

Semi-Supervised Bayesian Learning

by

Yuanyuan Guo

**Master of Computer Application Technology, China University of
Geosciences, 2006**

**Bachelor of Computer Science and Technology, China University
of Geosciences, 2003**

**A DISSERTATION SUBMITTED IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR THE DEGREE OF**

Doctor of Philosophy

In the Graduate Academic Unit of Computer Science

Supervisor(s): Huajie Zhang, Ph.D, Faculty of Computer Science, UNB
 Bruce Spencer, Ph.D, National Research Council Canada

Examining Board: Weichang Du, Ph.D, Faculty of Computer Science, UNB
 Yunli Wang, Ph.D, National Research Council Canada
 Donglei Du, Ph.D, Faculty of Business Administration, UNB

External Examiner: Xin Wang, Ph.D, Department of Geomatics Engineering,
 University of Calgary

This dissertation is accepted by the

Dean of Graduate Studies

THE UNIVERSITY OF NEW BRUNSWICK

May, 2012

©Yuanyuan Guo, 2012

Abstract

In traditional supervised learning, classifiers are learned from a number of training data examples where each data example has a label showing the category that the example falls into. Bayesian network classifiers, representing the joint probability and the conditional independencies among a set of random variables, have been widely used in traditional supervised classification. As a special case of Bayesian Networks, Naive Bayes has also been popularly used in many applications such as text classification. However, in many real-world machine learning applications, it may be expensive or time consuming to obtain sufficient labeled data because it needs human efforts to categorize them. On the other hand, it is easier to collect a large amount of unlabeled data. Learning from a small number of labeled data may not generate good classifiers. Semi-supervised learning is one method to deal with the problem by utilizing the unlabeled data to help learn better classifiers. Although various semi-supervised methods have been studied by researchers, there are still some problems remaining unknown in semi-supervised learning. One of the problems is the performance of different Bayesian network classifiers in

semi-supervised learning scenario. An extensive study is needed to get a general picture of the performance of these methods. A second problem is their application on cost-sensitive learning where misclassifying different classes are associated with unequal costs. For example, misclassifying a person with cancer as healthy could be more serious than misclassifying a healthy person as cancerous. Particularly, the classification performance may degrade when the datasets have skewed class distributions. The insufficiency of labeled data makes it more difficult to build a good Bayesian classifier to identify classes having different misclassification costs. Specific techniques are required to make optimal cost-sensitive classification decisions. In supervised learning, many techniques have been applied to solve the cost-sensitive problem. However, only a little work has been conducted in semi-supervised learning and many questions remain unclear in the research area. A third problem is to learn Bayesian network structures in semi-supervised learning scenario. Sufficient training data examples are needed to learn a good Bayesian network structure. It is a big challenge to learn good Bayesian networks when the labeled data is scarce.

This thesis firstly systematically studies the performance of several commonly used semi-supervised learning methods, when different Bayesian network classifiers are used as the underlying classifiers. Several novel and interesting observations are found from the experiments and the performance analysis. Motivated by the observations, an instance selection method is thus presented to improve the classification accuracy when using Naive Bayes in

self-training and co-training methods. The presented method can roughly prevent adding errors in the training data so as to preventing the degradation of accuracy. Then, three cost-sensitive semi-supervised learning methods are presented to improve the performance of Naive Bayes when the labeled data is scarce and different misclassification errors are associated with different costs. Misclassification costs are incorporated into the three methods to generate cost-sensitive classifiers. Experimental results show that, the new methods can obtain lower total misclassification costs than the corresponding opponents. Finally, a new method is designed to learn Bayesian network classifiers with more complex graph structures for classification in semi-supervised learning. The experiments show that, the proposed method achieves higher accuracy while obtaining better Bayesian network structures to represent the relationship among the variables.

Acknowledgements

The completion of this dissertation gives me an opportunity to express my appreciation to all those who have helped me over the years.

First of all, I would like to especially thank my PhD supervisor Dr. Huajie Zhang who has provided great guidance, encouragement, and support in the past four years, not only for my PhD research but also for many aspects of my life. I had limited knowledge on machine learning when I started the PhD study in 2008. Dr. Zhang gave me enlightened advices on different areas including research methodologies and paper writing skills. This work could not have been finished without his supervision and support. I am also indebted to my co-supervisor Dr. Bruce Spencer for his encouragement and professional suggestions on research and career development. Thank Dr. Amira Djebbari for offering me the great internship opportunities in National Research Council. She provided guidance on my research and shared valuable experience with me.

Thanks to the academic committee members. They carefully read my proposal and thesis and provided me great suggestions to improve my work.

Thank the external examiner for reviewing my thesis and providing the valuable suggestions.

Thanks to my former supervisor Dr. Zhihua Cai for his detailed guidance to the research of data mining during my master study. Thanks to my friends Jiang Su, Bin Wang, Liangxiao Jiang, and Jun Du for their selfless help on machine learning related research and for their guidance in life. Thanks to my former co-workers Qu Li, Siwei Jiang, Wenyin Gong, and Xiaobo Liu. Thanks to Sandy Liu, Jidi Zhao, Hui Tang, Weihong Song, and all other friends that I have met in Canada.

I would like to give my deepest gratitude to my family: my parents, my sister, and my brother. Thank them for years of unconditional love, trust, encouragement and support.

Finally, thanks to University of New Brunswick for providing me a wonderful study environment; and thanks to Faculty of Computer Science for all the assistance and opportunities in research, courses study, and social networks.

Table of Contents

Abstract	ii
Acknowledgments	v
Table of Contents	xii
List of Tables	xv
List of Figures	xvii
Abbreviations	xviii
1 Introduction	1
1.1 Motivation	1
1.2 Contributions	6
1.3 Overview	7
2 Background and Literature Review	10
2.1 Semi-Supervised Learning	10
2.1.1 Generative Models	11

2.1.2	Self-Training	12
2.1.3	Co-Training	13
2.1.4	Graph-Based Methods	15
2.1.5	Low-Density Separation Methods	15
2.1.6	Challenges in Semi-Supervised Learning	16
2.2	Cost-Sensitive Learning in Semi-Supervised Classification . . .	18
2.3	Learning Bayesian Network Structures in Semi-Supervised Learning	22
3	An Extensive Empirical Study on Semi-Supervised Bayesian Learning	28
3.1	Methodology	29
3.1.1	Learning Algorithms	29
3.1.2	Datasets	30
3.1.3	Experimental Settings	32
3.2	Performance Comparison among Different Methods	34
3.2.1	On UCI Datasets	34
3.2.2	On Benchmark Datasets	38
3.3	Performance Comparison between Two Selection Methods . . .	40
3.4	Performance Analysis on NB in Standard Self-Training	42
3.4.1	Does the Correctness Rate of Labeling Newly Added Instances Affect the Performance of Semi-Supervised Learning?	44

3.4.2	Does the Amount of Labeled Data Matter for Underlying Classifiers?	48
3.4.3	Does the Amount of Labeled Data Affect the Performance of Semi-Supervised Learning?	49
3.4.4	Does the Initial Performance of the Classifier Affect the Final Performance of Semi-Supervised Learning?	51
3.4.5	Does the Amount of Unlabeled Data Affect the Performance?	51
3.5	Summary	52
4	Instance Selection in Self-Training and Co-Training	54
4.1	Algorithm Description	55
4.1.1	ISBOLD for Self-Training	55
4.1.2	ISBOLD for Co-Training	57
4.2	Experimental Results and Analysis	59
4.2.1	Experimental Settings	59
4.2.2	Results Analysis	59
4.2.3	Learning Curves Analysis	62
4.3	Discussion	64
4.4	Summary	65
5	Cost-Sensitive Semi-Supervised Bayesian Learning Methods	66
5.1	A Study of Semi-Supervised Learning on Class-Imbalanced Data	67
5.2	Analysis of Cost-Sensitive Learning	68

5.3	A Single-Model Cost-Sensitive Self-Training Method	71
5.3.1	Algorithm Description	71
5.3.2	Experiments and Results	74
5.3.2.1	Datasets	75
5.3.2.2	Experimental Settings	76
5.3.2.3	Results on Thirteen UCI Datasets	77
5.3.2.4	Results on Three Text Datasets	80
5.3.3	Summary	82
5.4	A Cost-Sensitive Semi-Supervised Learning Method Based on Bagging	83
5.4.1	Algorithm Description	84
5.4.2	Experiments and Results Analysis	87
5.4.3	Summary	90
5.5	A Cost-Sensitive Semi-Supervised Learning Method Using Mul- tiple Classifiers with Randomly Selected Unlabeled Instances .	91
5.5.1	Algorithm Description	91
5.5.2	Experiments and Results Analysis	94
5.5.3	Summary	96
6	Learning Bayesian Network Structures from Labeled and Unlabeled Data	98
6.1	Two Definitions of Bayesian Networks	99
6.2	Algorithm Description	100

6.3	Experiments on Datasets Generated from Known Bayesian Networks	104
6.3.1	Two Kinds of Bayesian Networks	105
6.3.1.1	Benchmark Bayesian Networks	105
6.3.1.2	Artificial Bayesian Networks	106
6.3.2	Measurements	107
6.3.3	Experimental Settings	108
6.3.4	Results and Analysis	110
6.3.4.1	On Benchmark Bayesian Network Structures .	110
6.3.4.2	On Artificial Bayesian Network Structures . .	114
6.4	Experiments on Datasets with Unknown Bayesian Networks .	115
6.5	Summary	117
7	Conclusion and Future Work	119
7.1	Thesis Summary	119
7.2	Future Work	121
	Bibliography	134
A	Datasets	135
A.1	UCI Datasets	135
A.2	Benchmark Datasets for Semi-Supervised Learning	135
B	Measurements	137
B.1	Classification Performance Measurements	137

Vita

List of Tables

3.1	Accuracy comparison among different classifiers when using “confident selection” and $lp = 10\%$ (part 1)	35
3.2	Accuracy comparison among different classifiers when using “confident selection” and $lp = 10\%$ (part 2)	36
3.3	w/t/l counts of t-test for using the other five classifiers against using NB when $lp = 10\%$ and “confident selection”	37
3.4	Accuracy comparison results for LLGC when $lp = 10\%$	38
3.5	Accuracy comparison results for TSVM when $lp = 10\%$	39
3.6	Accuracy on benchmark datasets (10 labeled data)	39
3.7	Accuracy on benchmark datasets (100 labeled data)	40
3.8	The w/t/l counts of t-test results for using “random selection” against baseline when $lp = 10\%$	41
3.9	The w/t/l counts of t-test results for “random selection” against “confident selection” when $lp = 10\%$	41
3.10	Summary of T-Test results: accuracy of NB built on the labeled data only when lp changes	50

3.11	Summary of T-Test results: accuracy of NB in self-training when lp changes	50
4.1	Accuracy of CF vs ISBOLD in self-training and co-training .	60
4.2	AUC of CF vs ISBOLD in self-training and co-training . . .	61
5.1	Experimental Datasets	75
5.2	Average results of AC on 13 UCI datasets	78
5.3	Average results of TNR on 13 UCI datasets	79
5.4	Average results of AC on three text datasets	81
5.5	Average AC of <i>BaggingCSSSL</i> method	88
5.6	Average TNR of <i>BaggingCSSSL</i> method on 13 UCI datasets .	89
5.7	Average AC of <i>MultiRSCSSSL</i> method	95
6.1	Datasets generated from benchmark Bayesian networks	105
6.2	Datasets generated from artificial Bayesian networks	106
6.3	Average results of 10 runs of 10-fold cross-validation on “Hailfinder- 500”	111
6.4	Average results of 10 runs of 10-fold cross-validation on “Hailfinder- 1000”	111
6.5	Average results of 10 runs of 10-fold cross-validation on “Win95pts- 1000” when $lp = 10\%$	112
6.6	Average results of 10 runs of 10-fold cross-validation on “Insurance- AntiTheft-1000” when $lp = 1\%$	113

6.7 Average results of 10 runs of 10-fold cross-validation on three datasets from artificial networks when $lp = 10\%$ 115

6.8 Average accuracy of 10 runs of 4-fold cross-validation on five UCI datasets when $lp = 1\%$ 116

6.9 Average accuracy on two UCI datasets with fixed given test sets 116

A.1 Description of 26 UCI datasets used in the experiments 136

A.2 Description of six benchmark datasets used in the experiments 136

List of Figures

2.1	An Example of Bayesian Network	23
2.2	An Example of Naive Bayes	25
3.1	Learning curves on the <i>Iris</i> dataset when $lp = 10\%$ and $I = 500$	43
3.2	Learning curves on the <i>vehicle</i> dataset when $lp = 10\%$ and $I = 500$	45
3.3	Learning curves on the <i>colic</i> dataset when $lp = 10\%$ and $I = 500$	46
3.4	Learning curves on the <i>mushroom</i> dataset when $lp = 10\%$ and $I = 500$	48
3.5	Learning curves on the <i>mushroom</i> dataset when $lp = 10\%$ and $I = 3500$	49
4.1	Algorithm of ISBOLD for self-training	56
4.2	Algorithm of ISBOLD for co-training	58
4.3	Learning curves on the <i>vehicle</i> dataset	63
4.4	Learning curves on the <i>kr-vs-kp</i> dataset	64
5.1	Results on class-imbalanced data	68

5.2	Cost matrix for binary-class datasets	69
5.3	Three confusion matrixes	70
5.4	The CS-ST algorithm	73
5.5	The BaggingCSSSL algorithm	85
5.6	The structure of the BaggingCSSSL algorithm	86
5.7	The MultiRSCSSSL algorithm	92
5.8	The structure of the MultiRSCSSSL algorithm	93
6.1	The SSLBN algorithm	101
6.2	The EM method for updating parameters	104
B.1	Confusion matrix for binary-class datasets	138

List of Symbols, Nomenclature or Abbreviations

Σ \sum
 Π \product

Chapter 1

Introduction

1.1 Motivation

In this thesis, we focus on classification task where the class variable has discrete values. In traditional supervised learning, hypotheses are learned from a set of training examples. Each training example has a label which indicates the category that the corresponding example falls into. In some real-world machine learning applications, it is easy to obtain a large amount of data, while labeling them requires expert efforts. For example, in cancer diagnosis, we could collect information from many people but only a few among them have been confirmed to be cancerous or not. Another example is about building a web-page classifier that can automatically classify downloaded web pages. To train the classifier, we would typically rely on human effort to label some web pages, while it is much cheaper to gather hundreds

of millions of unlabeled web pages using a web crawler. Learning classifiers from a small number of labeled training data may not result in good classification performance. Therefore, many researchers have been working on utilizing the abundant but often ignored unlabeled data. Semi-supervised learning uses both labeled data and unlabeled data to learn better classifiers than those trained on the labeled data only. It has been widely used in the areas of text classification, natural language processing, online face recognition, image analysis, medical diagnosis classification, and so on. The general idea of semi-supervised learning is to pick some information from the unlabeled data, and use it with the labeled data to find better classifiers. Various algorithms have been proposed in the semi-supervised learning area such as self-training, co-training, graph-based methods, low-density separation, and so on. Traditional classifiers such as Naive Bayes, decision trees, support vector machines, k-nearest neighbor, and so on, are usually used as the underlying classifier inside the framework of some semi-supervised learning algorithms such as self-training and co-training.

