Single-View Trees (in bold in Table 14.1)

TCM	FCM	MVF_{single}
DT	MV	98.6±1.35
DS	MV	98.8 ±1.29
	Min	98.6 ± 1.46
	Max	99.1 ± 0.98
	Mean	99.2 ± 0.89
	Prod	99.1 ± 1.15
Best Tree		96.6± 1.86
Gain		2.58%

in building tree classifiers. The first column shows the rank of the tree classifier of the sorted list. The second column contains the views used by the tree classifier, represented as a binary string indicating whether a view is in use or not. The third and the fourth column list the CV test set accuracy of each tree classifier for decision-tree-like (DT) and Dempster-Shafer-based (DS) combination, respectively. Table 14.2 illustrates the classification results of the ensembles constructed by combining the five single-view tree classifiers (MVF_{single}). The ensembles combine the outputs of tree classifiers using Majority Voting (MV), minimum (Min), maximum (Max), mean (Mean) and product (Prod) rules as forest combination methods (FCM), respectively. Table 14.3 illustrates the classification results of the ensemble constructed using the first, the middle and the last 10 tree classifiers in the sorted list, respectively (MVF_1, MVF_2, MVF_3).

First, the accuracies of the five Single-View Trees and an ensemble of them are compared. From Table 14.1, it can be seen that the tree classifier based only on fs_1 lies at rank 11, tree classifier based only on fs_2 lies at rank 27, tree classifier based only on fs_3 lies at rank 29, tree classifier based only on fs_5 lies at rank 30

Table 14.3: Mean and Standard Deviation of the *Multi-View Forest*s

TCM	*FCM*	MVF_1	MVF_2	MVF_3
DT	*MV*	96.46 ±1.98	97.27±1.66	97.93± 1.59
DS	MV	96.60± 1.94	97.38± 1.62	98.05± 1.55
	Min	**97.89± 1.39**	98.88± 1.19	97.93± 1.70
	Max	97.81± 1.45	**98.96 ± 1.02**	98.15± 1.39
	Mean	97.74± 1.45	98.75± 1.10	**98.77 ± 1.25**
	Prod	97.83± 1.45	98.89± 1.00	98.73± 1.31
Best Tree		97.21± 1.55	96.69± 1.86	95.24± 2.67
Gain		0.68%	2.27%	3.53%

and finally comes tree classifier based only on fs_4 lies at rank 31. This means that the tree classifier based on fs_1 outperforms all other single-feature-set classifiers by about 4.5%. From Table 14.2, the best ensemble has an accuracy of 99.2% ± 0.89. Therefore, the ensemble of the Single-View Trees outperforms the best single individual classifier. The reason of this performance is the large diversity between the classifiers as each of them use different feature set. Second, the results of the Single-View Trees and the *Multi-View Tree*s are compared. From Table 14.1, we can observe that the best Single-View tree classifier, based only on feature (fs_1), is at rank 11, thus 10 *Multi-View Tree* classifiers outperform the best single-view classifier. The tree classifier based on feature fs_1, fs_2 and fs_5, is at first rank, and achieves an accuracy of 96.8% ± 1.66 (DT) and 97.2% ± 1.55 (DS). So it outperforms the corresponding single view tree classifiers.

Third, we compare between *Multi-View Tree*s and Ensemble of *Multi-View Tree*s. From Table 14.3, we can see that the best ensemble, based on the 10 most accurate tree classifiers, achieves an accuracy of 97.8% ± 1.39 (DS + Min) while the best of the 10 trees has a rate 97.2% ± 1.55(DS). Therefore, there is a gain in accuracy only 0.68%. For the second ensemble, based on the second 10 tree classifiers in the list, the best result is 98.9% ± 1.02 (DS + Max). This means that the gain in accuracy is 2.2%. For the last ensemble, based on the following 10 tree classifiers in the list (weaker classifiers), the best rate is 98.7% ± 1.25 (DS + Mean) with a gain about 3.5%.

Finally, we compare among the three constructed ensembles. From Table 14.1, we can find that the ten classifiers of ensemble MVF_1 use fs_1 as best feature set in about 4 of their 6 binary classifiers while only 6 trees of the ten of MVF_2 use fs_1. Therefore, ensemble MVF_2 is more diverse than MVF_1 as it contains weaker and less identical tree classifiers. For this reason, ensemble MVF_2 has more gain than MVF_1 and ensemble MVF_3 gains more than MVF_2. The weaker and the diverse the combined individual classifiers are, the higher will be the gain in the ensemble accuracy. Although the ensemble MVF_3 is consisting of less accurate individual classifiers than that of MVF_1 and MVF_2, the observed gain of MVF_3

is higher than that of MVF_1 and MVF_2 and in many cases it outperforms MVF_1 and MVF_2.

14.4.2 Results on the Handwritten Digits

The performance was evaluated using the StatLog handwritten digits data set was defined in Section 7.1.2 (see Figure 7.4). Each example is represented by five feature types (views) described in Table 7.1: *image-vector*, *orienthisto*, *pca-40*, *rows-sum* and *cols-sum*. The Top-Down Approach defined in Algorithm 26 is used to build the class hierarchies. RBF networks have been used as binary classifiers such that the hidden layer consists of 20 RBFs per class (c=20) and the number of the input layer nodes equals to the dimension of the feature vector (see Algorithm 12 in Chapter 8). The results are the average of one run of 10-fold cross-validation (CV).

First, we construct a tree classifier for each possible combination of views. Table 14.4 illustrates the performance of the 5 single-view tree classifiers.

Table 14.4: Results of the five Single-View Tree Classifiers for the handwritten digits

TCM	*image-vector*	*orienthisto*	*pca-40*	*rows-sum*	*cols-sum*
DT	95.89±0.47	96.05±0.59	94.96±0.81	94.07±0.65	93.75±0.95
DS	**96.23±0.55**	**96.51±0.54**	**95.66±0.61**	**94.52±0.57**	**94.08±0.97**

Then, we construct three ensembles: MVF_{single} consists of the five Single-View tree classifiers (3rd column in Table 14.5), $MVF(31)$ based on the 31 constructed classifiers (4th column in Table 14.5) and $MVF(5)$ is constructed by removing similar classifiers from the forest $MVF(31)$ and keeping only the 5 most diverse classifiers using kappa agreement measure [124]. Note that if there are two identical tree classifiers, we select the tree that uses less number of views. The results show that MVF_{single} outperforms the best Single-View tree classifier and shows better performance than $MVF(31)$. In addition, we found that the top five diverse classifiers in $MVF(5)$ are the single-view ones. That is, for the digits data set the constructed multi-view trees are similar to the single-view ones. This results confirm our hypothesis that an ensemble of diverse tree classifiers outperform its individual members.

14.5 Conclusions

In this study, a new ensemble method, called *Multi-View Forest*, is proposed. It requires that the instances are represented by two or more sufficient and independent views. It constructs ensembles of multi-view tree-structured classifiers using different combination methods. As demonstrated by experiments, multi-view learning can improve the accuracy in complex pattern recognition problems

Table 14.5: Results of the three *Multi-View Forest*s for the digits

TCM	FCM	MVF_{single}	$MVF(31)$	$MVF(5)$
DT	MV	96.80±0.44	94.08±0.64	96.80±0.44
DS	MV	97.14±0.45	94.59±0.61	97.14±0.45
	Min	97.43±0.53	97.41±0.52	97.43±0.53
	Max	97.63±0.54	**97.62±0.51**	97.63±0.54
	$Mean$	97.64±0.57	95.69±0.63	97.64±0.57
	$Prod$	**97.71±0.47**	96.51±0.50	**97.71±0.47**
Best Tree		96.51±0.54	96.62±0.57	96.51±0.54

with a large number of classes. In addition, the trees generated by each individual feature set seem to complement each other by showing part of the discriminating information. This motivates the use of multiple feature sets to generate one consolidated tree, through multi-view hierarchical clustering or k-means clustering. Also the results show that the bottom-up approach constructs unbalanced trees compared to the top-down approach that results in more balanced trees. In order to construct forest ensembles not only by *majority voting* hard combiner but also by soft combiners such as minimum, maximum, mean, and product, evidence-theoretic combination method is adapted for combining the intermediate outputs of binary classifiers within each class hierarchy. The motivation of adapting this evidence-theoretic combiner is that it provides not only a crisp class label but also a class probabilities estimate of the given examples. Experiments show that the soft combination rules together with the evidence-theoretic approach outperform the *majority voting*.

Chapter 15

An Information Theoretic Perspective on Classifier Selection

15.1 Introduction

Ensemble learning has become a hot research topic during the last decade. Typically, ensemble methods comprise two phases: the construction of multiple individual classifiers and their combination. Recent work has considered an additional intermediate phase that deals with the reduction of the ensemble size prior to combination. This phase has several names in the literature such as ensemble pruning, selective ensemble, ensemble thinning and classifier selection, the last one of which is used within this chapter. Classifier selection is important for two reasons: classification accuracy and efficiency. An ensemble may consist not only of accurate classifiers, but also of classifiers with lower predictive accuracy. Pruning the poor-performing classifiers while maintaining a good diversity of the ensemble is typically considered as the main factor for an effective ensemble. The second reason is equally important, efficiency. Having a very large number of classifiers in an ensemble adds a lot of computational overhead. For example, decision tree classifiers may have large memory requirements and lazy learning methods have a considerable computational cost during classification phase. The minimization of classification time complexity is crucial in certain applications, such as stream mining.

Recently an information-theoretic view was presented for feature selection. It derives a space of possible selection criteria and show that several feature selection criteria in the literature are points within this continuous space. The contribution of this paper is to export this information-theoretic view to solve an open issue in ensemble learning which is classifier selection. I investigated a couple of information-theoretic selection criteria that are used to rank individual classifiers. The work in this chapter has been previously published ([10]).

15.2 Entropy and Mutual Information

The building block of information theory is the entropy of a random variable. The *entropy* of a random variable X, denoted as $H(X)$, is a measure of the uncertainty on X and represents the amount of information provided by X. It is written as

$$H(X) = -\sum_{x_j \in X} p(x_j) \log p(x_j) \tag{15.1}$$

where a discrete random variable X has possible values $\{x_1, ..., x_m\}$, the base of the logarithm has a common value 2 (in this case, the unit of entropy is bit) and p indicates the probability mass function of X. That is, $p(X = x_j)$ gives the probability that X is exactly equal to some value x_j, the number of examples taking on value x_j divided by the total number of examples M. Like *probability theory*, entropy can be conditional on other random variable Y, this denotes the amount of information or uncertainty still remaining in X if the value of Y is known. The *conditional entropy* of X given Y is defined as,

$$H(X|Y) = -\sum_{y \in Y} p(y) \sum_{x \in X} p(x|y) \log p(x|y) \tag{15.2}$$

Shannon *Mutual Information* between X_1 and X_2 measures the amount of information shared between the two random variables and is defined as follows.

$$\begin{aligned} I(X_1; X_2) &= H(X_1) - H(X_1|X_2) = H(X_2) - H(X_2|X_1) \\ &= \sum_{x_1 \in X_1} \sum_{x_2 \in X_2} p(x_1, x_2) \log \frac{p(x_1, x_2)}{p(x_1)p(x_2)} \end{aligned} \tag{15.3}$$

The Mutual Information can also be conditioned on other random variable Y, the *conditional mutual information* is,

$$\begin{aligned} I(X_1; X_2|Y) &= H(X_1|Y) - H(X_1|X_2Y) \\ &= \sum_{y \in Y} p(y) \sum_{x_1 \in X_1} \sum_{x_2 \in X_2} p(x_1, x_2|y) \log \frac{p(x_1, x_2|y)}{p(x_1|y)p(x_2|y)} \end{aligned} \tag{15.4}$$

which can be simplified as follows,

$$I(X_1; X_2|Y) = \sum_{y \in Y} \sum_{x_1 \in X_1} \sum_{x_2 \in X_2} p(x_1, x_2, y) \log \frac{p(y)p(x_1, x_2, y)}{p(x_1, y)p(x_2, y)} \tag{15.5}$$

This measures the amount of information shared between X_1 and X_2 when Y is known. The relation between all these quantities can be seen in Figure 15.1.