However, there are still some issues to be studied further in semi-supervised learning. One issue in semi-supervised learning is that it is unknown whether the various Bayesian network classifiers work well in the semi-supervised scenario. In addition to Naive Bayes, there are also other Bayesian network classifiers with more complex graph structures that obtain good performance in supervised classification. Some questions have not been well answered, e.g., whether semi-supervised learning methods outperform base classifiers learned

only from the labeled data, when different base classifiers are used; whether selecting unlabeled data with efforts is superior to random selection; and how the quality of the learned classifier changes at each iteration of learning process. It is necessary to conduct an extensive study on the performance of Bayesian network classifiers and present algorithms to deal with the possible shortcomings accordingly. And new algorithms should be presented to improve the performance of Bayesian network classifiers in semi-supervised learning scenario according to the observations of study.

Another issue in classification is the cost-sensitive problem for Bayesian learning where misclassifying different classes causes different costs, which challenges the common assumption that classes have the same misclassification cost. For example, misclassifying a cancerous patient to be healthy is more serious than misclassifying a healthy person as cancerous. In supervised learning, commonly used cost-sensitive learning methods include using sampling methods to change the distribution of training data, incorporating the misclassification cost into the probability estimation to reduce the expected cost, boosting, and reweighing. In semi-supervised learning, due to the insufficiency of labeled data, it would be more difficult to build a good Bayesian classifier to identify classes having different misclassification costs. The main concern is how to make the optimal classification decisions without jeopardizing the performance of semi-supervised learning methods.

A special case is to adapt the algorithms to the application on class-imbalanced datasets. For example, in a binary-class data set, the ratio of positive class

to negative class could be 1:100, which is severely imbalanced. Suppose a learning algorithm assumes or expects a balanced class distribution, when building a classifier on class-imbalanced data, the majority class will take great advantage. The classifier tends to have high accuracy on the majority class and very low accuracy on the minority class. In cancer diagnosis applications, if the accuracy on the minority class is low, it means that most cancerous people (the minority) would be misclassified to be healthy (the majority). It may happen in similar applications such as fraud detection, network intrusion detection, to name a few. Therefore, a good classifier is required to provide high accuracy for the minority class without severely impairing the accuracy of the majority class. When giving different cost values to the classes with different distribution, it is natural to use cost-sensitive learning methods to deal with class-imbalanced datasets.

For Bayesian classifiers, it is still unclear about many basic questions concerning the application of semi-supervised learning methods on cost-sensitive learning problems or class-imbalanced data sets. Moreover, conventional measures such as classification accuracy or error rate alone cannot provide adequate information for performance evaluation. More appropriate standard measure metrics are necessary for fair and conclusive performance evaluation in the presence of unequal misclassification costs.

A third issue is to learn Bayesian network structures that are directed acyclic graphs in semi-supervised scenario. A Bayesian network is a probabilistic graph model representing the conditional independencies and the joint prob-

ability among a set of random variables. It supports probabilistic reasoning from data with uncertainty and thus is a powerful tool in various applications such as modeling, diagnosis, prediction, and so on. For example, some researchers learn Bayesian networks to analyze the structure of regulatory interaction between different genes. Learning the structures of Bayesian networks is NP-hard. As a special case of Bayesian network, Naive Bayes has a simple structure that the class variable is the only parent of each non-class variable. Although Naive Bayes has shown its good performance in classification, the assumption may be violated in some real-world applications. Besides, the simple structure of Naive Bayes may not be capable of modeling complex relationships among the variables. It is necessary to build complex Bayesian network structures for applications that require a sophisticated model to conduct reasoning or inference from data with uncertainty. However, in semi-supervised learning, the insufficiency of labeled data makes it more difficult to discover a good Bayesian network structure from the given data. How to get high classification accuracy and smaller graph structure differences is a challenge in semi-supervised learning, especially when the number of features is large and the number of labeled training data is scarce. Although a number of semi-supervised learning algorithms have been proposed by researchers, only a little work has been done to tackle above issues in semi-supervised learning, especially for Bayesian classifiers. Considering that these issues are common in real-world applications, we believe that studying them would be a promising direction to further advance the performance of

semi-supervised learning.

1.2 Contributions

The main contributions of this thesis are listed as following:

- An extensive empirical study is conducted on the performance of several commonly used semi-supervised learning methods when using various Bayesian network classifiers as the underlying classifier, respectively. Experimental results on a graph-based method and a low density method are also included for comparison. The results show a general picture of how the parameters of the algorithms might affect the performance on the used UCI datasets. Some interesting observations are also found from the experiments.
- An instance selection method is presented to improve the performance of two semi-supervised learning methods, self-training and co-training, when using Naive Bayes as the underlying classifier.
- A set of cost-sensitive semi-supervised learning methods are proposed, including a single-model method using one classifier and two ensemble methods using multiple classifiers, to reduce the misclassification cost when labeled instances are scarce and different misclassification errors are associated with different costs. The methods are applied both on UCI datasets and some text mining datasets.

- We use unlabeled data to help learn complex Bayesian network classifiers. The new method can discover Bayesian network structures with smaller structure differences and high classification accuracy in a semi-supervised learning scenario.

1.3 Overview

The remainder of the thesis is organized as follows.

Chapter 2 introduces semi-supervised learning, cost-sensitive learning, and Bayesian network structure learning, and briefly reviews existing research papers on these topics.

Chapter 3 conducts an extensive empirical study on the performance of several commonly used semi-supervised learning methods when various Bayesian classifiers (Naive Bayes, Naive Bayes Trees, Tree-augmented Naive Bayes, HGC hill-climbing method, Hidden Naive Bayes, and DNB) are used as the underlying classifier, respectively. Results on Transductive SVM and a graph-based semi-supervised learning method LLGC are also studied for comparison. Twenty-six UCI datasets and six widely used benchmark datasets are used in the experiments. One interesting finding is that, for standard self-training and co-training, when selecting the most confident unlabeled instances during learning process, the performance is not necessarily better than that of random selection of unlabeled instances. Another observation is that, adding more unlabeled data might hurt the performance on some UCI

datasets.

Motivated by the observations in Chapter 3, a new instance selection method based on the original labeled data (ISBOLD) is presented in Chapter 4. ISBOLD considers not only the prediction confidence of the current classifier on unlabeled data but also its performance on the original labeled data only. In each iteration, ISBOLD uses the change of accuracy of the newly learned classifier on the original labeled data as a criterion to decide whether the selected most confident unlabeled instances will be accepted to the next iteration or not. Experimental results on 26 UCI datasets show that, ISBOLD can significantly improve accuracy and AUC of self-training and co-training when Naive Bayes is used as the underlying classifier.

In Chapter 5, several semi-supervised learning methods are applied on the datasets with imbalanced class distributions to observe the performance under different settings. The results give some general idea on how self-training works on the datasets with skewed class distributions. Then, several cost-sensitive semi-supervised learning methods are proposed, including a single-model method and two ensemble methods. Some techniques used in supervised learning are brought to semi-supervised learning. Some mathematical analysis is given to show how cost-sensitive learning methods might get further smaller total misclassification costs.

In Chapter 6, we utilize the unlabeled data to help learn Bayesian network classifiers that have high classification accuracy and smaller structure differences. Experiments are conducted both on UCI datasets and some datasets

generated from several benchmark Bayesian networks and artificial networks. Finally, in Chapter 7, we draw the conclusions of the thesis, and discuss possible work that can be extended based on the research work of this thesis.

Chapter 2

Background and Literature Review

2.1 Semi-Supervised Learning

In real-world machine learning applications, labeled data may be scarce and expensive or time-consuming to obtain while unlabeled data is abundant and relatively easy to collect. Learning classifiers on a small set of labeled training data may not produce good performance. Therefore, semi-supervised learning methods have been proposed to effectively utilize the unlabeled data to help learn a strong hypothesis from the limited amount of labeled data [4, 56, 75, 76]. Distinguished by prediction goals, there are two kinds of semi-supervised learning settings: inductive learning, where the goal is to predict unseen data; and transductive learning, where the goal is to pre-

dict the given unlabeled data. Many semi-supervised learning methods have been proposed by researchers, which can generally fall into following categories: generative models, self-training, co-training, graph-based methods, low-density separation, and so on [4, 76]. Follow the common notation in the research community, we represent the set of labeled training instances as L and the set of unlabeled instances as U . The different categories of semi-supervised learning methods will be briefly introduced as follows.

2.1.1 Generative Models

Represent the distribution of unlabeled data as $p(x)$, the distribution of class labels as $p(y)$. The generative models method assumes that, the learned model is a model that, for the joint distribution $p(x, y) = p(y)p(x|y)$, $p(x|y)$ is an identifiable mixture distribution. For example, for a binary class datasets, it could assume that examples of each class follow a Gaussian distribution; then with large amount of unlabeled data, ideally only one labeled data instance is needed per component to fully determine the mixture distribution [76]. The mixture components can be considered as “soft clusters” which separate the data points into different groups following Gaussian distributions.

There are several issues that we should pay attention to when using generative mixture models: (1) whether the mixture model is identifiable; (2) whether the model assumption is correct. If the mixture model is not identifiable, it is difficult to get the mixture distribution from labeled data and

unlabeled data. If the assumed model is wrong, using the unlabeled data may hurt the classification accuracy. Cozman et al. [12] analyze the performance of semi-supervised learning of mixture models and found that unlabeled data may affect the bias and hurt the classification accuracy.

2.1.2 Self-Training

In self-training [71], a classifier is built from L and used to predict labels for the instances in U , and then a certain number of instances in U are selected according to a given selection criterion and moved (with their newly assigned labels) to expand L . These selected instances with labels assigned by the classifier are called “self-labeled” instances in [39]. The whole process iterates until stopped. The stopping criterion in self-training is that either there is no unlabeled instance left or the maximum number of iterations has been reached.

Different self-training methods can be obtained by using different underlying classifiers and selection metrics. Generally, the underlying classifier could be any classification method such as Naive Bayes, decision trees, SVM, k-nearest neighbor, and so on. A commonly used selection criterion is to use confidence of prediction to select the unlabeled instances [71]. During the selection process of self-training, the unlabeled examples are ranked and selected according to the prediction confidence of classifiers and the class distribution. For example, if the positive-to-negative ratio in a binary-class dataset is 3:1, after the unlabeled data are classified by the classifier, three “positive” examples

and one “negative” example with the highest predicted posterior probabilities are selected in each iteration. In this thesis, for simplicity, this selection criterion is denoted as “confident selection”.

Researchers have presented different variants of self-training algorithms. A self-training style method, semi-supervised EM, is presented in [51], which uses all the unlabeled instances in each iteration so that no selection criterion is needed. During each iteration, all the unlabeled instances are assigned predicted class labels and then used to enlarge the training data and to update the classifier. Some researchers also used different selection techniques to decide which unlabeled instances should be used in each iteration. In [66], an adapted Value Difference Metric is presented as the selection metric in self-training. In [39], a data editing method is applied to identify and remove the mislabeled examples from the self-labeled data.

2.1.3 Co-Training

Co-training works in a similar way as self-training except that it is a two-view learning method [2]. Initially, the attribute set (view) is partitioned into two conditionally independent sub-sets (sub-views). A data pool U' is created by randomly choosing some instances from U for each sub-view, respectively. On each sub-view, a classifier is built from the labeled data and then used to predict labels for the unlabeled data in the corresponding data pool. A certain number of unlabeled instances that one classifier has high classification confidence are labeled and moved to expand the labeled

data of the other classifier. And the same number of unlabeled instances are randomly moved from U to replenish U' . Then the two classifiers are rebuilt from their corresponding updated labeled data, respectively. The process iterates until stopped. In other words, in co-training, it iteratively and alternately uses one classifier to help “train” another classifier. The stopping criteria in self-training and co-training are the same.