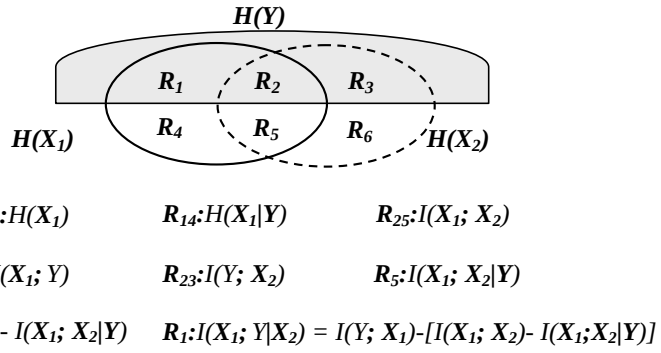


Figure 15.1: Graphical illustration of entropy, conditional entropy, mutual information and conditional mutual information

15.3 Information Theoretic Classifier Selection

If a message Y was sent through a communication channel, and a value X is received, then a decoding operation, $\hat{Y} = g(X)$, is performed to decode X and recover the correct Y. In *ensemble learning* (Chapter 3) terms: Y is the original (unknown) class label distribution, $X_{1:N}$ is the joint variable of all the classifiers trained to solve a classification task, and g is an ensemble combination function. The set of trained classifiers may or may not be sufficient to perfectly recover Y; that is, there may be a classification error. *Information theory* can provide a bound on $p(\hat{Y} \neq Y)$, for any combiner g. The error of predicting target variable Y from input $X_{1:N}$ is bounded by two inequalities [35] as follows,

$$\frac{H(Y) - I(X_{1:N}; Y) - 1}{log(|Y|)} \leq p(\hat{Y} \neq Y) \leq \frac{1}{2} H(Y|X_{1:N}). \tag{15.6}$$

Note that Fano's inequality provides the lower bound on the Bayes error and Hellman-Raviv provides its upper bound. The bound should be minimized in order to minimize the probability that the combiner g can not predict Y which is equivalent to maximizing the joint mutual information $I(X_{1:N}; Y)$ defined in Eq. (15.3). Unfortunately, $I(X_{1:N}; Y)$ involves high dimensional joint probability mass functions $p(x_1, \ldots, x_N)$ and $p(x_1, \ldots, x_N, y)$ that are hard to be estimated as explained in [145]. For instance, suppose that each classifier discriminates among ten classes using m training examples. The N ensemble classifiers could have a maximum $min(10^N; m)$ joint states. When the number of joint states increases very quickly and gets comparable to the number of examples, m, the joint probability of these classifiers, as well as the mutual information, cannot be estimated correctly. Another drawback of directly calculating $I(X_{1:N}; Y)$ is the slow computational speed. In the following subsections, we show how it can be decomposed into simpler terms.

15.3.1 Interaction Information

Shannon's Mutual Information $I(X_1; X_2)$ is a function of two variables. It is not able to measure properties of multiple (N) variables. McGill [126] presented what is called *Interaction Information* as a multi-variate generalization for Shannon's Mutual Information. For instance, the *Interaction Information* between three random variables is

$$I(\{X_1, X_2, X_3\}) = I(X_1; X_2|X_3) - I(X_1; X_2) \tag{15.7}$$

That is, the difference of the conditional mutual information, defined in Eq. (15.5) and the simple Shannon mutual information, defined in Eq. (15.3). The general form for X where $|X| \geq 2$ is defined recursively.

$$I(X \cup \{Y\}) = I(X|Y) - I(X) \tag{15.8}$$

For a full treatment of this topic, the reader is directed to [35].

15.3.2 Mutual Information Decomposition

15.3.1. THEOREM. *Given a set of classifiers $S = \{X_1, \ldots, X_N\}$ and a target class label Y, the Shannon mutual information between $X_{1:N}$ and Y can be decomposed into a sum of Interaction Information terms,*

$$I(X_{1:N}; Y) = \sum_{X \subseteq S, |X| \geq 1} I(X \cup \{Y\}). \tag{15.9}$$

Proof See [35]

For a set of classifiers $S = \{X_1, X_2, X_3\}$, the mutual information between the joint variable $X_{1:3}$ and a target Y can be decomposed as

$$\begin{aligned} I(X_{1:3}; Y) = {} & I(X_1; Y) + I(X_2; Y) + I(X_3; Y) \\ & + I(\{X_1, X_2, Y\}) + I(\{X_1, X_3, Y\}) + I(\{X_1, X_3, Y\}) \\ & + I(\{X_1, X_2, X_3, Y\}) \end{aligned} \tag{15.10}$$

Each term can then be decomposed into class unconditional $I(X)$ and conditional $I(X|Y)$ according to Eq. (15.7).

$$\begin{aligned} I(X_{1:3}; Y) = {} & \sum_{i=1}^{3} I(X_i; Y) - \sum_{\substack{X \subseteq S \\ |X|=2}} I(X) + \sum_{\substack{X \subseteq S \\ |X|=2}} I(X|Y) \\ & - I(\{X_1, X_2, X_3\}) + I(\{X_1, X_2, X_3\}|Y) \end{aligned} \tag{15.11}$$

For an ensemble S of size N and according to Eq. (15.8),

$$I(X_{1:N};Y) = \sum_{i=1}^{N} I(X_i;Y) - \sum_{\substack{X\subseteq S \\ |X|=2..N}} I(X) + \sum_{\substack{X\subseteq S \\ |X|=2..N}} I(X|Y) \tag{15.12}$$

Ensemble mutual information $I(X_{1:N};Y)$ is decomposed into three terms. The first term, $\sum_{i=1}^{N} I(X_i;Y)$ is the sum of mutual information between each individual classifier and the target where $I(X_i;Y)$ is called the *relevance* of the i^{th} classifier output to the target class label. The second contains terms of the form $I(X)$ and is independent of the class label Y , and so is analogous to the concept of *diversity*. It measures the interaction information among all possible subsets of classifiers, drawn from the ensemble. This is called the ensemble *redundancy*. Note that this term is subtractive from the overall mutual information. A large value of $I(X)$ indicates strong correlations among the classifiers, and reduces the value of $I(X_{1:N};Y)$, and hence the overall achievable accuracy. The third contains terms of the form $I(X|Y)$ and is a function of the class label Y. It is called *conditional redundancy*. Note that this term is additive to the ensemble mutual information. This term indicates that an effective ensemble requires high class-conditional correlations while it has low correlations among its individual members. The decomposition equation in Eq. (15.12) shows that *diversity* exists at multiple levels of correlation within an ensemble. If the classifiers are statistically independent, then *redundancy* and *conditional redundancy* would be zero and $I(X_{1:N};Y) = \sum_{i=1}^{N} I(X_i;Y)$. If the classifiers have only pairwise interactions, then the second-order and higher *redundancy* and *conditional redundancy* terms should be omitted from the decomposition equation. This assumption of pairwise interactions gives us,

$$I(X_{1:N};Y) \simeq \sum_{i=1}^{N} I(X_i;Y) - \sum_{i=1}^{N-1}\sum_{j=i+1}^{N} I(X_i;X_j) + \sum_{i=1}^{N-1}\sum_{j=i+1}^{N} I(X_i;X_j|Y) \tag{15.13}$$

15.4 Classifier Selection Criteria

The objective of an information-theoretic classifier selection method, see Algorithm 27, is to select a subset of K classifiers (S) from a pool of N classifiers (Ω) that carries as much information as possible about the target class Y using a predefined selection criterion,

$$\begin{aligned} J(X_{u(j)}) &= I(X_{1:k+1};Y) - I(X_{1:k};Y) \\ &= I(X_{u(j)};Y) - \sum_{i=1}^{k} I(X_{u(j)};X_{v(i)}) + \sum_{i=1}^{k} I(X_{u(j)};X_{v(i)}|Y) \end{aligned} \tag{15.14}$$

Algorithm 27 Pseudo Code of Classifier Selection

Require: set of classifiers ($\Omega = \{X_1, \ldots, X_N\}$), target class labels Y, number of required classifiers ($K < N$), redundancy parameter (β), class-conditional redundancy parameter (γ)

```
Select the most relevant classifier, v(1) = arg max_{1≤j≤N} I(X_j; Y)
S = {X_{v(1)}}
for k = 1 : K − 1 do
    for j = 1 : |Ω \ S| do
        Calculate J(X_{u(j)}) as defined in Eq. (15.15)
    end for
    v(k + 1) = arg max_{1≤j≤|Ω\S|} J(X_{u(j)})
    S = S ∪ {X_{v(k+1)}}
end for
```

That is the difference in information, after and before the addition of $X_{u(j)}$ into S. This tells us that the best classifier is a trade-off between these components: the relevance of the classifier, the unconditional correlations, and the class-conditional correlations. In order to balance between these components, Brown [35] has parameterized Eq. (15.14) and defined the root criterion,

$$J(X_{u(j)}) = I(X_{u(j)}; Y) - \beta \sum_{i=1}^{k} I(X_{u(j)}; X_{v(i)}) + \gamma \sum_{i=1}^{k} I(X_{u(j)}; X_{v(i)}|Y). \quad (15.15)$$

Brown [35] presented a unifying viewpoint on the existing information theoretic feature ranking literature. He showed how several published heuristics [21, 145, 205, 121, 63] can all be rearranged into a common functional form, such that they can be reproduced by parameterizations of the root criterion in Eq. (15.15). Consequently, they all fit neatly into a unit square, illustrated in Figure 15.2. The remaining of this section shows the criteria that will be exported from the context of feature selection and applied for classifier selection in the experimental part of this chapter.

15.4.1 Maximal relevance (MR)

As a heuristic, we could assume the prediction of each classifier X_i is independent of all other classifiers and rank the classifiers in descending order based on the criterion

$$J(X_{u(j)}) = I(X_{u(j)}; Y). \quad (15.16)$$

However this is known to be suboptimal since the classifier predictions are often interdependent (see R_{12} in Figure 15.1).

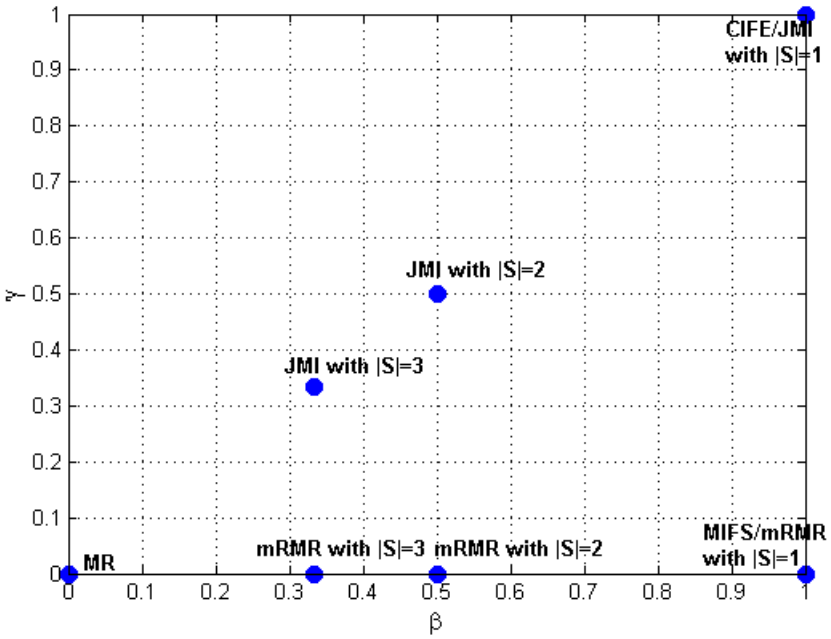


Figure 15.2: The full space of first-order classifier selection criteria, derived from Eq. (15.15) after omitting the second-order and higher *redundancy* and *conditional redundancy* terms.

15.4.2 Mutual Information Feature Selection (MIFS)

Battiti [21] proposed the *Mutual Information Feature Selection* criterion,

$$J(X_{u(j)}) = I(X_{u(j)}; Y) - \sum_{i=1}^{k} I(X_{u(j)}; X_{v(i)}). \tag{15.17}$$

The MIFS scheme shows a clear link to Eq. (15.14) as it includes *relevance* and unconditional *redundancy* terms but omits the conditional term.

15.4.3 Minimal Redundancy Maximal Relevance (mRMR)

Peng et al. [145] introduced the *Minimal Redundancy Maximal Relevance* criterion,

$$J(X_{u(j)}) = I(X_{u(j)}; Y) - \frac{1}{|S|} \sum_{i=1}^{k} I(X_{u(j)}; X_{v(i)}). \tag{15.18}$$

They proved theoretically that the combination of maximizing *relevance* and minimizing unconditional *redundancy* criteria is equivalent to maximizing the joint mutual information $I(X_{1:N}; Y)$ if one feature is selected (added) at one time. It is clear that *mRMR* is equivalent to MIFS with $\beta = \frac{1}{|S|}$. That is, it takes the

average of the unconditional *redundancy* terms, but again omits the conditional term.

15.4.4 Joint Mutual Information (JMI)

Yang and Moody [205] proposed using *Joint Mutual Information*,

$$J(X_{u(j)}) = \sum_{i=1}^{k} I(X_{u(j)}X_{v(i)}; Y). \tag{15.19}$$

This is the information between the target class Y and a joint random variable, defined by pairing the candidate classifier $X_{u(j)}$ with each classifier $X_{v(i)}$ already picked in S. This can be re-written as,

$$J(X_{u(j)}) = I(X_{u(j)}; Y) - \frac{1}{|S|}\sum_{i=1}^{k}\left[I(X_{u(j)}; X_{v(i)}) - I(X_{u(j)}; X_{v(i)}|Y)\right]. \tag{15.20}$$

Intermediate steps for this re-writing are given in [35]. JMI captures the conditional redundancy, but takes the mean value. It is clear that the JMI criterion is the MRMR criterion plus the *conditional redundancy* term.