There are two assumptions in co-training to ensure good accuracy [2]: (1) the two sub-views are conditionally independent of each other given the class; and (2) each sub-view is sufficient to build a good classifier. The two assumptions may be violated in real-world applications. In [50], it is stated that, co-training still works when the attribute set is randomly divided into two separate subsets, although the performance may not be as good as when the attributes are split sufficiently and independently.

The selection criteria in self-training also apply in co-training. In addition to “confident selection”, there are several other metrics proposed by researchers. In [50], co-training is combined with EM to generate a new algorithm co-EM which uses all the unlabeled data in each iteration instead of picking a number of instances out of the data pool. Another kind of methods is to use active learning methods to select unlabeled instances and then ask human experts to label them. Hence, no mislabeled examples will occur, in principle. In [48], an active learning method is used to select unlabeled instances for the multi-view semi-supervised co-EM algorithm. And labels are assigned to the selected unlabeled instances by experts. However, active learning methods

are not applicable if human experts are not available.

2.1.4 Graph-Based Methods

Graph-based methods consider the labeled and unlabeled examples as nodes on a graph and represent the similarity of examples as edges. It can be viewed as estimating a function f on a graph, which should be close to the given labels on the labeled examples and be smooth on the whole graph. It can be expressed in a regularization framework where the first term is a loss function and the second term is a regularizer [76].

Many researchers have proposed different methods on graph construction, as well as selection of the loss function and regularizer. A semi-supervised learning method using Gaussian fields and harmonic functions is given in [77]. In [74], a method learning with local and global consistency (LLGC) is described, which designs a sufficiently smooth function over the graph represented by labeled and unlabeled data. A b -matched graph construction method is presented in [33] that each node on the graph has the same number of edges.

2.1.5 Low-Density Separation Methods

Low-density separation methods aim to find the decision boundary in the low density area. SVM based methods are frequently used [34,41]. A typical algorithm is Transductive SVM that utilizes unlabeled data to regularize the

decision boundary [34].

2.1.6 Challenges in Semi-Supervised Learning

Researchers have published many papers comparing and analyzing the performance of different semi-supervised learning algorithms on different datasets. Naive Bayes (NB) is popularly selected in self-training and co-training due to its good performance in text mining. Some other classifiers are also commonly used such as C4.4, C4.5, k -NN, SVM, and so on. In [2] and [50], some newsgroup datasets for text classification are used in the experiments to demonstrate the performance of NB in standard co-training. Ling et al. use NB in co-training and analyze how it can get better performance on 32 binary UCI datasets [43]. Li and Zhou evaluate their algorithm on 12 binary UCI datasets using the J4.8 decision tree, BP neural networks, and NB [40]. Some UCI datasets have also been used in other papers [39, 44, 70]. In [4], eight benchmark datasets are introduced and used for performance comparison of different semi-supervised learning methods. Although much work has been done, it is not quite clear about the performance comparison among several commonly used standard semi-supervised learning algorithms, when using state-of-the-art non-naive Bayesian classifiers that may obtain better performance than NB in the supervised learning setting.

In addition, researchers have been interested in whether unlabeled data can always help semi-supervised learning or under what circumstances unlabeled data can be more helpful. In [11], Cozman and Cohen point out that un-

labeled data may degrade classification performance, not only in somewhat extreme conditions but also under common assumptions when the model assumptions are incorrect. Their study is based on the assumption that the labeled data and unlabeled data come from the same distribution. In [5], Chawla and Karakoulas conduct an empirical study to examine the effect of labeled data and unlabeled data in four methods (co-training, Reweighting, ASSEMBLE and Common-component mixture with EM) when each dataset has an extremely imbalanced class distribution and the distributions of data points in labeled and unlabeled datasets are different. AUC was adopted as the performance metric. Some main conclusions drawn in their paper are: (1) Semi-supervised learning techniques may not consistently provide an improvement over supervised learning for all different ratios of labeled data to unlabeled data; (2) the amount of labeled and unlabeled data is domain dependant and no consistent conclusions can be drawn from the experiments; and (3) with sample-selection bias, more unlabeled data may help when a better representation of the data can be induced. However, their conclusions only show the final comparison results. No detailed analysis of performance during the learning process was given in their work. Moreover, their work only used Naive Bayes as the base classifier for the four methods, and C4.5 as another base classifier in the co-training experiments. It is not clear that whether using other non-naive Bayesian classifiers will have similar problems. Despite all the work that has been done on semi-supervised learning methods, three questions listed as follows are not quite clear: (1) It is not clear whether

standard semi-supervised learning methods can always outperform the classifiers built only on the small amount of labeled data, especially when different non-naive Bayesian classifiers are used as the base classifier, respectively. (2) It has not been indicated whether selecting unlabeled instances with efforts to enlarge the training set can have better performance than does randomly expanding the training set. (3) No detailed analysis of learning curves or correctness rate of labeling unlabeled instances has ever been conducted to examine the performance of semi-supervised learning methods during each iteration of learning process. These questions should be systematically studied to provide a general picture and some guidance to researchers in this area. New algorithms will be presented to deal with the problems found from the study accordingly.

2.2 Cost-Sensitive Learning in Semi-Supervised Classification

To better understand the cost-sensitive problem, costs for classifying the classes are described here. Let $C(i, j)$ be the misclassification cost of predicting an instance to belong to class i when the true class is j . If the prediction is correct (that is, $i = j$), then $C(i, j) = 0$; otherwise $C(i, j) > 0$. As depicted in [17], the optimal prediction for an example x is the class i

that minimizes the expected cost computed by

$$L(x, i) = \sum_j P(j|x)C(i, j) \quad (2.1)$$

where $P(j|x)$ is the prediction probability of the class j given the instance x . A common measure to evaluate the performance of a cost-sensitive learning method is the total cost which is the sum of misclassification costs for each class on a given testing dataset. Another measure is the average misclassification cost, computed by dividing the total cost by the number of instances in the testing dataset. A good classifier should be able to correctly identify more instances and reduce the misclassification cost.

In supervised learning, researchers have proposed to change the class distribution of training data in cost-sensitive learning. In [16], it is concluded that under-sampling beats over-sampling when applying C4.5 to class imbalance and cost-sensitive problems. In over-sampling some examples of the minority class are added into the training data, while under-sampling technique deletes some examples of the majority class from the training data. Another popular method is to modify the learning algorithms. However, based on the investigation of the behavior of Bayesian and decision tree learning methods, it is concluded that re-balancing the training examples has little effect on learned classifiers [17]. The authors suggest learning classifiers from the original data but using either one of the two equations given in the paper which incorporates the misclassification costs into the probability estima-

tion. Other methods for cost-sensitive learning include the k-nearest neighbor based method that assigns bigger weights to high-cost examples, boosting [18] [63], cost-sensitive GP [38] and so on. The results of cost-sensitive learning can also be utilized for ranking by using calibration methods to obtain a ranked list [19] [72].

In the semi-supervised learning scenario where the number of labeled training data is small, some techniques such as under-sampling cannot be used. There is limited work done on the cost-sensitive problem in semi-supervised learning. In [55], a decision tree classifier with smoothing is used as the underlying classifier. An EM procedure is applied to iteratively assign labels to the unlabeled instances and learn the classifier on the combination of the labeled data and the updated unlabeled data. When assigning labels to the unlabeled data, the estimated “optimum” label with the smallest conditional risk (expected cost) is assigned to an unlabeled instance, and the corresponding expected cost is normalized and used as the weight of the unlabeled instance. The C4SVM algorithm is presented in [42], which incorporates misclassification costs into the optimization function of a semi-supervised SVM using label means. Active learning has been applied to cost-sensitive learning problems as well [15, 44, 46]. In [46], misclassification costs are added into the loss function of active learning to pick the most informative unlabeled instance and then labels are inquired from experts. In [44], in each iteration, uncertainty sampling is used to select some unlabeled instances and labels are later assigned to those selected instances; the selected instances are moved

to the labeled data and a classifier is built on the expanded labeled data; and then a cost-sensitive classifier is built on the combination of the expanded labeled data and all remaining unlabeled instances with self-labeled labels. Donmez and Carbonell [15] also use active learning method but propose a method for learning from multiple imperfect oracles. The active learning methods require interaction with experts, which might be infeasible to some applications if experts are not available.

A special case of cost-sensitive learning is that the dataset has a skewed class distribution, which is also called class-imbalanced problem. In supervised learning, current techniques for class-imbalanced learning can be generally summarized into several categories: sampling methods, cost-sensitive methods, kernel-based methods, active learning methods and one-class learning [25]. Sampling methods adopt various random or informative over-sampling and under-sampling techniques to balance the class distribution [32, 53]. In [31], experiments are conducted to compare seven sampling techniques with 11 commonly-used learning algorithms on 35 benchmark data sets to help researchers understand the strengths and weaknesses of the techniques. Cost-sensitive methods usually target the imbalanced learning problem by using different cost matrices that describe different costs for misclassifying the examples. There is a strong connection between cost-sensitive learning and class-imbalanced learning; hence many cost-sensitive learning methods were naturally applied to class-imbalanced learning problems [45] [62].

The class-imbalanced problem also triggered interest in the semi-supervised

learning community. In [54], EM mixture Gaussian, TSVM and semi-supervised learning using Gaussian random fields are compared on seven UCI data sets with different levels of imbalance: the percentage of the minority class ranges from 3.9% to 50%. Error rate and AUC are used as the measure metrics in the paper. In [30], the final outputs of two semi-supervised learning methods are normalized and the error rates of the two methods are compared on a data set. A method that combines under-sampling with self-training method is shown in [24]. An extreme under-sampling technique is used to randomly change the class distribution of the labeled training data. Then, it applies self-training to the new training data, together with a set of unlabeled data specifically required from the web. Only accuracy is evaluated and compared in the paper. In [37], some important issues of semi-supervised learning under the conditions of imbalanced data sets are mentioned, including how to identify whether an unlabeled data example came from a balanced or imbalanced underlying distribution, what kind of bias to be introduced, and so on.

2.3 Learning Bayesian Network Structures in Semi-Supervised Learning

A Bayesian network (BN) is an acyclic directed graph (DAG). On the graph, each node X_i ($i = 1, \dots, n$) represents a random discrete variable in the domain. An edge $X \rightarrow Y$ on the graph represents a parent-child relation in

which X is a parent of Y . All parents of Y consist the parent set of Y which can be denoted as Π_Y . An example of a Bayesian network with five variables is shown in Figure 2.1.

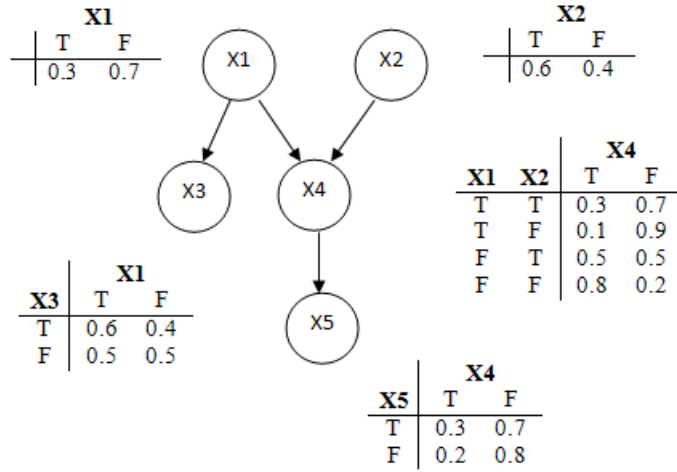


Figure 2.1: An Example of Bayesian Network

In addition to the graph structure, each node has a conditional probability table (CPT) that specifies the probability of each possible state of the node given each possible combination of states of all the nodes in its parent set. Based on the Bayesian rules, the joint probability distribution represented by a Bayesian network can be expressed as

$$P(X_1, X_2, \dots, X_n) = \prod_i P(X_i | \Pi_{X_i}) \quad (2.2)$$

For simplicity, $X_i = k$ is used to represent that the i -node takes the k -th state in its value domain, $\Pi_{X_i} = j$ specifies that Π_{X_i} takes the j -th combinational

state, and N_{ijk} denotes the counts of $X_i = k$ and $\Pi_{X_i} = j$. The conditional probability $P(X_i = k | \Pi_{X_i} = j)$, also called parameter θ_{ijk} , is computed from a given complete dataset by the following equation:

$$P(X_i = k | \Pi_{X_i} = j) = \theta_{ijk} = \frac{N_{ijk}}{\sum_k N_{ijk}} \quad (2.3)$$

Bayesian networks are popular in the community of machine learning and artificial intelligence because they are able to support probabilistic reasoning from data with uncertainty. With a given Bayesian network, we can apply probabilistic inference methods to compute the probability of the states of some variables when other variables are observed [28]. Hence, Bayesian network is a powerful tool in modeling complex systems such as the gene regulatory networks in Bioinformatics [22], medical diagnostic system [52], printer troubleshooting, and so on.