15.4.5 Conditional Infomax Feature Extraction (CIFE)

Lin and Tang [121] introduced a criterion, called *Conditional Infomax Feature Extraction*, which maximizes the joint class-relevant information by explicitly reducing the class-relevant redundancies among classifiers.

$$J(X_{u(j)}) = I(X_{u(j)}; Y) - \sum_{i=1}^{k}\left[I(X_{u(j)}; X_{v(i)}) - I(X_{u(j)}; X_{v(i)}|Y)\right]. \tag{15.21}$$

15.4.6 Conditional Mutual Information Maximization (CMIM)

Fleuret [63] proposed the criterion based on *Conditional Mutual Information Maximization*,

$$J(X_{u(j)}) = \min_{1\le i\le k} I(X_{u(j)}; Y|X_{v(i)}). \tag{15.22}$$

which can be re-written as,

$$J(X_{u(j)}) = I(X_{u(j)}; Y) - \max_{1\le i\le k}[I(X_{u(j)}; X_{v(i)}) - I(X_{u(j)}; X_{v(i)}|Y)] \tag{15.23}$$

The proof is again available in [35] (see R_1 in Figure 15.1). CMIM examines the information between a candidate classifier $X_{u(j)}$ and the target class Y, conditioned on each classifier $X_{v(i)}$ already in S. This means that $X_{u(j)}$ is good only if

it provides information about Y, and this information has not been provided by any of the classifiers $\{X_{v(i)}\}_{i=1}^{k}$ already picked. That is, the score $J(X_{u(j)}, S)$ is low if at least one of the classifiers already picked is similar to $X_{u(j)}$ (or if $X_{u(j)}$ does not provide information about Y).

15.5 Related Work

Tsoumakas et al. [186] categorize the state-of-the-art classifier selection methods into a taxonomy. They propose to organize them into the four categories: a) Search-based, b) Clustering-based, c) Ranking-based and d) Other. They further divide the first category into two subcategories, based on the search paradigm: a) greedy search, and b) stochastic search. The greedy search based methods attempt to find the globally best subset of classifiers and use different directions for searching the space of all possible classifier subsets such as forward selection and backward elimination.

Gasen-b [216] performs stochastic search in the space of model subsets using a standard genetic algorithm. The ensemble is represented as a bit string, using one bit for each model. A classifier is included or excluded from the ensemble based on the value of its corresponding bit. The generalization error of the ensemble is used as a function for evaluating the fitness of individuals in the population. The authors experimented with bagged ensembles and avoided using datasets with less than 1000 examples. They conclude that pruning not only reduce the complexity of the ensemble but also improve its generalization ability.

Margineantu and Dietterich [124] introduce heuristics to calculate the benefit of adding a classifier to an ensemble, using forward selection in a number of them. These heuristics depend on the diversity and the accuracy of the classifiers. The authors experiment with boosting ensembles and conclude that pruning help to reduce the ensemble complexity but it may sacrifice its generalization ability.

Meynet and Thiran [129] suggest a heuristic cost function, designed to compromise between ensemble accuracy and diversity. The selection criterion consists of two information theoretic terms. The first is simply the average mutual information between each ensemble member and the class label, which they call the Information Theoretic Accuracy, $ITA = \frac{1}{N}\sum_{i=1}^{N} I(X_i;Y)$. The second is the inverse of the average pairwise mutual information between ensemble members, which they call the Information Theoretic Diversity,

$$ITD = \left(\frac{2}{M(M-1)} \sum_{i=1}^{N-1} \sum_{j=i}^{N} I(X_i;X_j) \right)^{-1} . \tag{15.24}$$

The aim is to simultaneously maximize ITA and ITD although there is a trade-off between them. The authors represent the trade-off between the terms by a

second-order polynomial: the Information Theoretic Score is defined

$$ITS = (1 + ITA)^3.(1 + ITD) \tag{15.25}$$

Comparing this heuristic to Brown's work in [35] that is used in this chapter, it is clear that ITS ignores the class-conditional redundancy term $I(X_i; X_j|Y)$ and all the higher-order terms. The main difference between this heuristic and Brown's work is that the former was hand-designed, while the latter has shown a natural derivation of diversity at multiple levels of correlation within an ensemble as defined in Eq. (15.12).

15.6 Experimental Evaluation

15.6.1 Methodology

The effectiveness of the six selection criteria on classifier selection is evaluated on 11 data sets from the UCI machine learning repository [27] (see Table 15.1). Each experiment is conducted twice: one using *Bagging* [31] (Section 3.4.1.1) to construct an ensemble of N=50 *C4.5* decision trees (Section 2.3) and another time using *Random Forest* [32] (Section 3.4.2.2) to construct an ensemble of N=50 *random* trees. Each selection criterion is evaluated with the target number of classifiers K=40, 30, 20 and 10. This corresponds to 20%, 40%, 60% and 80% pruning percentage. Each test accuracy percentage reported is the average of performing 5 runs of 10-fold cross-validation. The training sets of the decision trees used to constitute the ensembles are bootstrap samples from the training set of each fold. As well the validation set used by each selection criterion for mutual information measurement is a bootstrap sample from the training set of each fold. For any selection criterion, the *normalized test accuracy* is defined to

Table 15.1: Description of the 11 data sets used in this study

id	name	Classes	Examples	Features	
				Discrete	Continuous
d_1	*anneal*	6	898	32	6
d_2	*autos*	7	205	10	16
d_3	*wisconsin-breast*	2	699	0	9
d_4	*bupa liver disorders*	2	345	0	6
d_5	*german-credit*	2	1000	13	7
d_6	*pima-diabetes*	2	768	0	8
d_7	*glass*	7	214	0	9
d_8	*cleveland-heart*	2	303	7	6
d_9	*hepatitis*	2	155	13	6
d_{10}	*ionosphere*	2	351	0	34
d_{11}	*vehicle*	4	846	0	18

be the difference between the accuracy of ensemble pruned by this criterion and single tree divided by the difference between the unpruned ensemble and single tree ($normalized\_test\_acc = \frac{pruned\_ens\_acc - single\_tree\_acc}{unpruned\_ens\_acc - single\_tree\_acc}$). Note that in all of the

11 data sets, *unpruned_ens_acc* > *single_tree_acc*. Hence, a *normalized test accuracy* of 1.0 indicates that the pruned ensemble construed by a given selection criterion obtains the same performance as the unpruned ensemble. A *normalized test accuracy* of zero indicates that the performance of the pruned ensemble is the same as a single tree classifier.

15.6.2 Results

Table 15.2 and Table 15.3 show the test accuracy under 80% pruning percentage for *Bagging* and *Random Forest*, respectively (the tables for other pruning percentages are dropped as they show the same behaviour). The statistical test for the comparison between different algorithms is the *corrected paired t-test* at 0.05 significance level. The mark ($^\bullet$) means that the corresponding pruned ensemble is significantly better than the single tree, while the mark ($^\diamond$) means that it is significantly worse than the unpruned ensemble. Although as many as 80% of the classifiers is pruned, the pruned ensemble still gives accuracy comparable to the unpruned one in all datasets, except for *german-credit* where the ensemble pruned by MR, JMI and CMIM is significantly worse than the unpruned one in case of *Bagging* and the one pruned by MIFS and CIFE significantly underperform the unpruned one in case of *Random Forest*.

Table 15.2: Test accuracy for single *C4.5* decision tree, ensemble constructed using *Bagging* before pruning and after pruning by the 6 selection criteria under 80% pruning percentage

id	*C4.5*	*Bagging*	*MR*	*MIFS*	*mRMR*	*JMI*	*CIFE*	*CMIM*
d_1	98.5	98.7	99.0	98.8	**99.2**	99.2	98.7	99.0
d_2	82.3	84.5	84.3	83.3	85.1	**85.7**	82.5	84.5
d_3	95.0	96.1$^\bullet$	96.0$^\bullet$	96.1	**96.2**	96.1	96.1	96.0
d_4	66.4	73.1$^\bullet$	70.7	70.7	69.8	**70.7**	70.4	70.3
d_5	71.3	74.7$^\bullet$	72.2$^\diamond$	73.6	73.2	72.6$^\diamond$	**73.9**$^\bullet$	72.5$^\diamond$
d_6	74.9	75.8	75.2	75.3	75.2	75.3	**75.8**	75.3
d_7	69.0	73.3	73.0	**73.2**	73.1	71.7	71.8	72.6
d_8	77.0	79.9	78.9	**81.0**	78.8	79.2	81.0	79.8
d_9	79.7	81.4	80.7	80.3	81.3	**81.7**	80.8	81.3
d_{10}	89.6	92.3	92.5	92.4	**92.8**	92.7	92.3	92.7
d_{11}	71.9	75.0$^\bullet$	75.2	74.7	**75.3**$^\bullet$	75.1	74.3	74.5

An appropriate way [52] to compare two or more algorithms on multiple datasets depends on their average rank across all datasets. For each dataset, the algorithm with the highest accuracy gets rank 1.0, the one with the second highest accuracy gets rank 2.0 and so on. In case two or more algorithms tie, they all receive the average of the ranks that correspond to them. Table 15.4 and Table 15.5 show the rank of each criterion on each dataset and the average ranks under 80% pruning percentage for *Bagging* and *Random Forest*, respectively. The tables for other pruning percentages are dropped as they show the same behaviour. On average and using *Bagging*, the best criterion is JMI (2.77), followed by mRMR (2.82), MIFS (3.50), CMIM (3.68),CIFE (3.95) and MR (4.27). On average and

Table 15.3: Test accuracy for single Random Tree (RT), ensemble constructed using *Random Forest* (RF) before pruning and after pruning by the 6 selection criteria under 80% pruning percentage

id	*RT*	*RF*	*MR*	*MIFS*	*mRMR*	*JMI*	*CIFE*	*CMIM*
d_1	98.1	99.6$^{\bullet}$	**99.6**$^{\bullet}$	99.3$^{\bullet}$	**99.6**$^{\bullet}$	99.5$^{\bullet}$	99.4$^{\bullet}$	99.4$^{\bullet}$
d_2	75.9	84.4$^{\bullet}$	**83.7**$^{\bullet}$	82.2	82.7$^{\bullet}$	83.4$^{\bullet}$	82.3$^{\bullet}$	83.6$^{\bullet}$
d_3	94.1	96.5$^{\bullet}$	96.1$^{\bullet}$	96.0$^{\bullet}$	**96.1**$^{\bullet}$	96.0$^{\bullet}$	96.1$^{\bullet}$	95.9$^{\bullet}$
d_4	65.2	71.7$^{\bullet}$	69.0	68.8	**69.4**	68.5	68.7	68.4
d_5	66.6	75.8$^{\bullet}$	73.7$^{\bullet}$	72.8$^{\bullet\diamond}$	74.0$^{\bullet}$	**74.8**$^{\bullet}$	73.0$^{\bullet\diamond}$	73.9
d_6	70.3	76.3$^{\bullet}$	**74.6**$^{\bullet}$	74.0$^{\bullet}$	74.5$^{\bullet}$	74.4$^{\bullet}$	74.3$^{\bullet}$	**74.6**$^{\bullet}$
d_7	69.7	78.3$^{\bullet}$	76.2	76.1	76.4	76.3	75.8	**76.7**$^{\bullet}$
d_8	76.0	81.8$^{\bullet}$	**81.1**	80.6	80.3	79.8	80.5	80.3
d_9	78.0	83.6	82.2	82.2	**83.0**	82.1	81.9	82.5
d_{10}	88.4	93.5$^{\bullet}$	92.8$^{\bullet}$	92.4	**93.5**$^{\bullet}$	93.3$^{\bullet}$	92.8$^{\bullet}$	93.2$^{\bullet}$
d_{11}	70.6	75.8$^{\bullet}$	75.4$^{\bullet}$	74.9$^{\bullet}$	75.1$^{\bullet}$	75.2$^{\bullet}$	74.9$^{\bullet}$	**76.5**$^{\bullet}$

using *Random Forest*, the best criterion is mRMR (2.23), followed by MR (2.41), CMIM (3.23), JMI (3.64), CIFE (4.59) and MIFS (4.91).

Table 15.4: Corresponding rank for different selection criteria using *Bagging* under 80% pruning percentage

id	*MR*	*MIFS*	*mRMR*	*JMI*	*CIFE*	*CMIM*
d_1	4.00	5.00	1.00	2.00	6.00	3.00
d_2	4.00	5.00	2.00	1.00	6.00	3.00
d_3	5.50	4.00	1.00	2.50	2.50	5.50
d_4	3.00	2.00	6.00	1.00	4.00	5.00
d_5	6.00	2.00	3.00	4.00	1.00	5.00
d_6	5.50	3.50	5.50	3.50	1.00	2.00
d_7	3.00	1.00	2.00	6.00	5.00	4.00
d_8	5.00	1.00	6.00	4.00	2.00	3.00
d_9	5.00	6.00	2.50	1.00	4.00	2.50
d_{10}	4.00	5.00	1.00	2.50	6.00	2.50
d_{11}	2.00	4.00	1.00	3.00	6.00	5.00
Av. Rank	4.27	3.50	2.82	**2.77**	3.95	3.68

Table 15.5: Corresponding rank for different selection criteria using *Random Forest* under 80% pruning percentage

id	*MR*	*MIFS*	*mRMR*	*JMI*	*CIFE*	*CMIM*
d_1	1.50	6.00	1.50	3.00	4.00	5.00
d_2	1.00	6.00	4.00	3.00	5.00	2.00
d_3	2.50	4.00	1.00	5.00	2.50	6.00
d_4	2.00	3.00	1.00	5.00	4.00	6.00
d_5	4.00	6.00	2.00	1.00	5.00	3.00
d_6	1.50	6.00	3.00	4.00	5.00	1.50
d_7	4.00	5.00	2.00	3.00	6.00	1.00
d_8	1.00	2.00	4.00	6.00	3.00	5.00
d_9	3.00	4.00	1.00	5.00	6.00	2.00
d_{10}	4.00	6.00	1.00	2.00	5.00	3.00
d_{11}	2.00	6.00	4.00	3.00	5.00	1.00
Av. Rank	2.41	4.91	**2.23**	3.64	4.59	3.23

Figure 15.3 and Figure 15.4 show the *normalized test accuracy* of each selection criterion for each dataset and the average over the 11 datasets under different

pruning percentages. A *normalized test accuracy* greater than 1.0 indicates that the pruned ensemble outperform its corresponding unpruned ensemble.

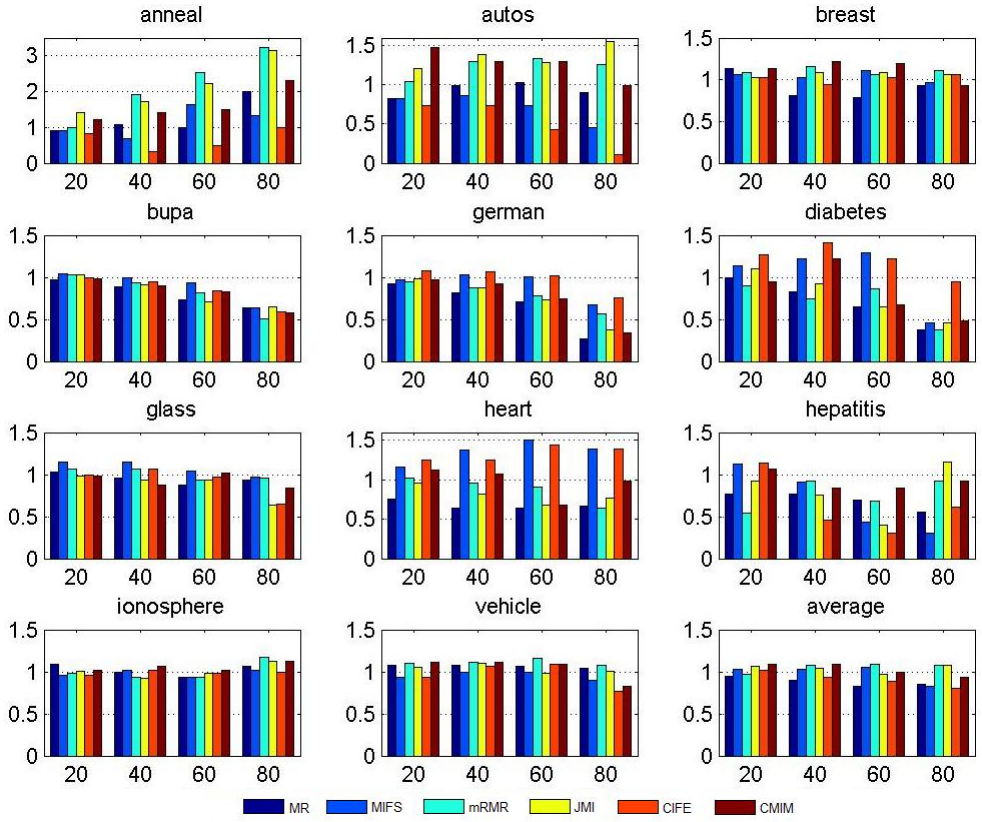


Figure 15.3: Comparison of the normalized test accuracy of the ensemble of *C4.5 decision trees* constructed by *Bagging* and pruned using: *MR*, *MIFS*, *mRMR*, *JMI*, *CIFE* and *CMIM* on 11 classification tasks (x-axis = percentage of pruning, y-axis = normalized test accuracy)

15.7 Conclusion and Future Work

This chapter examined the issue of classifier selection from an information theoretic viewpoint. The main advantage of information theoretic criteria is that they capture higher order statistics of the data. The ensemble mutual information is decomposed into accuracy and diversity components. Although diversity was represented by low and high order terms, we keep only the first-order terms in this chapter. There are many interesting directions for future work.

1. In further study, we will study the influence of including the higher-order terms on pruning performance.

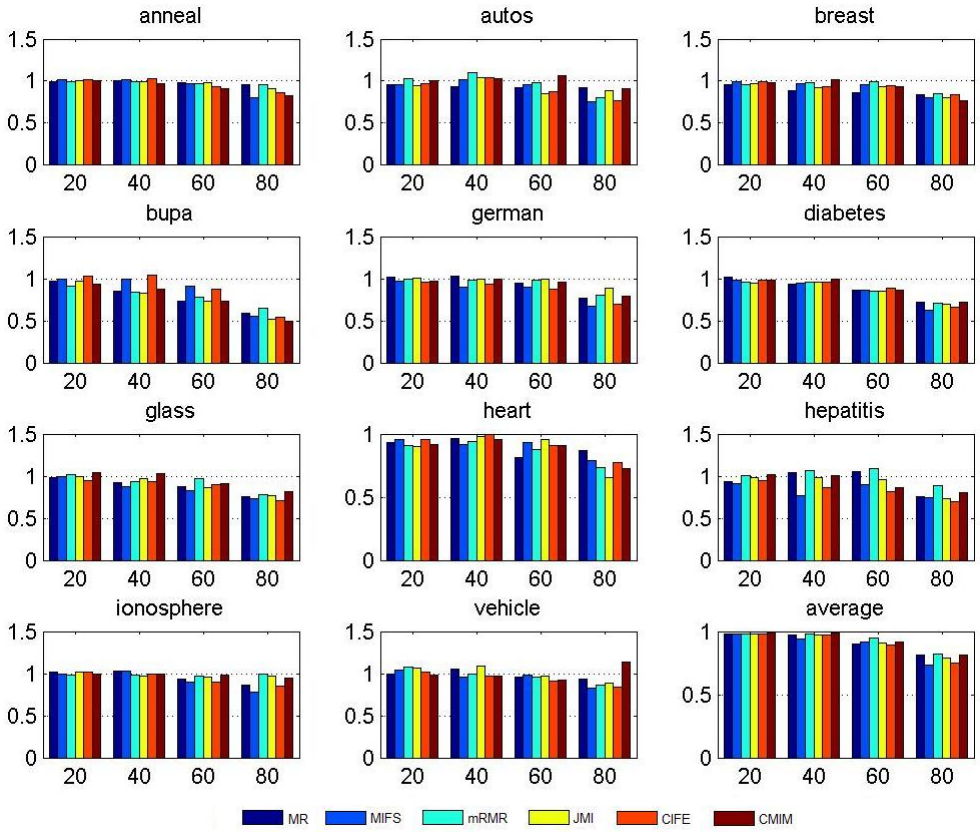


Figure 15.4: Comparison of the normalized test accuracy of the ensemble of *random trees* constructed by *Random Forest* and pruned using: *MR*, *MIFS*, *mRMR*, *JMI*, *CIFE* and *CMIM* on 11 classification tasks (x-axis = percentage of pruning, y-axis = normalized test accuracy)

2. Although Brown [35] derives a space of possible selection criteria, we select some points within this continuous space, that represent well-known feature selection criteria in the literature, such as mRMR, CIFE, JMI and CMIM, and use them for classifier selection. In a future work, we will explore other points in this space that may lead to more effective pruning.

3. In chapter 9, a single view version of *Co-Training*, called *CoBC*, is introduced. This algorithm comprises two phases executed iteratively: training an ensemble of individual classifiers and the prediction of the class labels of unlabeled data. It is clear that the computational complexity of *CoBC* is linear with respect to the number of ensemble members. Future work should consider an additional intermediate "classifier selection" phase that deals with the reduction of the ensemble size in order to reduce the complexity of *CoBC*.

Chapter 16

Conclusion

In todays information-rich digital world, supervised machine learning algorithms are used successfully to solve real-world challenges in extracting knowledge from large data sources. However, they require a large amount of labeled training data which is often tedious, difficult, time-consuming, or expensive to obtain. This thesis has focused on effective semi-supervised learning approaches for such problem domains. The methods introduced in this thesis aim to reduce the overall cost of acquiring labeled data by allowing the underlying predictors to effectively select and label the instances on which they are trained. In this chapter, I summarize the specific contributions of this work, and discuss several open research directions aimed at better exploiting the available data and labeling resources for machine learning problems through semi-supervised learning.

16.1 Main Contributions

This thesis has made several contributions to the state of the art in semi-supervised learning. These contributions answer the research questions mentioned in Chapter 1. Specific contributions include:

- *Semi-supervised learning with class hierarchies.* One contribution of this thesis is two novel approaches in Chapter 8 for semi-supervised learning in multiple view learning domains. The approaches take in account the similarities among classes that are represented as class hierarchies. In the first approach, a tree-structured ensemble of binary RBF networks is trained on each given view. Then, using *Co-Training* the most confident unlabeled examples labeled by each tree ensemble classifier are added to the training set of the other tree classifier; we call this scheme *cotrain-of-trees*. This approach provide a positive answer to the question: "Can the Dempster-Shafer evidence-theoretic combiner be appropriate for confidence measure?". In the second approach, first the given K-class problem is decomposed into

K-1 simpler binary problems using the tree-structured approach. Then using *Co-Training* a binary RBF network is trained on each given view to solve each binary problem; we call this last scheme *tree-of-cotrains*. In order to combine the intermediate results of the internal nodes within each tree, a combination method based on Dempster-Shafer evidence theory is used. Both *cotrain-of-trees* and *tree-of-cotrains* were evaluated on three real-world 2D and 3D visual object recognition tasks. The work in this chapter has been previously published in [7, 9].

- *A new framework for single-view committee-based semi-supervised learning.* The study presented in Chapter 9 answers an important question: "What if there is not a natural feature splitting?". In this chapter, a single-view variant of *Co-Training*, called *Co-Training by Committee* (*CoBC*), is proposed, in which an ensemble of diverse classifiers is used instead of a set of redundant and independent views required by the original *Co-Training* algorithm. *CoBC* relax these requirements as they are hard to be satisfied in many real-world domains because there are not multiple representations available or it is computationally expensive to extract more than one feature set for each example. The aim of *CoBC* is to exploit the unlabeled data to improve the recognition rate of the underlying ensemble and to minimize the cost of data labeling. This chapter provide an answer to the question: "How to construct multiple classifiers to be co-trained?". As the *random subspace method* is used to construct the ensemble members based on different random feature subsets. This method can be used only if the features are abundant and redundant. This chapter answers the question: "How to measure prediction confidence?". A new method is introduced to measure the confidence that is based on estimating the local accuracy of the committee members on the neighborhood of a given unlabeled example. The work in this chapter has been previously published in [5, 4].

- *Two new frameworks for combining committee-base semi-supervised learning and active learning.* The study presented in Chapter 10 answers an important question: "Can active learning improve the performance of semi-supervised learning with committees?" I introduce two new learning frameworks, denoted as *QBC-then-CoBC* and *QBC-with-CoBC*, which combine the merits of committee-based *semi-supervised learning* and *active learning*. In *QBC-then-CoBC* approach, *CoBC* is to run after *QBC*. The objective is that active learning can help *CoBC* through providing it with a better starting point instead of randomly selecting examples to label for the starting point. In *QBC-with-CoBC* approach, *CoBC* is interleaved with *QBC*, so that *CoBC* not only runs on the results of active learning, but *CoBC* also helps *QBC* in the sample selection process as it augments the labeled training set with newly automatically labeled examples. Thus, mutual benefit

can be achieved. It is clear semi-supervised learning starts in *QBC-with-CoBC* at an earlier iterations compared to *QBC-then-CoBC*. Thus, *QBC-with-CoBC* can outperform *QBC-then-CoBC* only if the initial classifiers are accurate enough to automatically label the unlabeled examples. The work in this chapter has been previously published in [5, 4].