A special case of Bayesian network classifier is Naive Bayes (NB). An example of a Naive Bayes with three non-class variables and a class variable is shown in Figure 2.2. There is a conditional independence assumption in Naive Bayes stating that, given the class node, the other nodes are conditionally independent with each other. Therefore, in a Naive Bayes classifier, each non-class node has the class node as the only parent and the class node has no parent. The joint probability distribution represented by a Naive Bayes

classifier is described as

$$P(X_1, X_2, \dots, X_{n-1}, C) = P(C) \prod_{i=1}^{n-1} P(X_i|C) \quad (2.4)$$

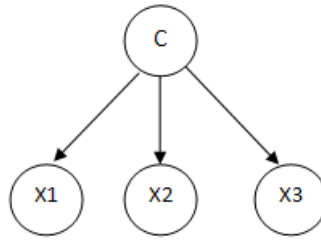


Figure 2.2: An Example of Naive Bayes

Due to the simplicity of structure and implementation as well as the good classification performance, Naive Bayes has been widely applied in supervised classification tasks [47]. Researchers have presented some variants of NB that relax the conditional independence assumption. NBTree [35] is a hybrid method combining decision tree and Naive Bayes. Tree-augmented Naive Bayes (TAN) [21] is an improved *ChowLiu* algorithm for learning tree-augmented Naive Bayes, where each non-class node can have one augmenting edge that points to it from another non-class node. Hidden Naive Bayes (HNB) [73] is an improved Bayesian model that each attribute has a hidden parent created by the average of weighted one-dependence estimators. DNB [61] learns parameters of Bayesian networks by discriminatively computing frequencies from data, hence integrating the advantage of both generative and discrimi-

native learning. In [60], a full Bayesian network classifier is presented which uses a full Bayesian network as the structure and learns a decision tree for each CPT.

Generally speaking, learning Bayesian networks from data consists of two steps: structure learning which discovers the graph structure from data; and parameter learning which computes values in the conditional probability tables associated with the known structure [49] [27]. Existing structure learning methods generally fall into two main categories [6]: the dependency analysis method which uses the results of statistical dependency tests as constraints and guidance to construct the Bayesian network structure; and the score-and-search method that uses a scoring metric to evaluate candidate network structures and a search method to find the structure with the best score [26] [36]. Since searching for the network structure is NP-Hard [7], heuristic search algorithms such as greedy hill-climbing and genetic algorithms are typically used. HGC (greedy hill-climbing) [27] is a Bayesian network structure learning algorithm using hill climbing as the search method. Greedy hill-climbing starts with a random network structure G and iteratively searches for better candidate structures. At each iteration, the entire neighborhood of current G is searched and possible local changes are evaluated. The candidate network structure that has the largest degree of improvement is picked as the new structure. This process is repeated until the current solution is not significantly different from the previous one. Greedy hill-climbing may get stuck in local maxima and heuristics such as random restarts, simulated

annealing and tabu search can be used to escape from the local maxima. Some researchers work on using genetic algorithms to learning Bayesian networks [58] [20] [68] [69]. With a known Bayesian network structure, the parameters can be computed directly from the given dataset or be estimated by Bayesian inference algorithms [26] [49] [28].

Naive Bayes has been widely applied for text classification in self-training and co-training [2] [51] [59]. However, the conditional independence assumption may be violated in some real-world applications. For some complex systems, Bayesian networks should be used instead of Naive Bayes to better model the conditional dependencies among the variables. When the labeled training data examples are scarce, they cannot provide enough information to learn good Bayesian networks. On the other hand, useful information can be found from the unlabeled data. Cohen et al. [10] present a stochastic structure search algorithm for learning the structures of Bayesian network classifiers using both labeled and unlabeled data. How to utilize the information contained in the unlabeled data to find Bayesian networks with good graph structures and high classification accuracy is an interesting topic.

Chapter 3

An Extensive Empirical Study on Semi-Supervised Bayesian Learning

In this chapter, we conduct an extensive empirical study on semi-supervised learning. Different from existing research work, we compare the classification accuracy of standard self-training, co-training and co-EM algorithms using six Bayesian classifiers (NB, NBTree [35], TAN [21], HGC [27], DNB [61], and HNB [73]), as well as TSVM and LLGC. The experiments are conducted on 26 UCI datasets and 6 widely used benchmark datasets containing binary-class datasets and multi-class datasets, and the labeled data and the original training data have the same class distribution. To examine the effectiveness of selecting particular unlabeled instances, we run experiments on ran-

domly selecting unlabeled instances during semi-supervised learning process for comparison. Learning curves are also drawn to analyze the effect of the amount of labeled and unlabeled data on classification performance. The correctness rate of newly added unlabeled instances in each iteration of the standard self-training method is also shown in a scatter plot.

3.1 Methodology

3.1.1 Learning Algorithms

The algorithms used in the experiments include the standard self-training, co-training and co-EM, as well as TSVM and LLGC. Self-training, co-training, and co-EM can be used for both inductive learning and transductive learning, while TSVM and LLGC are used for transductive learning in nature. As stated in Chapter 2, there are two assumptions in co-training to make sure it has good accuracy, which may be violated in practical applications. However, co-training still helps when the feature set is randomly separated into two sub-sets, although the performance may not be as good as when the features are split sufficiently and independently [50]. In our experiments, for co-training and co-EM, we randomly split the attributes into two subsets.

Several instance selection metrics have been proposed in self-training and co-training, including “confident selection” and other methods described in Section 2.1.2 and Section 2.1.3. However, it is not clear whether selecting unlabeled instances with efforts can outperform simple random selection. For

comparison, we also conducted experiments that randomly select unlabeled instances instead of selecting the most confident ones. For simplicity, the random selection method is denoted as “random selection”.

The performances of self-training, co-training and co-EM are compared among using the six different base classifiers: Naive Bayes (NB), NBTree [35], TAN [21], HGC [27], HNB [73], and DNB [61]. Performances of TSVM and LLGC are also reported for comparison.

3.1.2 Datasets

In the experiments, we use 26 UCI datasets from WEKA web site ¹ and 6 benchmark datasets from [4]. Properties of the datasets are shown in Table A.1 and Table A.2 of Appendix A, respectively.

The 26 UCI datasets, including 18 binary class datasets and 8 multi-class datasets, were originally downloaded from a package of 37 classification problems, “datasets-UCI.jar”, from WEKA web site. We only use 26 datasets out of the package because the other 11 datasets have extremely skewed class distributions. For example, in the *hypothyroid* dataset, the frequency of each class value is 3481, 194, 95 and 2 respectively. When randomly sampling the labeled data in semi-supervised learning, the classes that have very small values of frequency may not appear in some generated datasets if we want to keep the same class distribution. Usually the minority class can be merged into a majority class, or instances from the minority class can simply be

¹<http://www.cs.waikato.ac.nz/ml/weka/>

deleted if the amount is very small. However, to minimize any possible influence, we do not use those 11 datasets with extremely skewed class distributions. We preprocessed each dataset in WEKA software [67] by removing any attribute that its number of attribute values is almost equal to the number of instances in the dataset, replacing missing values for nominal and numeric attributes with the modes and means, and discretizing each numeric attribute by the ten-bin binning method. Attributes “*Hospital Number*” (in *colic.ORIG* dataset) and “*Instance_name*” (in *splice* dataset) were removed because they contribute little to classification.

Details of the six benchmark datasets are described in the book “*semi-supervised learning*” [4]. For each benchmark dataset, the number of sampled labeled instances is either 10 or 100. Twelve subsets of labeled data and unlabeled data are available for each combination of dataset and number of labeled instances. These labeled datasets and unlabeled datasets are provided by the book². In each labeled data, there is at least one instance from each class. While “USPS” has imbalanced class distributions with relative size of 1:4, the other five datasets have balanced class distributions. These datasets are suitable for transductive learning.

²The datasets can be download from <http://olivier.chapelle.cc/ssl-book/benchmarks.html>

3.1.3 Experimental Settings

We implemented the standard self-training, co-training and co-EM methods in WEKA³. All experiments in this study utilize the implementations of NB, NBTree, TAN, HGC and HNB from WEKA release 3.7.0 [67]. In our experiments, similar to [61], the maximum number of parents for each node is set to 2 in HGC, and NB is used as the underlying classifier in DNB. The maximum number of iterations in self-training or co-training, represented by I , is set to 80. The maximum number of iterations in co-EM is set to 30. The size of data pool in co-training is set to 50% of U . For TSVM and LLGC, default parameters as given in their implementation are used in our experiments. Only binary-class datasets are used for TSVM following its code implementation provided by the author of TSVM method [34].

The performance measure is accuracy that is commonly used to evaluate semi-supervised learning methods. In order to examine whether unlabeled data really help improve the classification accuracy, for self-training, co-training and co-EM, performance of corresponding base classifiers which are learned from the original labeled data only are shown as the baseline results. For TSVM, performance of SVM learned from the original labeled data is used as the baseline. For LLGC, to follow the work in [74], performance of 1-NN learned from the original labeled data is used as the baseline.

³The DNB code is from authors of [61] for academic research. TSVM software is available from <http://svmlight.joachims.org/>. The LLGC code is downloaded from <http://www.scms.waikato.ac.nz/fracpete/projects/collective-classification/> and modified to our experimental settings.

Experiments on two kinds of datasets are conducted differently. The details are given as follows.

1. For each UCI dataset, ten runs of four-fold stratified cross-validation are conducted. For each run, firstly, the positions of instances in the dataset are randomized; then the dataset is split into four subsets. The training process repeats four times (folds) where at each time a different subset is kept as the testing set and the other three subsets are combined together as the training set. In each fold, 25% of the original data are kept aside as the testing set to evaluate the performance of learning algorithms, which actually follows the data splitting ratio in [2,39,43]; the remaining 75% data are randomly partitioned into labeled data L and unlabeled data U according to a given percentage of labeled data (denoted as lp in the thesis and $lp = |L|/(|L| + |U|)$). Suppose lp is 10%, then 25% data is kept as the testing set, 10% of the 75% data is randomly sampled as L while the remaining 90% of the 75% data is used as U . Different values are set to lp in the experiments ranging from 5% to 100%. When generating L , some techniques are used to make sure that L and the original training data have the same class distribution. Noticeably, for TSVM and LLGC, the testing set is included in U because these two methods are transductive. The average accuracy of ten runs of four-fold stratified cross-validation for each method on each UCI dataset will be discussed in following sections.

2. For each benchmark dataset, each semi-supervised learning method are applied in transductive setting. That is, classifiers learned from labeled and unlabeled data are evaluated on the corresponding unlabeled data. The average accuracy on twelve subsets of each benchmark dataset will be shown in following sections.

3.2 Performance Comparison among Different Methods

In this section, performance among different semi-supervised learning methods on two kinds of datasets are compared and analyzed. The performance of the corresponding classifier built on the initial labeled data is used as the baseline performance. The results show whether each semi-supervised learning method can perform better than the baseline.

3.2.1 On UCI Datasets

Table 3.1 and Table 3.2 show the average results of using different Bayesian classifiers in self-training, co-training and co-EM on UCI datasets when lp is 10%. “Confident selection” is used to select the most confident unlabeled data during self-training and co-training iterative learning process. Each figure displays the average accuracy of ten runs of four-fold stratified cross-validation. The figures in each “base” column represent the baseline per-

Table 3.1: Accuracy comparison among different classifiers when using “confident selection” and $lp = 10\%$ (part 1)

Dataset	NB				NBTree				TAN			
	base	selftrain	cotrain	coEM	base	selftrain	cotrain	coEM	base	selftrain	cotrain	coEM
balance-scale	77.79	63.73 *	67.49 *	46.08 *	73.18	62.09 *	60.19 *	46.08 *	68.42	60.39 *	59.69 *	46.08 *
breast-cancer	70.41	70.55	71.46	70.32	70.41	66.57	71.04	70.35	64.99	64.72	71.19	70.28
breast-w	96.24	96.94	96.81	97.31	96.24	96.67	96.69	97.25	91.63	93.26	95.81 v	95.82 v
colic	76.85	75.57	77.77	66.63 *	76.85	81.33	77.34	69.92	70.92	78.04	78.86	71.01
colic.ORIG	67.23	55.24 *	56.90 *	58.64 *	67.23	58.26 *	59.43 *	65.57	63.04	59.27	59.92	65.54
credit-a	83.19	82.82	82.82	59.10 *	84.55	83.70	80.82	57.59 *	76.81	80.61	76.59	57.26 *
credit-g	71.14	63.27 *	65.73 *	69.81	68.25	66.00	66.83	69.94	69.61	66.75	67.62	69.99
diabetes	71.43	71.54	68.31	65.09 *	69.45	69.91	68.55	65.10	67.19	67.70	69.19	65.22
heart-c	79.01	82.11	82.80	77.62	79.01	80.04	78.74	67.37	68.84	78.94 v	72.83	66.46
heart-h	81.43	83.34	81.43	74.24	81.43	81.94	81.77	70.42 *	72.73	81.81 v	78.24	67.67
heart-statlog	79.30	81.52	82.51	56.37 *	79.30	78.74	77.37	57.55 *	67.82	73.04	73.30	54.96 *
hepatitis	81.75	83.31	82.07	81.23	81.75	83.10	80.91	79.42	76.78	82.01	80.46	79.16
ionosphere	83.88	84.93	84.81	80.28	83.88	83.67	85.81	83.79	80.77	83.16	84.67	84.87
iris	83.10	91.18	83.45	81.21	83.10	91.11	83.93	80.79	59.89	80.47 v	77.83 v	74.49
kr-vs-kp	83.64	73.22 *	66.17 *	51.57 *	92.36	92.59	78.39 *	54.92 *	89.97	81.03 *	77.48 *	54.51 *
labor	79.88	90.20	82.90	68.57	79.88	87.67	81.31	67.68	75.13	83.21	75.05	68.68
letter	62.99	46.28 *	44.56 *	28.37 *	63.58	61.95	54.34 *	21.13 *	66.93	64.10 *	56.98 *	34.54 *
mushroom	93.50	92.80 *	92.70 *	89.68 *	99.49	99.40	99.11	97.44 *	99.68	99.63	99.09 *	93.80 *
segment	80.45	66.27 *	64.55 *	63.26 *	78.65	78.79	69.77 *	55.95 *	80.74	80.28	71.49 *	59.10 *
sick	95.13	92.90 *	93.58 *	93.69 *	96.80	95.69	94.16 *	93.87 *	96.09	93.63 *	93.73 *	93.87 *
sonar	64.71	57.60	57.98	53.94 *	64.71	59.62	59.71	55.43	58.80	61.35	58.13	55.67
splice	91.63	85.57 *	80.38 *	73.73 *	90.31	83.25 *	73.48 *	68.85 *	77.57	76.18	73.82	63.42 *
vehicle	55.03	44.47 *	44.60 *	44.15 *	55.04	51.28	49.07	50.07	57.21	54.85	53.49	52.59
vote	89.66	88.30	88.09	88.35	90.76	89.59	90.49	89.04	91.38	92.16	91.15	89.40
vowel	35.05	21.76 *	24.07 *	14.04 *	36.45	24.89 *	27.15 *	17.13 *	29.18	28.50	26.32	22.44 *
waveform-5000	78.75	80.27 v	76.55 *	55.63 *	78.47	80.11	70.49 *	40.39 *	71.27	73.18	63.28 *	42.55 *
w/t/l	-	1/14/11	0/14/12	0/10/16	-	0/22/4	0/17/9	0/14/12	-	3/19/4	2/17/7	1/14/11

formance. The last row shows the two-tailed paired t-test results under the significant level of 95%. Each entry “ $w/t/l$ ” means that the semi-supervised learning method in the corresponding column wins on w datasets (marked by ‘v’), ties on t datasets, and loses on l datasets (marked by ‘*’) against corresponding “base”.

It can be observed from the results in Table 3.1 and Table 3.2 that, on most UCI datasets, the standard self-training algorithm cannot improve classification accuracy by utilizing unlabeled data. The performance of using NB

Table 3.2: Accuracy comparison among different classifiers when using “confident selection” and $lp = 10\%$ (part 2)

Dataset	HGC				DNB				HNB			
	base	selftrain	cotrain	coEM	base	selftrain	cotrain	coEM	base	selftrain	cotrain	coEM
balance-scale	61.91	61.58	46.42 *	46.08 *	75.76	63.62 *	67.31 *	46.08 *	69.81	59.91 *	54.88 *	46.22 *
breast-cancer	62.44	64.52	71.04 v	70.52 v	63.28	62.84	69.05	70.52	63.94	63.36	67.89	70.77
breast-w	95.25	96.18	97.00	97.05	94.86	95.72	96.59	97.05 v	88.67	91.17	92.10 v	71.10 *
colic	73.72	78.04	80.33	66.17 *	74.70	78.72	78.89	73.40	73.67	76.25	77.39	69.37
colic.ORIG	60.90	60.41	62.36	64.05	66.85	61.66	60.65	64.05	66.60	56.85 *	59.76	64.65
credit-a	82.29	83.09	76.85	56.79 *	79.91	80.29	77.51	61.89 *	77.06	79.04	72.27	65.33
credit-g	67.20	65.93	68.99	69.99	69.80	68.17	66.73	69.94	70.48	66.59	67.22	69.89
diabetes	69.48	71.32	67.76	65.36	69.82	70.57	69.86	65.04	68.46	70.55	69.06	65.79
heart-c	71.21	79.50	76.80	70.53	74.22	78.25	77.97	69.56	72.50	79.80 v	74.44	72.01
heart-h	75.78	81.30	77.71	69.69	77.82	80.69	79.12	74.40	75.64	81.60	75.34	69.97
heart-statlog	70.33	79.29	75.78	67.26	75.34	78.23	79.29	63.02	75.71	77.74	76.15	74.19
hepatitis	74.14	80.21	79.88	79.22	80.65	84.46	81.75	80.00	80.12	80.47	79.29	79.42
ionosphere	82.99	84.30	85.52	85.70	84.47	85.73	84.56	80.71	85.98	89.91	88.68	83.90
iris	82.72	93.20	81.20	77.08	84.01	91.86	82.84	80.79	70.88	82.45	61.27	38.94 *
kr-vs-kp	90.63	81.03 *	77.94 *	54.01 *	93.03	92.94	76.40 *	52.24 *	89.19	80.92 *	75.63 *	52.70 *
labor	72.81	86.37	77.56	69.88	81.52	89.89	84.18	77.87	81.31	84.71	77.99	73.33
letter	68.74	68.12	56.67 *	27.66 *	67.10	66.66 *	45.53 *	26.09 *	74.74	71.18 *	60.89 *	38.50 *
mushroom	99.75	99.67	99.14 *	94.36 *	99.69	99.68	99.25 *	93.47 *	99.79	99.79	99.69	95.22 *
segment	86.54	83.34	70.17 *	60.37 *	85.58	85.34	69.71 *	63.34 *	87.65	86.62	74.58 *	63.70 *
sick	96.98	94.36 *	93.83 *	93.88 *	96.66	96.57	93.79 *	93.86 *	97.03	94.88 *	93.81 *	93.88 *
sonar	58.94	65.05	54.23	51.06	64.47	66.49	64.13	59.66	66.20	62.69	62.69	60.82
splice	93.86	89.78 *	83.21 *	59.54 *	90.34	90.41	78.29 *	68.22 *	91.97	87.46 *	81.54 *	68.07 *
vehicle	51.92	53.80	47.59	44.98	55.03	55.51	48.71	44.94 *	60.05	56.81	56.20	58.81
vote	92.51	92.65	91.36	89.50	93.36	93.15	90.49	89.03 *	92.94	91.70	90.48	89.42 *
vowel	40.92	38.51	30.40 *	15.67 *	37.00	33.22	25.97 *	21.79 *	40.79	35.85 *	32.13 *	31.32 *
waveform-5000	79.67	81.22 v	77.65	41.55 *	77.89	78.07	70.46 *	46.30 *	79.88	81.26 v	74.35 *	65.64 *
w/t/l	-	1/22/3	1/17/8	1/14/11	-	0/24/2	0/17/9	1/13/12	-	2/17/7	1/17/8	0/14/12

in self-training improves only on one dataset but degrades on 11 datasets. When using TAN or HNB in self-training, it wins slightly on more datasets over “base” than using NB. When using HGC or DNB in self-training, it loses on less datasets over “base” than that of using NB. By observing the performance of co-training, it is found that, the standard co-training algorithm does not improve classification accuracy on most UCI datasets, neither. In the standard co-training, using TAN or HGC or HNB improves only on one or two datasets, slightly better over “base” than using the three other

Table 3.3: w/t/l counts of t-test for using the other five classifiers against using NB when $lp = 10\%$ and “confident selection”

	NBTree	TAN	HGC	DNB	HNB
self-training	5/21/0	7/15/4	9/17/0	9/16/1	8/16/2
co-training	3/21/2	5/17/4	5/20/1	3/22/1	7/13/6
co-EM	3/22/1	4/20/2	1/23/2	2/24/0	4/19/3

classifiers. For standard co-EM, the performance is even worse on these UCI datasets.

To compare performance among using different classifiers in each of the three semi-supervised learning method, Table 3.3 displays the “w/t/l” counts of t-test for using each other classifier against using NB, respectively. The first row, results of self-training, indicates that, on the UCI datasets, using the other five Bayesian classifiers generally perform a bit better than using NB. The second row and the third row show that, for co-training and co-EM, using the other five Bayesian classifiers is closer to using NB.

In Table 3.4, the comparison results of LLGC against 1-NN are shown. When the downloaded LLGC code was run on the “letter” and “mushroom” datasets, it did not give any results which are hence omitted in the table. The t-test results show that LLGC improves classification accuracy on two UCI datasets but degrades it on 17 UCI datasets. In Table 3.5, the average results of TSVM and SVM on 18 binary-class UCI datasets are displayed. Results indicate that TSVM improves classification accuracy on 10 UCI datasets and degrades it on seven datasets.

It is interesting to find that these methods hardly improve classification ac-

Table 3.4: Accuracy comparison results for LLGC when $lp = 10\%$

Dataset	1-NN	LLGC
balance-scale	62.59	77.22 v
breast-cancer	62.82	70.28
breast-w	92.46	65.52 *
colic	68.86	63.04
colic.ORIG	64.97	66.30
credit-a	75.13	55.58 *
credit-g	65.44	70.00 v
diabetes	64.84	65.10
heart-c	72.70	54.46 *
heart-h	75.55	63.95 *
heart-statlog	74.08	55.55 *
hepatitis	79.02	79.35
ionosphere	83.68	64.10 *
iris	81.21	39.20 *
kr-vs-kp	80.31	52.22 *
labor	80.23	64.88 *
segment	83.15	15.90 *
sick	95.49	93.88 *
sonar	65.19	53.37 *
splice	66.28	51.88 *
vehicle	58.33	26.46 *
vote	90.78	61.38 *
vowel	39.61	10.35 *
waveform-5000	60.04	33.84 *
vs 1-NN: w/t/l	-	2/5/17

curacy on most of the 26 UCI datasets, a finding that contradicts the core idea of semi-supervised learning.

3.2.2 On Benchmark Datasets

Table 3.6 and Table 3.7 display average accuracies of NB in self-training and co-training on benchmark datasets when 10 or 100 labeled instances are used. Since the benchmark datasets have numeric attributes, we only conducted experiments using NB that can deal with numeric attributes. The results of SVM and TSVM are from [4]. Improved performance of TSVM over SVM

Table 3.5: Accuracy comparison results for TSVM when $lp = 10\%$

Dataset	SVM	TSVM
breast-cancer	0.70	0.64 *
breast-w	0.96	0.97 v
colic	0.64	0.68 v
colic.ORIG	0.66	0.59 *
credit-a	0.66	0.60 *
credit-g	0.70	0.64 *
diabetes	0.70	0.71 v
heart-c	0.73	0.74 v
heart-h	0.67	0.76 v
heart-statlog	0.79	0.77 *
hepatitis	0.79	0.78 *
ionosphere	0.65	0.75 v
kr-vs-kp	0.86	0.89 v
labor	0.65	0.85 v
mushroom	0.88	0.86 *
sick	0.94	0.97 v
sonar	0.58	0.58
vote	0.93	0.94 v
vs SVM: w/t/l	-	10/1/7

and self-training or co-training over “base” is represented by “v” following the figures.

It can be observed that, when 10 labeled data is used, self-training gets improved performance on three datasets while co-training on two datasets. When 100 labeled data is used, both self-training and co-training can only improve the accuracy on one dataset. TSVM improves average accuracy on

Table 3.6: Accuracy on benchmark datasets (10 labeled data)

Dataset	base(NB)	selftrain	cotrain	SVM	TSVM
g241c	51.15	61.50 v	55.28 v	52.86	75.29 v
g241d	50.84	49.54	49.69	53.34	49.92
Digit1	70.44	89.33 v	88.06 v	69.40	82.23 v
USPS	80.20	75.26	73.56	79.97	74.80
COIL	22.44	22.75 v	19.81	31.64	32.50 v
BCI	50.47	49.21	48.74	50.15	50.85 v

Table 3.7: Accuracy on benchmark datasets (100 labeled data)

Dataset	base(NB)	selftrain	cotrain	SVM	TSVM
g241c	72.64	85.04 v	80.46 v	76.89	81.54 v
g241d	69.46	50.01	50.20	75.36	77.58 v
Digit1	93.80	93.40	93.64	94.47	93.85
USPS	83.24	70.65	74.49	90.25	90.23
COIL	53.20	34.30	34.93	77.07	74.20
BCI	52.17	50.06	50.56	65.69	66.75 v

about 50% of the datasets, which is similar to its performance on the 26 UCI datasets.

Hence, to summarize, these semi-supervised learning methods (except TSVM) generally do not work well on the 26 UCI datasets and the six benchmark datasets, although the classifiers used in these frameworks have good performance in supervised learning. And TSVM improves the performance with around 50:50 chance. Some analysis will be conducted in Section 3.4.

3.3 Performance Comparison between Two Selection Methods

Little work has ever been done to examine the effect of randomly selecting unlabeled instances. In this section, performance of “random selection” against “confident selection” in self-training and co-training are discussed. Table 3.8 shows the average results of using “random selection” in self-training and co-training against “base” on 26 UCI datasets when lp is 10%. Figures of average accuracies are not listed due to space limitation. The

“w/t/l” counts of t-test results for “random selection” against “confident selection” are summarized in Table 3.9. For self-training, it is observed that “random selection” even outperforms “confident selection” on three or four UCI datasets when using NB, TAN, HGC and HNB. For co-training, “random selection” outperforms “confident selection” when using NB, while the two methods get similar performance when using the other five classifiers. The observations demonstrate that, on these datasets, particularly selecting the most confident unlabeled data in self-training and co-training is not necessarily superior to randomly selecting unlabeled instances.