- *An extension of committee-based semi-supervised learning for regression.* I have proposed in Chapter 11 an extension of *CoBC* framework for regression, *CoBCReg*. I provided an answer to the question: "How to construct multiple classifiers to be co-trained?". The novel framework is based on an ensemble of *RBF network* regressors constructed by *Bagging*. This is achieved not only by training regressors using different training subsets but also through using different Minkowski distance orders and different random initialization of the regressors parameters. The applicability of the proposed algorithm is broader than standard *Co-Training* algorithm because it does not require multiple redundant and independent views. This chapter answers the question: "How to measure prediction confidence?". The main challenge for *CoBCReg* is the mechanism for estimating the confidence because the number of possible predictions in regression is unknown. In fact, I did not measure the *labeling confidence* but I provided another confidence measure called *selection confidence*. The most relevantly selected example is the one which minimizes the regressor error on the validation set. Fortunately, since the bootstrap sampling Br96 is used to construct the committee, the *out-of-bootstrap* examples are considered for a more accurate estimate of validation error. Experimental results show that *CoBCReg* can effectively exploit the unlabeled examples to improve the generalization error and it is robust to output noise. The work in this chapter has been previously published in [8].

- *A novel multi-view framework for semi-supervised learning with tri-class SVMs.* In Chapter 12, I developed a new framework for multi-class semi-supervised learning. First, multi-class problem is decomposed into a set of binary problems and then *Co-Training* is used to exploit unlabeled data in solving each binary problem. This chapter provide an answer to the question: "How to construct multiple classifiers to be co-trained effectively?" through the multi-view assumption. That is, for each binary problem, a classifier is trained based on each view. In order to answer the question: "How to measure prediction confidence?". In this chapter, a new probabilistic interpretation of the outputs of Tri-Class Support Vector Machine (SVM) is introduced where the confidence is derived from the predicted class probabilities. The main advantage of *Tri-Class SVM* is that it can discriminate between uncertainty and ignorance so it can reject the examples that do not belong to its target classes. In addition, a modified

version of the *Sequential Minimal Optimization* (SMO) algorithm is introduced for faster learning of the *Tri-Class SVM*s since *Co-Training* is an iterative method. The effectiveness of the proposed framework is evaluated on facial expressions recognition from image sequences. A task that involves a large number of classes and a small amount of labeled data. The results have shown that *Co-Training* with an ensemble of three multi-view *Tri-Class SVM*s can automatically improve the recognition rate using a small amount of human-labeled videos which minimize the cost of data labeling. The Gaussian Mixture Model (GMM) approach is used to extract the features, called *super vectors*, from facial expression videos. These *GMM super vectors* are the input of *Tri-Class SVM*s. The work in this chapter has been previously published in [2].

- *A new combination method for hierarchical ensembles.* Chapter 13 answers to the question: "Can a trainable combiner outperform non-trainable ones for hierarchical ensembles?" I developed a new trainable fusion method that integrates statistical information about the individual binary classifier outputs (in the form of *clustered decision templates*) into an RBF network combiner. Multivariate Gaussian function was used as similarity measure to match a *hierarchical decision profile* with decision templates. Not only RBF network was used as combiner but also it was used to construct the ensemble classifiers. The experiments were conducted on nine real-world multi-class object recognition tasks including digits, letters, fruits, 3d objects and textures. The experiments have shown that the *RBF Network* tree combiner significantly outperforms the three existing non-trainable tree combiners and the *decision templates* based combiner proposed by Kuncheva. The results also demonstrate that this neural combiner is robust to changes in the training set size and the number of decision templates per class.The work in this chapter has been previously published in [3].

- *A novel ensemble method based on evidence theory.* In Chapter 14, I presented a new ensemble method, denoted as *Multi-view Forest*. The aim is to answer the question "Can an ensemble of class hierarchies outperform a single class hierarchy?" Error diversity is an essential requirement to build an effective classifier ensemble. Diversity among classifiers means that they have independent (uncorrelated) errors. In order to construct diverse individual class hierarchies, it is assumed that the examples to be classified are described by multiple feature sets (views). The aim is to construct different tree classifiers using different combinations of views to improve the accuracy of the multi-class learning. Thus the output ensemble (forest) consists of both multi-view and single-view trees. For the decision fusion of the binary classifiers within each class hierarchy, Dempster's unnormalized rule of combination is applied and an evidence-theoretic decision profile is pro-

posed to combine the decisions of different trees. Experiments have been performed on two real-world data sets: a data set of handwritten digits, and another data set of 3D visual objects. The results indicate that the proposed forest efficiently integrates multi-view data and outperforms the individual tree classifiers. The work in this chapter has been previously published in [6, 1, 11].

- *An information-theoretic perspective for ensemble pruning.* I presented in Chapter 15 an information-theoretic perspective for classifier selection. Typically, ensemble methods comprise two phases: the construction of multiple individual classifiers and their combination. Recent work has considered an additional intermediate phase that deals with the reduction of the ensemble size prior to combination. Classifier selection is important for two reasons: classification accuracy and efficiency. An ensemble may consist not only of accurate classifiers, but also of classifiers with lower predictive accuracy. Pruning the poor-performing classifiers while maintaining a good diversity of the ensemble is typically considered as the main factor for an effective ensemble. The second reason is equally important, efficiency. Having a very large number of classifiers in an ensemble adds a lot of computational overhead. For example, decision tree classifiers may have large memory requirements and lazy learning methods have a considerable computational cost during classification phase. The minimization of classification time complexity is crucial in certain applications, such as stream mining. The aim of this paper is to answer the question "Can information theory be used to prune ensemble?" through using several selection criteria based on entropy and mutual information that take in account accuracy and diversity of the individual classifiers.

16.2 Future Directions

Through my research on semi-supervised learning, I have encountered many practical challenges and interesting empirical results, which have inspired several ideas for novel ways of looking at semi-supervised learning. This section introduces some of these ideas and problem settings, which I feel are fruitful directions for future work.

- *Semi-supervised learning via reinforcement learning.* Reinforcement Learning (RL) addresses the problem of how an agent can learn a behavior through trial-and-error interactions with a dynamic environment [180]. The agent, at each time step, interacts with the environment via actions, and tries to find an optimal policy of behavior with respect to "rewards" it receives from the environment. The objective of the agent is to maximize the cumulative reward received over time. For instance, consider a machine

that is learning how to play chess. In a supervised setting, one might provide the agent with board configurations from a database of chess games along with labels indicating which moves resulted in a win or loss. In a reinforcement setting, each board configuration (state) allows for certain moves (actions), which result in rewards that are positive (e.g., capturing the opponents queen) or negative (e.g., having its own queen taken). The agent aims to improve as it plays more games. The relationship with semi-supervised learning is that, in order to perform well, the learner must be proactive. In order to improve, a reinforcement learner must take risks and try out actions for which it is uncertain about the outcome, just as a semi-supervised classifier predicts class labels to unlabeled examples it is uncertain how to label. This is often called the exploration-exploitation trade-off in the reinforcement learning literature. Note that the issue of classifier selection was reformulated successfully as a reinforcement learning problem in [144].

- *Genetic algorithms based semi-supervised learning.* A genetic algorithm is a search heuristic that belong to the larger class of evolutionary algorithms (EA), It can generate solutions to optimization and search problems using techniques inspired by natural evolution [70], such as inheritance, mutation, selection, and crossover. Semi-supervised learning can be formulated as an optimization problem where the objective is to search for the unlabeled examples that when automatically labeled and added to the training set can improve the classification performance of the underlying classifier. Note that genetic algorithms are applied successfully for feature selection [206] and classifier selection [216].

- *Multi-instance semi-supervised learning.* The vast majority of semi-supervised learning research has assumed that each example corresponds to a single instance. In multi-instance learning problems, instances are naturally organized into bags and it is the bags, instead of individual instances, that are labeled for training. Many real-world learning problems can be reformalized under this framework. For instance, in text categorization, each document usually consists of several sections or paragraphs where each can be considered as an instance. In speech recognition, each speech generally encodes a number of segments each can be expressed as an instance. In video classification, a video generally contains several images each can be represented as an instance. Future research will investigate semi-supervised learning in multi-instance settings as a way to reduce the labeling burden.

- *Multi-label semi-supervised learning.* The vast majority of semi-supervised learning research has assumed that each example is associated to a single class label. In multi-label learning problems [185], an example is naturally assigned to a set of labels, instead of individual instances, that are labeled

for training. Many real-world learning problems belong to this framework. For instance, in text categorization, each document may be assigned to a set of predefined topics, such as "football", "South Africa", "World Cup" and "opening ceremony". In bioinformatics, each gene may be associated with several functional classes, such as metabolism, transcription and protein synthesis. In scene classification, an image can be related to multiple semantic classes simultaneously, such as sunset, sea and trees. Future research will consider semi-supervised learning in multi-label settings in order to further reduce the labeling cost.

- *Semi-supervised adversarial learning.* Many classification tasks, such as spam filtering, Fraud detection, Malware detection, intrusion detection in computers, and terrorism detection, are complicated by an adversary who wishes to avoid detection [114]. For instance, in spam filtering, classifiers often require a large training set of labeled emails to attain a good discriminant capability between spam and legitimate emails. In addition, they must be frequently updated to keep the filter effectiveness high and to deal with the changes introduced by spammers to their emails to avoid spam filters. Many spam filters allow the user to give a feedback on personal emails automatically labeled during filter operation, and some filters include a *Self-Training* technique to exploit the large number of unlabeled emails collected during filter operation. However, users are not willing to label many emails, and the benefits of *Self-Training* technique are limited. To address this issue active learning and semi-supervised learning methods can be used.

16.3 Last Words

In this thesis, I have explored semi-supervised learning in a variety of real-world problem domains characterized by single view or multiple view examples representation. This work has helped to answer research questions about semi-supervised learning in some of these applications, e.g., "How to construct multiple classifiers to be co-trained effectively?", "How to measure prediction confidence?" and "Can hierarchical neural network classifiers use unlabeled data to improve the accuracy of image classification?" At the same time, this work has also introduced answers for some questions about ensemble learning, e.g., "Can a trainable combiner outperform non-trainable ones for hierarchical ensembles?" and "Can an ensemble of class hierarchies outperform a single class hierarchy?" It is my hope that the research findings I have presented here will serve as a foundation for future work in semi-supervised learning applied to real-world learning problems.

Bibliography

[1] M.F. Abdel Hady, G. Palm, and F. Schwenker. Multi-view forests of radial basis function networks based on Dempster-Shafer evidence theory. In *Proc. of the 16th European Symposium on Artificial Neural Networks (ESANN 2008)*, pages 307–312. d-side publications, 2008.

[2] M.F. Abdel Hady, M. Schels, F. Schwenker, and G. Palm. Semi-supervised facial expressions annotation using Co-Training with fast probabilistic tri-class SVMs. In *Proc. of the 20th International Conference on Artificial Neural Networks (ICANN 2010)*, volume 6353 of *LNCS*, pages 70–75. Springer-Verlag, 2010.

[3] M.F. Abdel Hady and F. Schwenker. Decision templates based RBF network for tree-structured multiple classifier fusion. In *Proc. of the 18th International Workshop on Multiple Classifier Systems (MCS 2009)*, volume 5519 of *LNCS*, pages 92–101. Springer-Verlag, 2009.

[4] M.F. Abdel Hady and F. Schwenker. Combining committee-based semi-supervised learning and active learning. *Journal of Computer Science and Technology (JCST): Special Issue on Advances in Machine Learning and Applications*, 25(4):681–698, 2010.

[5] M.F. Abdel Hady and F. Schwenker. On combining committee-based semi-supervised and active learning and its application to handwritten digits recognition. In *Proc. of the 9th International Workshop on Multiple Classifier Systems (MCS 2010)*, volume 5997 of *LNCS*, pages 225–234. Springer-Verlag, 2010.

[6] M.F. Abdel Hady, F. Schwenker, and G. Palm. Multi-view forests based on Dempster-Shafer evidence theory: A new classifier ensemble method. In *Proc. of the 5th IASTED Conference on Signal Processing, Pattern Recognition and Applications (SPPRA 2008)*, pages 18–23. ACTA Press, 2008.

[7] M.F. Abdel Hady, F. Schwenker, and G. Palm. Semi-supervised learning of tree-structured RBF networks using Co-Training. In *Proc. of the 18th International Conference on Artificial Neural Networks (ICANN 2008)*, volume 5163 of *LNCS*, pages 79–88. Springer-Verlag, 2008.

[8] M.F. Abdel Hady, F. Schwenker, and G. Palm. Semi-supervised learning for regression with Co-Training by committee. In *Proc. of the 19th International Conference on Artificial Neural Networks (ICANN 2009)*, volume 5768 of *LNCS*, pages 121–130. Springer-Verlag, 2009.

[9] M.F. Abdel Hady, F. Schwenker, and G. Palm. Semi-supervised learning for tree-structured ensembles of RBF networks with Co-Training. *Neural Networks*, 23(4):497–509, 2010.

[10] M.F. Abdel Hady, F. Schwenker, and G. Palm. When classifier selection meets information theory: A unifying view. In *Proc. of the International Conference on Soft Computing and PAttern Recognition (SoCPaR 2010)*, pages 314–319. IEEE Computer Society, 2010.

[11] M.F. Abdel Hady, F. Schwenker, and G. Palm. Multi-View Forest: A new ensemble method based on Dempster-Shafer evidence theory. *International Journal of Applied Mathematics and Statistics (IJAMAS): Special Issue on Soft Computing and Approximate Reasoning*, 22(S11):2–19, 2011.

[12] N. Abe and H. Mamitsuka. Query learning strategies using boosting and bagging. In *Proc. of the 15th the International Conference on Machine Learning (ICML'98)*, pages 1–9, 1998.

[13] S. Abe. *Support vector machines for pattern classification*. Springer, 2005.

[14] E.L. Allwein, R.E. Shapire, and Y. Singer. Reducing multiclass to binary: A unifying approach for margin classifiers. *Journal of Machine Learning Research*, 1:113–141, 2000.

[15] E. Alpaydin and M.I. Jordan. Local linear perceptrons for classification. *IEEE Transactions on Neural Networks*, 7(3):788–792, 1996.

[16] C. Angulo, F. J. Ruiz, L. González, and J. A. Ortega. Multi-Classification by using Tri-Class SVM. *Neural Processing Letters*, 23(1):89–101, 2006.

[17] C. Aviles-Cruz, A. Gurin-Dugu, J.L. Voz, and D. Van Cappel. Databases, Enhanced Learning for Evolutive Neural Architecture. Tech. Rep. R3-B1-P, INPG, UCL, TSA (1995) 47, `http://www.dice.ucl.ac.be/neural-nets/Research/Projects/ELENA/elena.htm`.

[18] M.-F. Balcan, A. Blum, and K. Yang. Co-Training and expansion: Towards bridging theory and practice. In *Advances in Neural Information Processing Systems 17*, pages 89–96, 2005.

[19] S. Basu, A. Banerjee, and R. Mooney. Semi-supervised clustering by seeding. In *Proc. of the 19th International Conference on Machine Learning (ICML'02)*, pages 19–26, 2002.

[20] S. Basu, M. Bilenko, and R. Mooney. A probabilistic framework for semi-supervised clustering. In *Proc. of the 10th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD04)*, pages 59–68, 2004.

[21] R. Battiti. Using mutual information for selecting features in supervised neural net learning. *IEEE Transactions on Neural Networks*, 5(4):537–550, 1994.

[22] P. Bayerl and H. Neumann. Disambiguating visual motion through contextual feedback modulation. *Neural Computation*, 16(10):2041–2066, 2004.

[23] M. Belkin, P. Niyogi, and V. Sindhwani. Manifold regularization: A geometric framework for learning from labeled and unlabeled examples. *Journal of Machine Learning Research*, 7:23992434, 2006.

[24] K. Bennet, A. Demiriz, and R. Maclin. Exploiting unlabeled data in ensemble methods. In *Proc. of the 8th ACM SIGKDD International Conference On Knowledge Discovery and Data Mining*, pages 289–296, 2002.

[25] C. M. Bishop. *Neural networks for pattern recognition.* Oxford: Clarendon Press, 1995.

[26] C. M. Bishop. *Pattern Recognition and Machine Learning.* Springer, 2006.

[27] C. Blake and C.J. Merz. UCI repository of machine learning databases. University of California, Irvine, School of Information and Computer Sciences, 1998. `http://www.ics.uci.edu/~mlearn/MLRepository.html`.

[28] A. Blum and S. Chawla. Learning from labeled and unlabeled data using graph mincuts. In *Proc. of the 18th International Conference on Machine Learning (ICML'01)*, pages 19–26, 2001.

[29] A. Blum, J. Lafferty, M. Rwebangira, and R. Reddy. Semi-supervised learning using randomized mincuts. In *Proc. of the 21st International Conference on Machine Learning (ICML'04)*, pages 13–20, 2004.

[30] A. Blum and T. Mitchell. Combining labeled and unlabeled data with Co-Training. In *Proc. of the 11th Annual Conference on Computational Learning Theory (COLT 1998):*, pages 92–100. Morgan Kaufmann, 1998.

[31] L. Breiman. Bagging predictors. *Machine Learning*, 24(2):123–140, 1996.

[32] L. Breiman. Random forests. *Machine Learning*, 45(1):5–32, 2001.

[33] L. Breiman, J.H. Friedman, and R.A. Olshen. *Classification and Regression Trees*. Taylor and Francis, Inc, 1984.

[34] D. Broomhead and D. Lowe. Multivariable functional interpolation and adaptive networks. *Complex Systems*, 2:321–355, 1988.

[35] G. Brown. A new perspective for information theoretic feature selection. In *Proc. of the 12th International Conference on Artificial Intelligence and Statistics (AI-STATS 2009)*, 2009.

[36] G. Brown, J. Wyatt, R. Harris, and X. Yao. Diversity creation methods: a survey and categorisation. *Information Fusion*, 6(1):5–20, 2005.

[37] W. M. Campbell, D. E. Sturim, and D. A. Reynolds. Support vector machines using GMM supervectors for speaker verification. *IEEE Signal Processing Letters*, 13(5):308–311, 2006.

[38] J. Canny. A computational approach to edge detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 8(6):679698, 1986.

[39] O. Chapelle and A. Zien. Semi-supervised learning by low density separation. In *Proc. of the 10th International Workshop on Artificial Intelligence and Statistics*, pages 57–64, 2005.

[40] V. Cherkassky and F. Mulier. *Learning from Data: Concepts, Theory and Methods*. John Wiley and Sons, Inc. New York, 1998.

[41] S.-B. Cho and J.H. Kim. Combining multiple neural networks by fuzzy integral and robust classification. *IEEE Transactions on Systems, Man, and Cybernetics*, 25:380–384, 1995.

[42] W. Chu and S. S. Keerthi. New approaches to support vector ordinal regression. In *Proc. of the 22nd International Conference on Machine Learning*, pages 145–152, 2005.

[43] D.M. Coppola, H.R. Purves, A.N. McCoy, and D. Purves. The distribution of oriented contours in the real world. *Proceedings of the National Academy of Sciences of the United States of America*, 95(7):40024006, 1998.

[44] C. Cortes and V. Vapnik. Support vector networks. *Journal of Machine Learning*, 20(3):273–297, 1995.

[45] F. G. Cozman and I. Cohen. Unlabeled data can degrade classification performance of generative classifiers. In *Proc. of the 15th International Conference of the Florida Artificial Intelligence Research Society (FLAIRS)*, page 327331, 2002.

[46] I. Dagan and S. Engelson. Committee-based sampling for training probabilistic classifiers. In *Proc. of the 12th International Conference on Machine Learning (ICML'95)*, pages 150–157, 1995.

[47] F. d'Alché Buc, Y. Grandvalet, and C. Ambroise. Semi-supervised MarginBoost. In *Neural Information Processing Systems Foundation, NIPS 2002*, 2002.

[48] B.V. Dasarathy. *Nearest Neighbor (NN) Norms: NN Pattern Classification Techniques*. IEEE Computer Society Press, Los Alamitos, California, 1990.

[49] I. Davidson. An ensemble technique for stable learners with performance bounds. In *Proc. of the 19th National Conference on Artificial Intelligence (AAAI- 2004)*, page 330335, 2004.

[50] A. P. Dempster. A generalization of bayesian inference. *Journal of the Royal Statistical Society*, pages 205–247, 1968.

[51] A.P. Dempster, N.M. Laird, and D.B. Rubin. Maximum likelihood from incomplete data via the em algorithm. *Journal of the Royal Statistical Society. Series B (Methodological)*, 39(1):1–38, 1977.

[52] J. Demsar. Statistical comparisons of classifiers over multiple data sets. *Journal of Machine Learning Research*, 7:1–30, 2006.

[53] C. Dietrich, G. Palm, and F. Schwenker. Decision templates for the classification of bioacoustic time series. *Information Fusion*, 4:101–109, 2003.

[54] T. Dietterich. *The Handbook of Brain Theory and Neural Networks*, chapter Ensemble Learning, page 405408. MIT Press, 2002.

[55] T.G. Dietterich. Ensemble methods in machine learning. In *Proc. of the First International Workshop on Multiple Classifier Systems (MCS'2000), Cagliari, Italy*, volume 1857 of *LNCS*. Springer-Verlag, 2000.

[56] T.G. Dietterich. An experimental comparison of three methods for constructing ensembles of decision trees: Bagging, boosting, and randomization. *Machine Learning*, pages 139–157, 2000.

[57] T.G. Dietterich and G. Bakiri. Solving multiclass learning problems via error-correcting output codes. *Journal of Artificial Intelligence Research*, 2:263–286, 1995.

[58] R.O. Duda, P.E. Hart, and D.G. Stork. *Pattern Classification*. John Wiley and Sons, New York, 2nd edition, 2001.

[59] R. P. W. Duin. The combining classifier: to train or not to train? In *16th International Conference on Pattern Recognition (ICPR02)*, page 765770, 2002.

[60] R. Fay. *Feature selection and information fusion in hierarchical neural networks for iterative 3D-object recognition*. PhD thesis, Ulm University, 2007.

[61] R. Fay, F. Schwenker, C. Thiel, and G. Palm. Hierarchical neural networks utilising dempster-shafer evidence theory. In *Proceedings of the 2nd IAPR TC3 Workshop on Artificial neural Networks in Pattern Recognition (ANNPR 2006)*, volume 4087 of *LNAI*, pages 198–209, 2006.

[62] F. Feger and I. Koprinska. Co-Training using RBF nets and different feature splits. In *Proc. of the International Joint Conference on Neural Networks (IJCNN'06)*, pages 1878–1885, 2006.

[63] F. Fleuret. Fast binary feature selection with conditional mutual information. *Journal of Machine Learning Research*, 5:1531–1555, 2004.

[64] W.T. Freeman and M. Roth. Orientation histograms for hand gesture recognition. In *IEEE International Workshop on Automatic Face and Gesture-Recognition*, page 296301, 1995.

[65] Y. Freund and R. E. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of Computer and System Sciences*, 55(1):119139, 1997.

[66] Y. Freund, H.S. Seung, E. Shamir, and N. Tishby. Selective sampling using the query by committee algorithm. *Machine Learning*, 28:133–168, 1997.

[67] A. Fujii, T. Tokunaga, K. Inui, and H. Tanaka. Selective sampling for example-based word sense disambiguation. *Computational Linguistics*, 24(4):573–597, 1998.

[68] K. Fukunaga. *Introduction to Statistical Pattern Recognition*. Academic Press, Inc., Orlando, FL, 1972.

[69] R. Ghani. Combining labeled and unlabeled data for text classification with a large number of categories. In *Proc. of the First IEEE Conference on Data Mining (ICDM 2001)*, pages 597–598. IEEE Computer Society, 2001.

[70] D.E. Goldberg. *Genetic Algorithms in Search Optimization and Machine Learning*. Addison Wesley, 1989.

[71] S. Goldman and Y. Zhou. Enhancing supervised learning with unlabeled data. In *Proc. of the 17th International Conference on Machine Learning (ICML'00)*, pages 327–334, 2000.

[72] R.C. Gonzales and R.E. Woods. *Digital Image Processing*. Addison-Wesley, 1992.

[73] Y. Grandvalet and Y. Bengio. Semi-supervised learning by entropy minimization. *Advances in Neural Information Processing Systems*, 17:529–536, 2005.

[74] A. Gupta and S. Dasgupta. Hybrid hierarchical clustering: Forming a tree from multiple views. In *Workshop on Learning with Multiple Views, 22nd ICML*, pages 35–42, 2005.

[75] A. Gupta and S. Dasgupta. Hybrid hierarchical clustering: Forming a tree from multiple views. In *Proc. of the 22nd ICML Workshop on Learning with Multiple Views*, pages 35–42, 2005.

[76] M. Halkidi, Y. Batistakis, and M. Vazirgiannis. Clustering validity checking methods: part ii. *ACM SIGMOD Record*, 31(3):19–27, 2002.

[77] J.V. Hansen. *Combining predictors: meta machine learning methods and bias/variance and ambiguity*. PhD thesis, University of Aarhus, Denmark, 2000.

[78] L. Hansen and P. Salamon. Neural network ensembles. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 12:993–1001, 1990.

[79] H.A. Hefny, A.H. Abdel Wahab, A.A. Bahnasawi, and S.I. Shaheen. A novel framework for hybrid intelligent systems. In *Proc. of the 12th International Conference on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems (IAE/AIE)*, volume 1611 of *LNCS*, pages 761–770, 1999.