Table 3.8: The w/t/l counts of t-test results for using “random selection” against baseline when $lp = 10\%$

(a)

	NB		NBTree		TAN	
	selftrain	cotrain	selftrain	cotrain	selftrain	cotrain
w/t/l	0/18/8	0/15/11	0/25/1	0/17/9	0/26/0	0/19/7

(b)

	HGC		DNB		HNB	
	selftrain	cotrain	selftrain	cotrain	selftrain	cotrain
w/t/l	1/24/1	1/17/8	1/23/2	0/16/10	0/24/2	0/15/11

Table 3.9: The w/t/l counts of t-test results for “random selection” against “confident selection” when $lp = 10\%$

underlying classifier	selftrain	cotrain
NB	4/21/1	3/23/0
NBTree	1/24/1	0/26/0
TAN	4/22/0	0/26/0
HGC	3/22/1	0/26/0
DNB	1/24/1	1/24/1
HNB	4/21/1	2/20/4

3.4 Performance Analysis on NB in Standard Self-Training

In this section, we analyze the performance of NB in standard self-training from different aspects. Learning curves of classification accuracy on the testing set and on the remaining unlabeled data at each iteration are drawn to help analyze the performance. The correctness rate of labeling the selected unlabeled instances at each iteration is also shown on scatter plots for clear comparison.

In the following subsections, the learning curves in a random single run of four-fold stratified cross-validation on four datasets are shown in Figure 3.1 to 3.5, respectively. The parameter settings are the same as in Section 3.2, except I is set to 500 to provide more details. Each sub-graph can be considered as a random running with the 25% : 7.5% : 67.5% splitting ratio when generating testing set, L , and U . Each curve “*testing data*” represents the learning curve of accuracy of the learned classifier on the testing set at each iteration, and each curve “*U’ data*” represents the accuracy of the learned classifier on the remaining unlabeled data. The first point in each graph actually shows the baseline accuracy (accuracy of the classifier built without using any unlabeled data). Therefore, if a curve goes down, it indicates that the added unlabeled data do not help improve the performance. The dots on the scatter plot (drawn on the same graph), denoted as “*labeling*”, represent the correctness rate of labeling the newly selected unlabeled instances

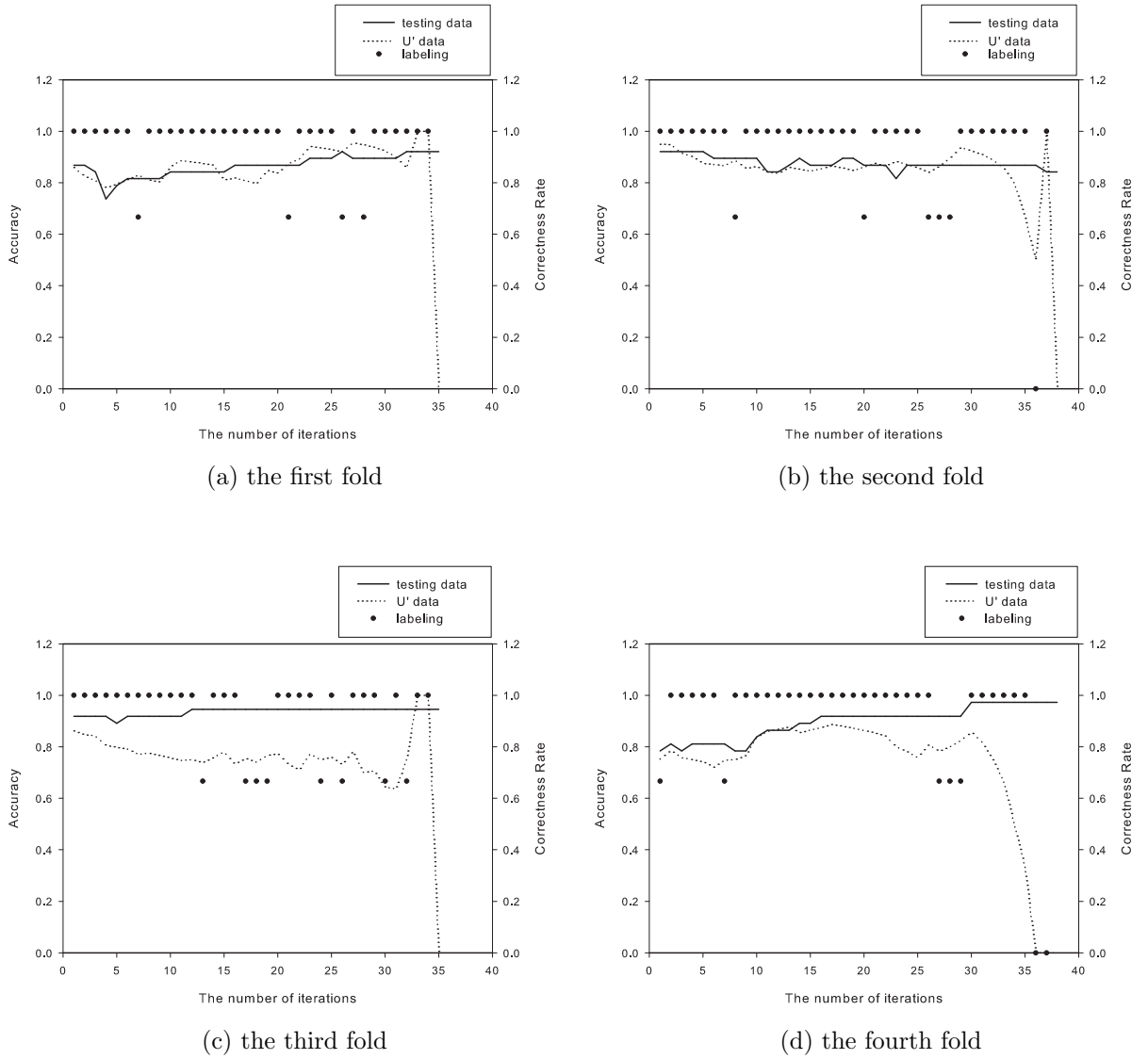


Figure 3.1: Learning curves on the *Iris* dataset when $lp = 10\%$ and $I = 500$

at each iteration. For example, at current iteration, if 10 unlabeled instances are selected and assigned labels, but only eight of the labels are correct (we

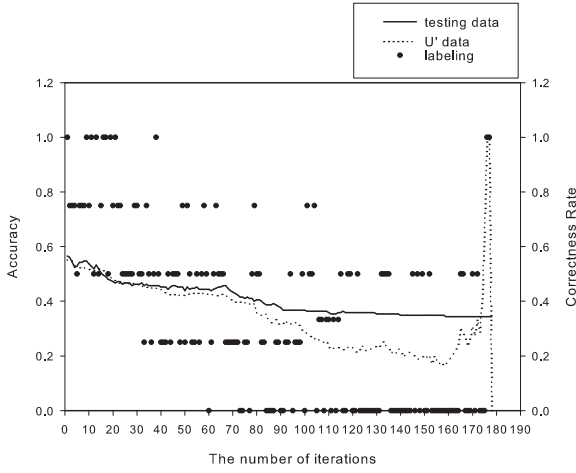
track it by checking the true labels in the original dataset used in our experiments), then the correctness rate is 0.8. Scatter plot “*labeling*” is added to the graphs to help us understand the performance of NB in self-training. Analysis for co-training is not conducted here because it is difficult to draw the learning curves under its two sub-views framework. The work remains for further study.

3.4.1 Does the Correctness Rate of Labeling Newly Added Instances Affect the Performance of Semi-Supervised Learning?

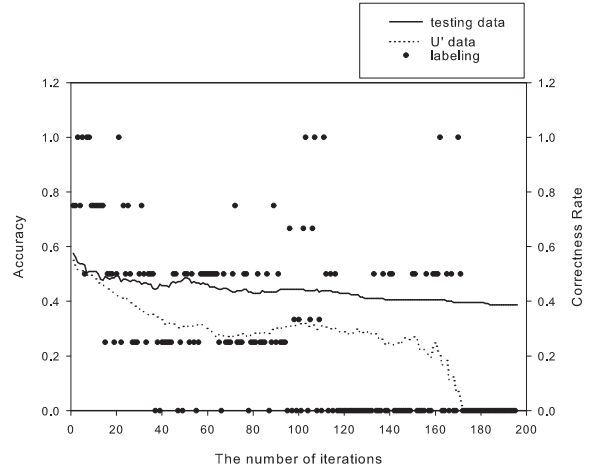
Here we utilize learning curves to discover the details of performance during each iteration. Due to space limitation, only a few learning curves are shown here.

In Figure 3.1, data points in “*labeling*” show that some selected unlabeled instances are wrongly labeled at some iterations. However, the final accuracy on testing data nevertheless reaches around 0.9. The performance on testing data fluctuates a bit but it is not ultimately affected in a significant way. A similar situation occurs on the *colic* dataset as displayed in Figure 3.3. Therefore, when there are wrongly labeled instances in only a few iterations, the correctness rate of labeling may not affect the final performance of the classifier significantly.

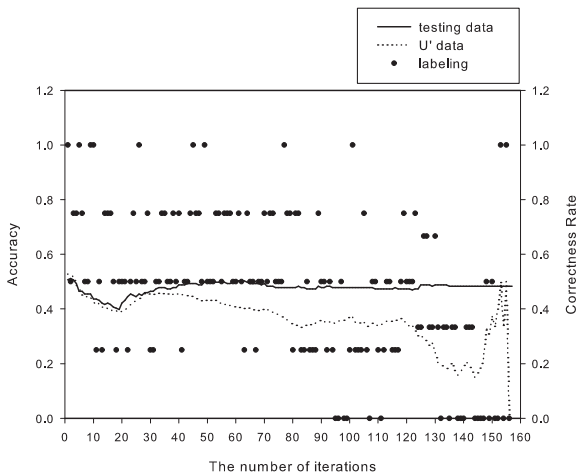
In Figure 3.2, the correctness rate of labeling falls below 0.8 in most of



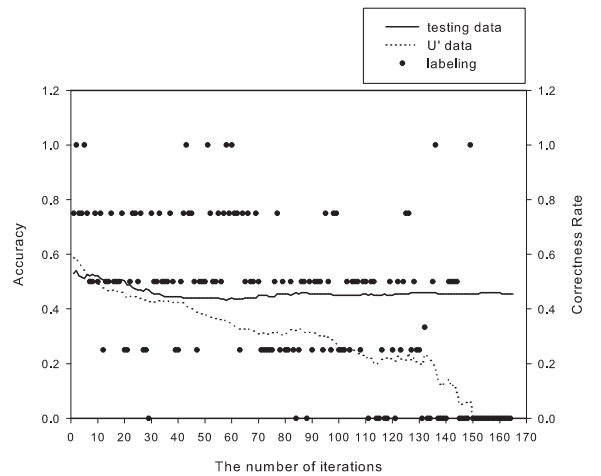
(a) the first fold



(b) the second fold



(c) the third fold



(d) the fourth fold

Figure 3.2: Learning curves on the *vehicle* dataset when $lp = 10\%$ and $I = 500$

the iterations, and the final accuracy on the testing data is generally worse than the accuracy in earlier iterations. In other words, when most of the