[80] T. K. Ho. Data complexity analysis for classifier combination. In *Multiple Classifier Systems*, pages 53–63. Springer, 2001.

[81] T.K. Ho. Nearest neighbors in random subspaces. In *Lecture Notes in Computer Science: Advances in Pattern Recognition*, pages 640–648. Springer, 1998.

[82] T.K. Ho. The random subspace method for constructing decision forests. *IEEE Transactions Pattern Analysis and Machine Intelligence*, 20(8):832–844, 1998.

[83] S.C.H. Hoi and Michael R. Lyu. A semi-supervised active learning framework for image retrieval. In *Proc. of IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 302–309, 2005.

[84] Y.S. Huang and C.Y. Suen. A method of combining multiple classifiers - a neural network approach. In *12th International Conference on Pattern Recognition (ICPR94)*, pages 473–475, 1994.

[85] Y.S. Huang and C.Y. Suen. A method of combining multiple experts for the recognition of unconstrained handwritten numerals. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 17:90–93, 1995.

[86] M. Inoue and N. Ueda. Exploitation of unlabeled sequences in hidden markov models. *IEEE Transactions On Pattern Analysis and Machine Intelligence*, 25(12):1570–1581, 2003.

[87] R.A. Jacobs, M.I. Jordan, S.J. Nowlan, and G.E. Hinton. Adaptive mixtures of local experts. *Neural Computation*, 3:79–87, 1991.

[88] T. Joachims. Transductive inference for text classification using support vector machines. In *Proc. of the 16th International Conference on Machine Learning*, pages 200–209, 1999.

[89] I.T. Jolliffe. *Principal Component Analysis*. Springer-Verlag, 1986.

[90] G. Jun and J. Ghosh. Hybrid hierarchical classifiers for hyperspectral data analysis. In *8th International Workshop on Multiple Classifier Systems (MCS 2009)*, volume 5519 of *LNCS*, pages 42–51. Springer-Verlag, 2009.

[91] G. Jun and J. Ghosh. Multi-class boosting with class hierarchies. In *8th International Workshop on Multiple Classifier Systems (MCS 2009)*, volume 5519 of *LNCS*, pages 32–41. Springer-Verlag, 2009.

[92] L. Kahsay, F. Schwenker, and G. Palm. Comparison of multiclass SVM decomposition schemes for visual object recognition. In *Proceedings of DAGM 2005, Munich, Germany*, volume 3663 of *LNCS*, pages 334–341. Springer-Verlag, 2005.

[93] T. Kanade, J. Cohn, and Y.-L. Tian. Comprehensive database for facial expression analysis. In *Proc. of the 4th IEEE International Conference on Automatic Face and Gesture Recognition (FG'00)*, pages 46–53, 2000.

[94] S. Keerthi, S. K. Shevade, C. Bhattacharyya, and K. R. K. Murthy. Improvements to Platt's SMO algorithm for SVM classifier design. *Neural Computation*, 13:637–649, 2001.

[95] T. Kemp and A. Waibel. Unsupervised training of a speech recognizer: Recent experiments. In *Proc. EUROSPEECH*, pages 2725–2728, 1999.

[96] S. Kiritchenko and S. Matwin. Email classification with Co-Training. In *Proc. of the 2001 Conference of the Centre for Advanced Studies on Collaborative research (CASCON'01)*, pages 8–19. IBM Press, 2001.

[97] R. Kohavi. Scaling up the accuracy of naive-bayes classifiers: A decision-tree hybrid. In *Proc. of the Second International Conference on Knowledge Discovery and Data Mining (KDD-96)*, pages 202–207. AAAI Press, 1996.

[98] T. Kohonen. *Self-Organizing Maps*. Springer, Berlin, 1995.

[99] T. Kohonen, J. Hynninen, J. Kangas, J. Laaksonen, and K. Torkkola. Lvq pak: The learning vector quantization program package. Technical report, Helsinki University of Technology, 1996.

[100] J.F. Kolen and J.B. Pollack. Back propagation is sensitive to initial conditions. *Advances in Neural Information Processing Systems*, 3:860–867, 1991.

[101] U. Kressel. Pairwise classification and support vector machines. *Advances in kernel methods: Support Vector Learning*, pages 255–268, 1999.

[102] V. Krishnamurthy. Algorithms for optimal scheduling and management of hidden Markov model sensors. *IEEE Transactions on Signal Processing*, 50(6):1382–1397, 2002.

[103] A. Krogh and J. Vedelsby. Neural network ensembles, cross validation, and active learning. *Advances in Neural Information Processing Systems*, 7:231–238, 1995.

[104] M. Kubat. Decision trees can initialize radial-basis-function networks. *IEEE Transactions on Neural Networks*, 9:813–821, 1998.

[105] S. Kumar. *Modular Learning through Output Space Decomposition*. PhD thesis, The University of Texas at Austin, Austin, TX, 2000.

[106] L. Kuncheva. *Pattern Recognition: From Classical to Modern Approaches*, chapter 15, page 427451. World Scientific, 2001.

[107] L. Kuncheva. *Combining Pattern Classifiers: Methods and Algorithms*. Wiley, 2004.

[108] L. Kuncheva and J. Bezdek. An integrated framework for generalized nearest prototype classifier design. *International Journal of Uncertainty, Fuzziness and Knowledge-based Systems*, 6(5):437–457, 1998.

[109] L. Kuncheva, J. Bezdek, and R. Duin. Decision templates for multiple classifier fusion: An experimental comparison. *Pattern Recognition*, 34(2):299314, 2001.

[110] L. I. Kuncheva and C. J. Whitaker. Measures of diversity in classifier ensembles and their relationship with the ensemble accuracy. *Machine Learning*, 51(2):181–207, 2003.

[111] L.I. Kuncheva. An application of owa operators to the aggregation of multiple classification decisions. *The Ordered Weighted Averaging operators. Theory and Applications*, pages 330–343, 1997.

[112] L. Lam and C.Y. Suen. Application of majority voting to pattern recognition: An analysis of its behavior and performance. *IEEE Transactions on Systems, Man, and Cybernetics*, 27(5):553–568, 1997.

[113] P. J. Lang. The emotion probe: Studies of motivation and attention. *American psychologist*, 50(5):372–385, 1995.

[114] P. Laskov and R. Lippmann. Machine learning in adversarial environments. *Machine learning*, 81(2):115–119, 2010.

[115] N.D. Lawrence and M.I. Jordan. Semi-supervised learning via gaussian processes. *Advances in Neural Information Processing Systems*, 17:753760, 2005.

[116] A. Levin, P. Viola, and Y. Freund. Unsupervised improvement of visual detectors using Co-Training. In *Proc. of the International Conference on Computer Vision*, pages 626–633, 2003.

[117] D. Lewis and J. Catlett. Heterogeneous uncertainty sampling for supervised learning. In *Proc. of the 11th International Conference on Machine Learning (ICML'94)*, pages 148–156, 1994.

[118] D. Lewis and W. Gale. A sequential algorithm for training text classifiers. In *Proc. of the ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 3–12, 1994.

[119] M. Li and Z.-H. Zhou. Improve computer-aided diagnosis with machine learning techniques using undiagnosed samples. *IEEE Transactions on Systems, Man and Cybernetics- Part A: Systems and Humans*, 37(6):1088–1098, 2007.

[120] H. Liang and Y. Yan. Improve decision trees for probability-based ranking by lazy learners. In *Proc. of the 18th IEEE International Conference on Tools with Artificial Intelligence (ICTAI'06)*, pages 427–435. IEEE Computer Society, 2006.

[121] D. Lin and X. Tang. Conditional infomax learning: An integrated framework for feature extraction and fusion. In *European Conference on Computer Vision*, 2006.

[122] Y. Liu. Active learning with support vector machine applied to gene expression data for cancer classification. *Journal of Chemical Information and Computer Sciences*, 44:1936–1941, 2004.

[123] J. MacQueen. Some methods for classification and analysis of multivariate observations. In *Proc. of 5th Berkeley Symposium on Mathematical Statistics and Probability*, pages 281–297, 1967.

[124] D.D. Margineantu and T.G. Dietterich. Pruning adaptive boosting. In *Proc. of the 14th International Conference on Machine Learning (ICML'97)*, pages 211–218. Morgan Kaufmann, 1997.

[125] A. K. McCallum and K. Nigam. Employing EM and pool-based active learning for text classification. In *Proc. of the 15th International Conference on Machine Learning (ICML'98)*, pages 350–358. Morgan Kaufmann, 1998.

[126] W. McGill. Multivariate information transmission. *IEEE Transactions on Information Theory*, 4(4):93–111, 1954.

[127] P. Melville and R.J. Mooney. Creating diversity in ensembles using artificial data. *Journal of Information Fusion: Special Issue on Diversity in Multi Classifier Systems*, 6(1):99–111, 2004.

[128] P. Melville and R.J. Mooney. Diverse ensembles for active learning. In *Proc. of the 21st International Conference on Machine Learning (ICML'04)*, pages 584–591, 2004.

[129] J. Meynet and J. Thiran. Information theoretic combination of classifiers with application to adaboost. In *Proc. of the 7th International Workshop on Multiple Classifier Systems (MCS 2007)*, volume 4472 of *LNCS*, pages 171–179, 2007.

[130] D. Michie, D.J. Spiegelhalter, and C.C. Taylor. *Machine Learning, Neural and Statistical Classification.* Ellis Horwood, 1994.

[131] D. J. Miller and H. S. Uyar. A mixture of experts classifier with learning based on both labelled and unlabelled data. *Advances in Neural Information Processing Systems*, 9:571577, 1997.

[132] J. T. Morgan. *Adaptive Hierarchical Classification with Limited Training Data.* PhD thesis, The University of Texas at Austin, Austin, TX, 2002.

[133] I. Muslea, S. Minton, and C. A. Knoblock. Active + semi-supervised learning = robust multi-view learning. In *Proc. of the 19th International Conference on Machine Learning (ICML'02)*, pages 435–442, 2002.

[134] I. Muslea, S. Minton, and C. A. Knoblock. Active learning with multiple views. *Journal of Artificial Intelligence Research (JAIR)*, 27:203–233, 2006.

[135] C. Nadeau and Y. Bengio. Inference for the generalization error. *Machine Learning*, 62:239–281, 2003.

[136] G. Nagy and G.L. Shelton. Self-corrective character recognition systems. *IEEE Transactions On Information Theory*, pages 215–222, 1966.

[137] S. Nene, S. Nayar, and H. Murase. Columbia object image library: COIL. `http://citeseer.ist.psu.edu/article/nene96columbia.html`, 1996.

[138] K. Nigam. *Using Unlabeled Data to Improve Text Classification*. PhD thesis, School of Computer Science, Carnegie Mellon University, Pittsburgh, USA, 2001.

[139] K. Nigam and R. Ghani. Analyzing the effectiveness and applicability of Co-Training. In *Proc. of the 9th International Conference on Information and Knowledge Management*, pages 86–93, New York, NY, USA, 2000.

[140] K. Nigam, A. K. McCallum, S. Thrun, and T. Mitchell. Text classification from labeled and unlabeled documents using EM. *Machine Learning*, 39(2-3):103–134, 2000.

[141] N. Oza and K. Tumer. Input decimation ensembles: decorrelation through dimensionality reduction. In *Proc. of the International Workshop on Multiple Classifier Systems (MCS 2001)*, volume 2096 of *LNCS*, pages 238–247, 2001.

[142] J. Park and I.W. Sandberg. Approximation and radial basis function networks. *Neural Computation*, 5:305–316, 1993.

[143] J. R. Parker. Rank and response combination from confusion matrix data. *Information Fusion*, 2(2):113–120, 2001.

[144] I. Partalas, G. Tsoumakas, I. Katakis, and I. Vlahavas. Ensemble pruning using reinforcement learning. In *Proc. of the 4th Hellenic Conference on Artificial Intelligence (SETN 2006)*, pages 301–310, 2006.

[145] H. Peng, F. Long, and C. Ding. Feature selection based on mutual information: Criteria of max-dependency, max-relevance, and min-redundancy. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 27(8):1226–1238, 2005.

[146] C. Persello and L. Bruzzone. Active learning for classification of remote sensing images. In *Proc. of the IEEE International Conference on Geoscience and Remote Sensing Symposium (IGARSS 2009)*, pages 693–696, 2009.

[147] J. C. Platt. *Advances in Kernel Methods: Support Vector Learning*. MIT Press, Cambridge, MA, 1998.

[148] J.C. Platt, N. Cristianini, and J. Shawe-Taylor. Large margin DAGs for multiclass classification. *Advances in Neural Information Processing Systems*, 12:547–553, 2000.

[149] T. Poggio and F. Girosi. Networks for approximation and learning. *Proceedings of the IEEE*, 78(9):1481–1497, 1990.