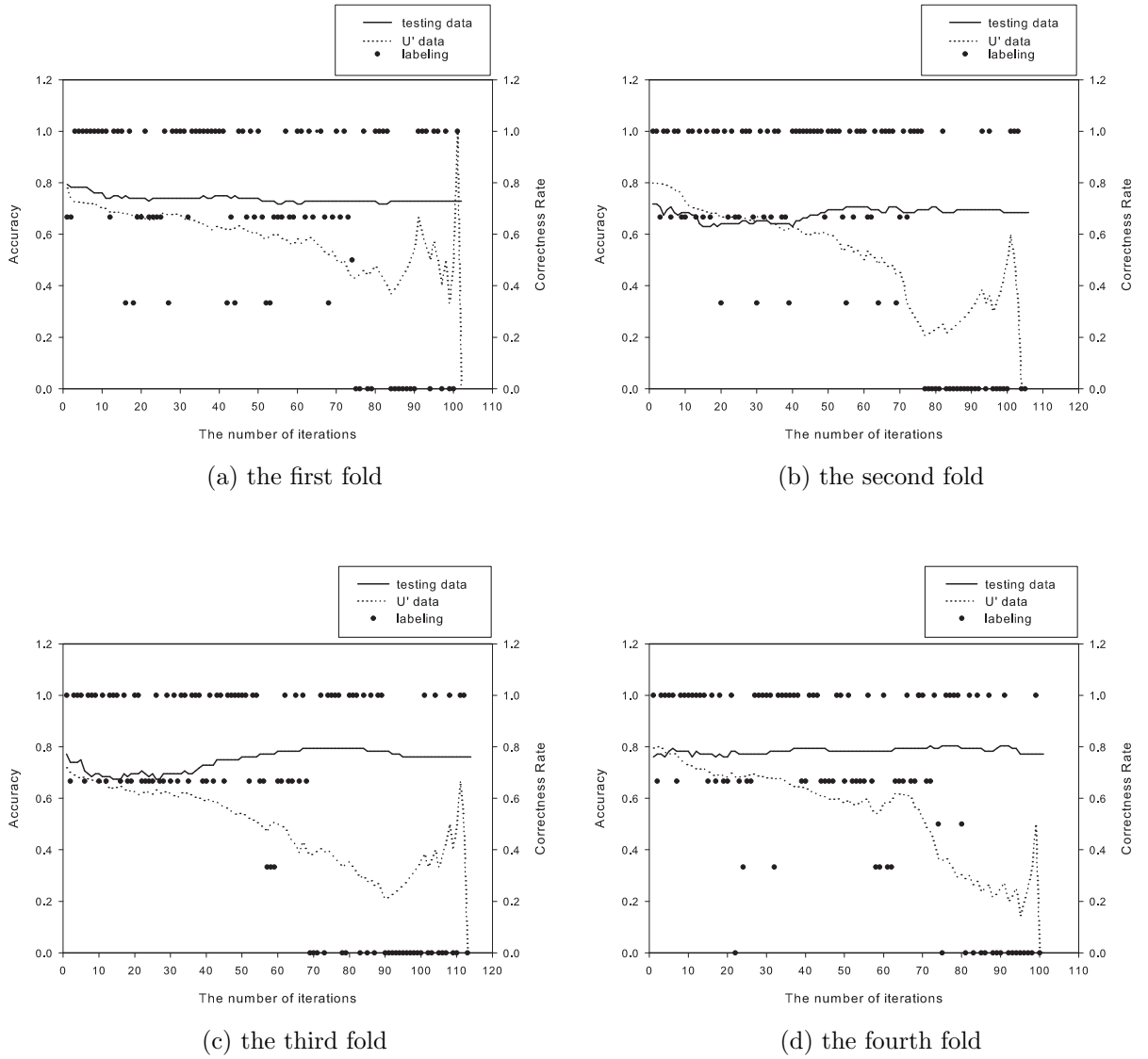


Figure 3.3: Learning curves on the *colic* dataset when $lp = 10\%$ and $I = 500$

selected unlabeled instances are wrongly labeled in most of the iterations, the accumulation of error causes bad performance of the final classifier.

Generally, the standard self-training and co-training should improve the performance if the selected unlabeled instances are labeled correctly (which is justified in Section 3.4.2 later). However, this was not the case in our experiments on the *mushroom* dataset.

A random running on the *mushroom* dataset in Figure 3.4 shows an interesting observation: the accuracy on the testing data and the accuracy on the remaining unlabeled data decrease although all the selected unlabeled instances were correctly labeled. To eliminate any possible influence caused by semi-supervised learning, we performed some experiments on the whole *mushroom* dataset: two small sets of instances were manually kept from the whole dataset as the initial training data and testing data; then a few instances were manually picked from the remaining data and added into the training data. The accuracies on the testing data were compared with each other when more instances were added into the training data. The results show that, on the *mushroom* dataset, when given more training data, accuracy is sometimes reduced. This is interesting because, intuitively, adding more instances should help improve the performance or should at least retain a similar performance - but the opposite situation occurred on the *mushroom* dataset here.

The curves of a random running on *mushroom*, when I is set to 3500, are shown in Figure 3.5 to examine the performance when more unlabeled instances are used. More numbers of incorrect labeling occur when more unlabeled data are used. It therefore seems that the initial labeled data may not

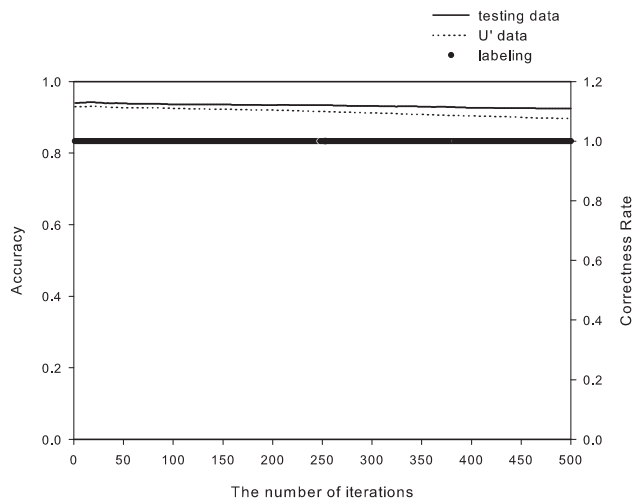


Figure 3.4: Learning curves on the *mushroom* dataset when $lp = 10\%$ and $I = 500$

be a good representative of the whole training data, and the new classifier tends to select more and more similar instances around the labeled data.

3.4.2 Does the Amount of Labeled Data Matter for Underlying Classifiers?

Table 3.10 shows the results of the two-tailed t -test when lp changes from 5% to 100%. Each entry $w/t/l$ shows that NB learned from the labeled data in the corresponding column wins in w datasets, ties in t datasets, and loses in l datasets, compared to NB learned from the labeled data in the corresponding row. The figures display that the performance of underlying classifier improves when lp increases, which means that there is lots of room to im-

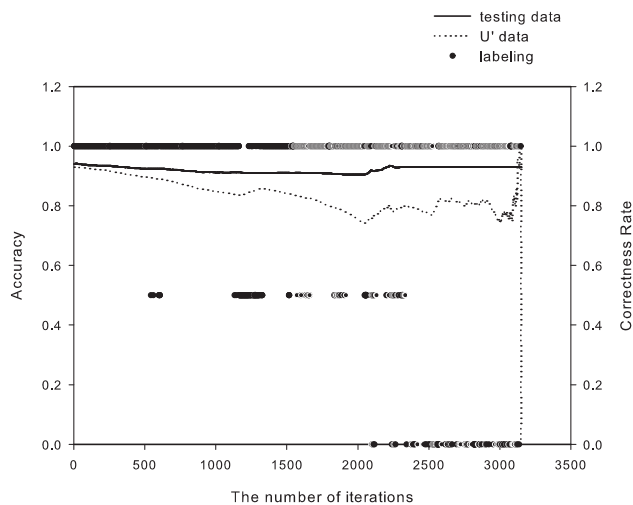


Figure 3.5: Learning curves on the *mushroom* dataset when $lp = 10\%$ and $I = 3500$

prove the performance by increasing instances to the labeled part, especially when lp is smaller than 25%.

3.4.3 Does the Amount of Labeled Data Affect the Performance of Semi-Supervised Learning?

Table 3.11 shows the results of the two-tailed t -test when lp changes from 5% to 100%. Each entry $w/t/l$ shows how self-training with lp in the corresponding column wins in w datasets, ties in t datasets, and loses in l datasets, compared to self-training with lp in the corresponding row. It is evident that, the overall performance of self-training on the 26 UCI datasets generally increases when the number of instances in original L increases.

Table 3.10: Summary of T-Test results: accuracy of NB built on the labeled data only when lp changes

lp	10%	15%	20%	25%	30%	35%	40%	100%
5%	5/21/0	10/16/0	12/14/0	13/13/0	15/11/0	15/11/0	15/11/0	20/6/0
10%		4/22/0	5/21/0	7/19/0	9/17/0	9/17/0	10/16/0	15/11/0
15%			1/25/0	4/22/0	4/22/0	5/21/0	6/20/0	10/16/0
20%				0/26/0	2/24/0	2/24/0	3/23/0	6/20/0
25%					0/26/0	0/26/0	1/25/0	5/21/0
30%						0/26/0	0/26/0	5/21/0
35%							0/26/0	5/21/0
40%								4/22/0

However, with a given lp , self-training still cannot win the corresponding “base” on most of the 26 datasets. Moreover, by examining the results on each dataset when lp changes from 5% to 40% (which are omitted here due to space limitation), the experimental results show that the standard self-training almost always get worse performance than the base classifiers (built on the labeled data only) on 5 multi-class datasets (*letter*, *segment*, *vehicle* and *vowel*) and 4 binary-class datasets (*colic.ORIG*, *credit-g*, *kr-vs-kp* and *mushroom*). These observations prompt further study in the following subsection.

Table 3.11: Summary of T-Test results: accuracy of NB in self-training when lp changes

lp	10%	15%	20%	25%	30%	35%	40%	100%
5%	4/22/0	6/20/0	6/20/0	8/18/0	8/18/0	11/15/0	12/14/0	15/11/0
10%		2/24/0	4/22/0	6/20/0	7/19/0	8/18/0	9/17/0	15/11/0
15%			2/24/0	4/22/0	5/21/0	5/21/0	6/20/0	16/10/0
20%				2/24/0	3/23/0	5/21/0	7/19/0	15/11/0
25%					2/24/0	3/23/0	5/21/0	14/12/0
30%						2/24/0	3/23/0	14/12/0
35%							2/24/0	14/12/0
40%								10/16/0

3.4.4 Does the Initial Performance of the Classifier Affect the Final Performance of Semi-Supervised Learning?

The initial classifier that is built on the original labeled data may affect the performance of semi-supervised learning, which is demonstrated in Figure 3.2. On *vehicle* dataset, the initial accuracies on testing are around 0.5 and decrease quickly with more selected unlabeled data. Scatter plot “labeling” shows that most of the selected unlabeled instances are incorrectly labeled. Our experimental results on underlying classifier (built on the labeled data only) also shows that NB can only achieve accuracy below 60% on testing sets of *vehicle* no matter what value is set to lp . If the initial classifier has a bad performance on the training set and the testing set, it is not accurate to predict labels for unlabeled instances. Thus the selected unlabeled data may hurt the performance.

3.4.5 Does the Amount of Unlabeled Data Affect the Performance?

There is no consistent conclusion that can be drawn from Figure 3.1 and Figure 3.3 about whether the amount of unlabeled data always affects the performance in one direction. It is observed in these two figures that, adding more unlabeled data may help increase the performance even if some newly selected instances are not correctly labeled, for example, the first and the

fourth sub-graph in Figure 3.1. There are also some situations that using more unlabeled data might degrade the performance such as the graph shown in Figure 3.4.

3.5 Summary

This chapter details the experiments conducted on 26 UCI datasets and 6 benchmark datasets using the standard self-training, co-training and co-EM with six different Bayesian classifiers, TSVM, and LLGC. Two kinds of selection criteria were also compared in the experiments. The results show that, some of these semi-supervised learning methods generally do not outperform the baseline performance of classifiers learned on the labeled data only; and particularly selecting unlabeled instances with some efforts during the learning process in self-training and co-training is not necessarily superior to randomly selecting unlabeled instances. To further explore these results, learning curves were drawn on several datasets, as well as for the correctness rate of labeling selected unlabeled instances at each iteration. The results illustrate a diversity of reasons that the standard self-training does not perform as well as may be ideally expected. The performance may be affected by the classifier used, or the dataset itself. An interesting observation on the “mushroom” dataset is that, the accuracy on the testing set can still decrease even when all the selected unlabeled instances at each iteration are correctly labeled. A method is presented in Chapter 4 to improve the

performance of Naive Bayes in the standard self-training and co-training.

Chapter 4

Instance Selection in Self-Training and Co-Training

As introduced in previous chapters, “confidence selection” is commonly used in self-training and co-training methods to select unlabeled instances to expand the labeled training set. Ideally, the selected unlabeled instances (together with the predicted labels) can finally help to learn a better classifier. However, in Chapter 3, results of the empirical study on 26 UCI datasets show that, the performance of using “confidence selection” is not necessarily superior to that of randomly selecting unlabeled instances. Using unlabeled data may degrade classification performance. In this chapter, we present an effective instance selection method based on the original labeled data (IS-BOLD) to improve the performance of self-training and co-training when using Naive Bayes as the underlying classifier.

4.1 Algorithm Description

The main idea of ISBOLD is to use the accuracy on the original labeled data only to prevent adding unlabeled instances that will possibly degrade the performance. ISBOLD considers both the prediction confidence of the current classifier on the self-labeled data and the accuracy on the original labeled data only. In each iteration, after the selection of the most confident unlabeled instances, the accuracy of the current classifier on the original labeled data is computed and then used to decide whether to add the selected instances to the training set in the next iteration. How to use ISBOLD in self-training and co-training scenarios is described in following two subsections, respectively.

4.1.1 ISBOLD for Self-Training

In order to describe our method, some notations are used here. In iteration t , we use L_t to denote the new labeled training set in iteration t , C_t to represent the classifier built on L_t , and Acc_t as the accuracy of C_t on the original labeled data L_0 . The detailed algorithm is shown in Figure 4.1.

The difference between ISBOLD and the common confidence selection method in self-training is displayed in steps 4(e) and 4(f). In iteration $t + 1$, after selecting the most confident unlabeled instances and assigning labels to them (for simplicity, the set of those selected instances is denoted as L_{t+1}^s), the training set $L_{t+1} = L_t \cup L_{t+1}^s$. Now we build a classifier C_{t+1} on L_{t+1} and

-
1. Set t , the iteration counter, to 0.
 2. Build a classifier C_t on the original labeled data L_0 .
 3. Compute Acc_t , which is the accuracy of C_t on L_0 .
 4. While the stopping criteria are not satisfied,
 - (a) Use C_t to predict a label for each instance in U .
 - (b) Generate L_{t+1}^s : select m unlabeled instances that C_t has high classification confidence, and assign a predicted label to each selected instance.
Delete the selected instances from U .
 - (c) $L_{t+1} = L_t \cup L_{t+1}^s$.
 - (d) Build a classifier C_{t+1} on L_{t+1} .
 - (e) Compute Acc_{t+1} , which is the accuracy of C_{t+1} on L_0 .
 - (f) If $Acc_{t+1} < Acc_t$, then $L_{t+1} = L_t$, and rebuild C_{t+1} on L_{t+1} .
 - (g) Increase t by 1.
 5. Return the final classifier.
-

Figure 4.1: Algorithm of ISBOLD for self-training

compute Acc_{t+1} . If $Acc_{t+1} < Acc_t$, L_{t+1} is reset to be equal to L_t , and C_{t+1} is updated on L_{t+1} accordingly. The whole process iterates until there is no unlabeled instance left or the maximum number of iterations is reached.