[150] F. J. Provost and P. Domingos. Tree induction for probability-based ranking. *Machine Learning*, 52(30), 2003.

[151] J.R. Quinlan. Induction of decision trees. *Machine Learning*, 1(1), 1986.

[152] J.R. Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann, San Francisco, CA, 1993.

[153] S. Rajan and J. Ghosh. An empirical comparison of hierarchical vs. two-level approaches to multiclass problems. In *5th Workshop on Multiple Classifier Systems (MCS 2004)*, volume 3077 of *LNCS*, page 283292. Springer-Verlag, 2004.

[154] D.A. Reynolds, T.F. Quatieri, and R.B. Dunn. Speaker verification using adapted gaussian mixture models. In *Digital Signal Processing*, 2000.

[155] G. Ridgeway, D. Madigan, and T. Richardson. Boosting methodology for regression problems. In *Proc. of Second International Conference on Artificial Intelligence and Statistics (AISTATS-99)*, pages 152–161. Morgan Kaufmann, 1999.

[156] B. Ripley. *Pattern recognition and neural networks*. Cambridge University Press, 1996.

[157] J. Rissanen. Universal coding, information, prediction, and estimation. *IEEE Transactions on Information Theory*, 30(4):629–636, 1984.

[158] G. Rogova. Combining the results of several neural network classifiers. *Neural Networks*, 7:777–781, 1994.

[159] F. Roli. Semi-supervised multiple classifier systems: Background and research directions. In *Proc. of the 6th International Workshop on Multiple Classifier Systems (MCS 2005)*, page 111, 2005.

[160] F. Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological Review*, 65(6):386–408, 1958.

[161] M. Rosenblum, Y. Yacoob, and L. Davis. Human expression recognition from motion using a radial basis function network architecture. *IEEE Transactions on Neural Networks*, 7(5):1121–1138, 1996.

[162] A. Salaheldin and N. El Gayar. New feature splitting criteria for Co-Training using genetic algorithm optimization. In *Proc. of the International Workshop on Multiple Classifier Systems (MCS 2010)*, volume 5997 of *LNCS*, pages 22–32. Springer-Verlag, 2010.

[163] M. Schels. Lernen mit unsicheren lehrersignalen zu erkennung von gesichtsausdrücken in bildsequenzen. Master's thesis, University of Ulm, Ulm, Germany, 2008.

[164] B. Schölkopf and A.J. Smola. *Learning with Kernels: Support Vector Machines, Regularization, Optimization and Beyond*. MIT Press, Cambridge, MA, 2002.

[165] F. Schwenker and C. Dietrich. Initialization of radial basis function networks using classification trees. *Neural Network World*, 10:473482, 2000.

[166] F. Schwenker, H.A. Kestler, and G. Palm. Three learning phases for radial basis function networks. *Neural Networks*, 14:439–458, 2001.

[167] F. Schwenker, H.A. Kestler, G. Palm, and M. Höher. Similarities of LVQ and RBF learning. In *Proc. IEEE International Conference SMC*, pages 646–51, 1994.

[168] F. Schwenker and G. Palm. Tree structured support vector machines for multi-class pattern recognition. In *Proc. of the Second International Workshop on Multiple Classifier Systems (MCS 2001)*, volume 2096 of *LNCS*, pages 409–417. Springer-Verlag, 2001.

[169] F. Schwenker, A. Sachs, G. Palm, and H. A. Kestler. Orientation histograms for face recognition. In *International Workshop on Artificial Neural Networks in Pattern Recognition (ANNPR)*, volume 4087 of *LNCS*, pages 253–259. Springer, 2006.

[170] M. Seeger. Learning with labeled and unlabeled data. Technical report, University of Edinburgh, Institute for Adaptive and Neural Computation, 2002.

[171] K. Sentz. *Combination of Evidence in Dempster-Shafer Theory*. PhD thesis, Binghamton University, Binghamton, NY, 2002.

[172] B. Settles. *Curious Machines: Active Learning with Structured Instances*. PhD thesis, Department of Computer Sciences, University of Wisconsin-Madison, Madison, WI, 2008.

[173] B. Settles. Active learning literature survey. Technical report, Department of Computer Sciences, University of Wisconsin-Madison, Madison, WI, 2009.

[174] G. Shafer. *A Mathematical Theory of Evidence*. Princeton University Press, Princeton, New Jersey, 1976.

[175] B. Shahshahani and D. Landgrebe. The effect of unlabeled samples in reducing the small sample size problem and mitigating the hughes phenomenon. *IEEE Transactions on Geoscience and Remote Sensing*, 32(5):1087–1095, 1994.

[176] A. Shashua and A. Levin. Taxonomy of large margin principle algorithms for ordinal regression problems. *Advances in Neural Information Processing Systems*, 15, 2002.

[177] D.L. Shrestha and D.P. Solomatine. Experiments with adaboost.rt, an improved boosting scheme for regression. *Neural Computation*, 18(7):1678–1710, 2006.

[178] D. Skalak. *Prototype Selection for Composite Nearest Neighbor Classifiers*. PhD thesis, Department of Computer Science, University of Massachusetts, 1996.

[179] P. Smyth, A. Gray, E. Gray, and U.M. Fayyad. Retrofitting decision tree classifiers using kernel density estimation. pages 506–514. Morgan Kaufmann, 1995.

[180] R. Sutton and A.G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, 1998.

[181] W. Tang and S. Zhong. Pairwise constraints-guided dimensinality reduction. In *Proc. of the SDM06 Workshop on Feature Selection for Data Mining*, 2006.

[182] R. Tibshirani and T. Hastie. Margin trees for high-dimensional classification. *Journal of Machine Learning Research*, 8:637–652, 2007.

[183] S. Tong and E. Chang. Support vector machine active learning for image retrieval. In *Proc. of the ACM International Conference on Multimedia*, pages 107–118. ACM Press, 2001.

[184] S. Tong and D. Koller. Support vector machine active learning with applications to text classification. In *Proc. of the 17th International Conference on Machine Learning (ICML'00)*, pages 999–1006, 2000.

[185] G. Tsoumakas and I. Katakis. Multi-label classification: An overview. *International Journal of Data Warehousing and Mining*, 3:1–13, 2007.

[186] G. Tsoumakas, I. Partalas, and I. Vlahavas. A taxonomy and short review of ensemble selection. In *ECAI 2008 Workshop on Supervised and Unsupervised Ensemble Methods and Their Applications (SUEMA)*, 2008.

[187] D. Tuia, F. Ratle, F. Pacifici, M. F. Kanevski, and W. J. Emery. Active learning methods for remote sensing image classification. *IEEE Transactions on Geoscience and Remote Sensing*, 47(7):2218–2232, 2009.

[188] K. Tumer and J. Ghosh. Error correlation and error reduction in ensemble classifiers. *Connection Science*, 8(3–4):385–403, 1996.

[189] G. Tur, D. Hakkani-Tür, and R.E. Schapire. Combining active and semi-supervised learning for spoken language understanding. *Speech Communication*, 45(2):171–186, 2005.

[190] M. Turk and A. Pentland. Eigenfaces for recognition. *Journal of Cognitive Neuroscience*, 3(1):71–86, 1991.

[191] N. Ueda and R. Nakano. Generalization error of ensemble estimators. In *International Conference on Neural Networks*, pages 90–95, 1996.

[192] N. Ussivakul and B. Kijsirikul. Multiclass support vector machines using adaptive directed acyclic graph. In *IEEE/INNS International Joint Conference on Neural Networks (IJCNN-2002)*, 2002.

[193] V. Vapnik. *The Nature of Statistical Learning Theory*. Springer-Verlag, 1995.

[194] P. Verlinde and G. Chollet. Combining vocal and visual cues in an identity verification system using k-nn based classifiers. In *Proc. of the IEEE Workshop on Multi-Media Signal Processing*, 1998.

[195] K. Wagstaff, C. Cardie, and S. Schroedl. Constrained k-means clustering with background knowledge. In *Proc. of the 18th International Conference on Machine Learning (ICML01)*, pages 577–584, 2001.

[196] G. Wahba. Multivariate function and operator estimation, based on smoothing splines and reproducing kernels. *Proc. Nonlinear Modeling and Forecasting, SFI Studies in the Sciences of Complexity*, XII:95–112, 1992.

[197] W. Wang and Z.-H. Zhou. Analyzing Co-Training style algorithms. In *Proc. of the 18th European Conference on Machine Learning (ECML 2007)*, page 454465, 2007.

[198] S. Wermter, G. Palm, and M. Elshaw. *Biomimetic Neural Learning for Intelligent Robots*. Springer-Verlag, 2005.

[199] K.-D. Wernecke. A coupling procedure for discrimination of mixed data. *Biometrics*, 48:497–506, 1992.

[200] B. A. Whitehead and T. D. Choate. Cooperative-competitive genetic evolution of radial basis function centers and widths for time series prediction. *IEEE Transactions on Neural Networks*, 7(4):869–880, 1996.

[201] I.H. Witten and E. Frank. *Data Mining: Practical Machine Learning Tools and Techniques with Java Implementations*. Morgan Kaufmann, 1999.

[202] D.H. Wolpert. Stacked generalization. *Neural Networks*, 5(2):241–260, 1992.

[203] K. Woods, W.P. Kegelmeyer, and K. Bowyer. Combination of multiple classifiers using local accuracy estimates. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19:405–410, 1997.

[204] L. Xu, A. Krzyzak, and C.Y. Suen. Methods of combining multiple classifiers and their application to handwriting recognition. *IEEE Transactions on Systems, Man, and Cybernetics*, 22:418–435, 1992.

[205] H. Yang and J. Moody. Data visualization and feature selection: New algorithms for nongaussian data. *Advances in Neural Information Processing Systems*, 12, 1999.

[206] J. Yang and V. Honavar. *Feature Extraction, Construction and Selection A Data Mining Perspective*. Kluwer, 1998.

[207] T.Y. Young and A. Farjo. On decision directed estimation and stochastic approximation. *IEEE Transactions On Information Theory*, pages 671–673, 1972.

[208] H. Yu. SVM selective sampling for ranking with application to data retrieval. In *Proc. of the International Conference on Knowledge Discovery and Data Mining (KDD)*, pages 354–363, 2005.

[209] D. Zhou, O. Bousquet, T. N. Lal, J. Weston, and B. Schölkopf. Learning with local and global consistency. *Advances in Neural Information Processing Systems*, 16:753760, 2004.

[210] Y. Zhou and S. Goldman. Democratic co-learning. In *Proc. of the 16th IEEE International Conference on Tools with Artificial Intelligence (ICTAI'04)*, pages 594–202, Washington, DC, USA, 2004. IEEE Computer Society.

[211] Z.-H. Zhou. When semi-supervised learning meets ensemble learning. In *Proc. of the 8th International Workshop on Multiple Classifier Systems (MCS 2009)*, volume 5519 of *LNCS*, pages 529–538, 2009.

[212] Z.-H. Zhou, K.-J. Chen, and Y. Jiang. Exploiting unlabeled data in content-based image retrieval. In *Proc. of the 15th European Conference on Machine Learning (ECML'04)*, pages 525–536. Springer, 2004.

[213] Z.-H. Zhou and M. Li. Semi-supervised learning by disagreement. *Knowledge and Information Systems*. in press.

[214] Z.-H. Zhou. and M. Li. Semi-supervised regression with Co-Training. In *Proc. of the 19th International Joint Conference on Artificial Intelligence (IJCAI'05)*, pages 908–913, 2005.

[215] Z.-H. Zhou. and M. Li. Tri-training: Exploiting unlabeled data using three classifiers. *IEEE Transactions on Knowledge and Data Engineering*, 17(11):1529–1541, 2005.

[216] Z.-H. Zhou, W. Tang, Zhi hua Zhou, and Wei Tang. Selective ensemble of decision trees. In *Lecture Notes in Artificial Intelligence*, pages 476–483. Springer, 2003.

[217] Z.-H. Zhou and Y. Yu. Adapt bagging to nearest neighbor classifiers. *Journal of Computer Science and Technology*, 20(1):48–54, 2005.

[218] Z.-H. Zhou, D. Zhang, and S. Chen. Semi-supervised dimensionality reduction. In *Proc. of the 7th SIAM International Conference on Data Mining (SDM'07)*, pages 629–634, 2007.

[219] X. Zhu. Semi-supervised learning literature survey. *Technical Report 1530*, 2008.

[220] X. Zhu, Z. Ghahramani, and J. Lafferty. Semi-supervised learning using gaussian fields and harmonic functions. In *Proc. of the 20th International Conference on Machine Learning (ICML'03)*, page 912919, 2003.

[221] X. Zhu, J. Lafferty, and Z. Ghahramani. Combining active learning and semi-supervised learning using gaussian fields and harmonic functions. In *Proc. of the ICML'03 Workshop on The Continuum from Labeled to Unlabeled Data*, 2003.

Index