The reason that we remove L_{t+1}^s from L_{t+1} once the accuracy on L_0 decreases is that, if adding L_{t+1}^s to the training set degrades the classifier’s performance on L_0 , it is very possible that the performance of the current classifier on the test set degrades as well. Hence, we use this method to roughly prevent possible performance degradation. Furthermore, notice that in step 4(b), all the selected instances are removed from U , which means that each selected instance is either added to the labeled data or removed from U .

4.1.2 ISBOLD for Co-Training

A similar selection method is used in co-training. We denote the classifiers on the two sub-views in iteration t as C_t^a and C_t^b . The algorithm is shown in Figure 4.2.

The difference between ISBOLD and the common confidence selection method in co-training is displayed in steps 8(f), 8(g), 8(i) and 8(j). In iteration $t + 1$, on sub-view a , after selecting a certain number of unlabeled instances that C_t^b has high classification confidence, a label is assigned to each selected instance (for simplicity, the set of those selected instances is denoted as $L_{t+1}^{a^s}$). Then $L_{t+1}^a = L_t^a \cup L_{t+1}^{a^s}$ and C_{t+1}^a is built on L_{t+1}^a . Now we compute Acc_{t+1}^a that represents the accuracy of C_{t+1}^a on L_0^a . If $Acc_{t+1}^a < Acc_t^a$, $L_{t+1}^a = L_t^a$ and C_{t+1}^a is updated accordingly. The same steps are repeated on sub-view b to

-
1. Set t , the iteration counter, to 0.
 2. Randomly partition the attribute set Att into two separate sets Att_a and Att_b .
Generate L_0^a and L_0^b from L . Generate U_a and U_b from U .
 3. Generate data pool U'_a and U'_b by randomly choosing u instances from U_a and U_b , respectively.
 4. Use L_0^a to train a classifier C_t^a .
 5. Use L_0^b to train a classifier C_t^b .
 6. Compute Acc_t^a , which is the accuracy of C_t^a on L_0^a .
 7. Compute Acc_t^b , which is the accuracy of C_t^b on L_0^b .
 8. While the stopping criteria are not satisfied,
 - (a) Use C_t^a to predict a label for each instance in U'_a . Use C_t^b to predict a label for each instance in U'_b .
 - (b) Generate $L_{t+1}^{a^s}$: select m unlabeled instances that C_t^b has high classification confidence, together with predicted labels. Delete the selected instances from U'_b .
 - (c) Generate $L_{t+1}^{b^s}$: select m unlabeled instances that C_t^a has high classification confidence, together with predicted labels. Delete the selected instances from U'_a .
 - (d) $L_{t+1}^a = L_t^a \cup L_{t+1}^{a^s}$. $L_{t+1}^b = L_t^b \cup L_{t+1}^{b^s}$.
 - (e) Use L_{t+1}^a to train a classifier C_{t+1}^a .
 - (f) Compute Acc_{t+1}^a , which is the accuracy of C_{t+1}^a on L_0^a .
 - (g) If $Acc_{t+1}^a < Acc_t^a$, then $L_{t+1}^a = L_t^a$, and rebuild C_{t+1}^a on L_{t+1}^a .
 - (h) Use L_{t+1}^b to train a classifier C_{t+1}^b .
 - (i) Compute Acc_{t+1}^b , which is the accuracy of C_{t+1}^b on L_0^b .
 - (j) If $Acc_{t+1}^b < Acc_t^b$, then $L_{t+1}^b = L_t^b$, and rebuild C_{t+1}^b on L_{t+1}^b .
 - (k) Randomly move m instances from U_a to replenish U'_a .
Randomly move m instances from U_b to replenish U'_b .
 - (l) Increase t by 1.

Figure 4.2: Algorithm of ISBOLD for co-training

generate L_{t+1}^b and C_{t+1}^b . New unlabeled instances are replenished from the remaining unlabeled data part to the data pool of each sub-view. The whole process iterates until there is no unlabeled instance left or the maximum number of iterations is reached.

4.2 Experimental Results and Analysis

4.2.1 Experimental Settings

We ran experiments on the same 26 datasets used in Chapter 3. In our experiments, ten runs of four-fold stratified cross-validation are conducted, and lp is set to be 5%. Naive Bayes is used in self-training and co-training. The maximum number of iterations in both is set to 80. For co-training, the attributes are randomly split into two subsets, and the size of data pool is set to be 50% of the size of U . Accuracy and AUC are used as performance measurements.

4.2.2 Results Analysis

Performance comparison results of using ISBOLD and using the common “confidence selection” method in self-training and co-training are shown in Table 4.1 and Table 4.2. For simplicity, the methods are denoted as **ISBOLD** and **CF** in the tables. In each table, figures on each row are the average accuracy or AUC over ten-runs of four-fold cross-validation on the

Table 4.1: Accuracy of **CF** vs **ISBOLD** in self-training and co-training

(a) self-training			(b) co-training		
Dataset	CF	ISBOLD	Dataset	CF	ISBOLD
balance-scale	59.52	66.21	balance-scale	59.10	67.17
breast-cancer	65.09	65.61	breast-cancer	70.41	71.00
breast-w	96.67	96.34	breast-w	96.85	96.47
colic	74.54	75.38	colic	76.60	75.76
colic.ORIG	55.05	60.57	colic.ORIG	55.19	62.04
credit-a	80.68	80.78	credit-a	81.36	79.67
credit-g	60.62	66.03 v	credit-g	63.04	67.72 v
diabetes	70.55	70.53	diabetes	67.51	69.58
heart-c	81.55	81.15	heart-c	82.77	80.13
heart-h	83.06	82.41	heart-h	81.46	78.60
heart-statlog	81.37	80.74	heart-statlog	82.03	80.30
hepatitis	79.70	78.34	hepatitis	81.04	80.21
ionosphere	80.97	79.86	ionosphere	81.50	83.08
iris	90.31	90.05	iris	80.79	78.98
kr-vs-kp	67.26	80.07 v	kr-vs-kp	59.22	77.36 v
labor	88.26	87.92	labor	77.21	78.43
letter	40.38	57.39 v	letter	36.67	56.05 v
mushroom	91.90	92.57 v	mushroom	91.74	92.38 v
segment	63.49	72.88 v	segment	61.49	71.64 v
sick	91.54	94.15	sick	93.40	93.56
sonar	55.72	57.93	sonar	55.43	58.08
splice	82.05	85.48 v	splice	73.91	82.63 v
vehicle	41.79	48.35	vehicle	41.57	47.86
vote	87.89	88.53	vote	88.21	88.60
vowel	18.75	21.78	vowel	18.83	23.36
waveform-5000	77.98	78.87	waveform-5000	71.61	75.91 v
mean	71.80	74.61	mean	70.34	73.71
vs CF: w/t/l	-	6/20/0	vs CF: w/t/l	-	7/19/0

corresponding dataset. Row “w/t/l” represents that using ISBOLD in the corresponding column wins on w datasets (marked by ‘v’), ties on t datasets, and loses on l datasets (marked by ‘*’) against using “confidence selection” in self-training or co-training, under a two-tailed pair-wise t-test with the significant level of 95%. Values in row “mean” are the average accuracy or AUC over the 26 datasets.

Table 4.1(a) shows the average accuracy of using **ISBOLD** and **CF** in self-training. The “w/t/l” t-test results show that, ISBOLD significantly im-

Table 4.2: AUC of **CF** vs **ISBOLD** in self-training and co-training

(a) self-training			(b) co-training		
Dataset	CF	ISBOLD	Dataset	CF	ISBOLD
balance-scale	61.37	66.68	balance-scale	60.44	65.34
breast-cancer	63.98	63.48	breast-cancer	63.51	64.37
breast-w	99.07	99.08	breast-w	99.22	99.19
colic	79.24	78.43	colic	78.99	79.08
colic.ORIG	51.62	58.49	colic.ORIG	49.62	55.82
credit-a	86.81	86.79	credit-a	88.05	86.35
credit-g	56.56	65.24 v	credit-g	55.33	61.62
diabetes	78.03	76.36	diabetes	72.61	74.95
heart-c	83.97	83.92	heart-c	84.02	83.80
heart-h	83.74	83.74	heart-h	83.77	83.50
heart-statlog	88.93	88.64	heart-statlog	90.03	88.03
hepatitis	83.02	80.99	hepatitis	78.38	73.19
ionosphere	86.86	86.68	ionosphere	87.89	88.92
iris	98.33	98.29	iris	93.21	92.27
kr-vs-kp	74.65	89.03 v	kr-vs-kp	66.86	86.39 v
labor	96.59	96.72	labor	87.76	85.18
letter	86.09	93.08 v	letter	82.98	92.57 v
mushroom	98.04	98.81 v	mushroom	97.89	98.75 v
segment	90.86	95.24 v	segment	87.93	94.82 v
sick	91.51	93.96	sick	87.74	93.83
sonar	58.64	62.21	sonar	59.59	62.93
splice	94.40	96.23 v	splice	88.65	94.87 v
vehicle	59.63	66.95 v	vehicle	59.56	67.09 v
vote	96.31	96.52	vote	96.31	96.46
vowel	57.65	64.49 v	vowel	57.97	66.44 v
waveform-5000	88.85	90.96 v	waveform-5000	84.22	89.54 v
mean	80.57	83.12	mean	78.56	81.74
vs CF: w/t/l	-	9/17/0	vs CF: w/t/l	-	8/18/0

proves classification accuracy on six datasets. Values in row “mean” also demonstrate that ISBOLD improves the average performance. Table 4.1(b) shows the average accuracies in co-training. The “w/t/l” t-test results tell that ISBOLD significantly improves the performance of co-training on seven datasets. And the mean value increases from 70.34 to 73.71.

Comparison results on AUC in self-training and co-training are displayed in Table 4.2. It can be observed that, using ISBOLD, the AUC of self-training is significantly improved on nine datasets. And the mean value increases

from 80.57 to 83.12. Similarly, the AUC of co-training is sharply improved on eight datasets, and the mean value is improved from 78.56 to 81.74.

4.2.3 Learning Curves Analysis

Based on our previous work [23], we guess that, the classifier should have a good prediction performance on the testing set if the accuracy on the original labeled data does not degrade. To verify our conjecture and to further examine the performance of ISBOLD during each iteration, learning curves of a random running of two self-training methods on datasets *vehicle* and *kr-vs-kp* are displayed in Figure 4.3 and Figure 4.4, respectively. The data splitting setting is the same as that in Section 4.2.1. Curves in co-training are omitted here due to space limitation.

On each graph, at each iteration t , the accuracy values of classifier C_t on the original labeled data L_0 and the testing set for using **ISBOLD** or **CF** in self-training are displayed, respectively. Curves “ISBOLD- L_0 ” and “ISBOLD-test” show accuracy values on the original labeled data L_0 and on the testing set, respectively, when using ISBOLD in self-training on the dataset. Curves “CF- L_0 ” and “CF-test” display accuracy values on L_0 and on the testing set, respectively, when using “confidence selection” in self-training on the dataset.

According to our conjecture, when the accuracy on the original labeled data L_0 decreases, the accuracy on the corresponding testing set generally decreases as well. This is actually observed on the trends of curve “CF- L_0 ”

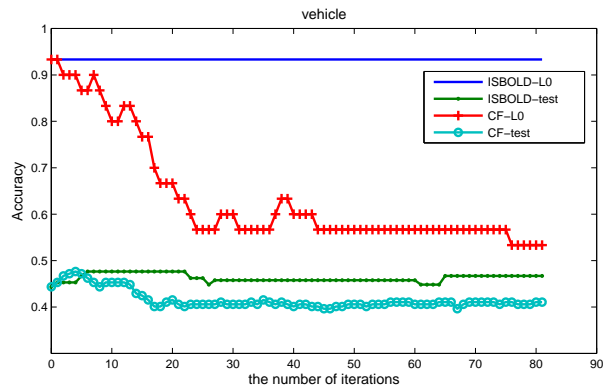


Figure 4.3: Learning curves on the *vehicle* dataset

and curve “CF-test” in Figure 4.3 and Figure 4.4. Curve “CF-test” generally goes down when curve “CF-L0” goes down.

ISBOLD is presented based on our conjecture that the classifier will have good prediction performance on the testing set if its accuracy on the original labeled data does not degrade during each iteration. As shown in Figure 4.3 and Figure 4.4, comparing curves on “confidence selection” method to curves on ISBOLD method, ISBOLD can sharply improve the accuracy on the testing set while improving it on L_0 . When the accuracy on L_0 does not degrade, the final accuracy on the testing set does not significantly decrease. These observations confirm that, it is an effective way to improve the classification accuracy of semi-supervised learning by using the accuracy on the original labeled data to further decide whether to accept the selected unlabeled instances into the next iteration or not.

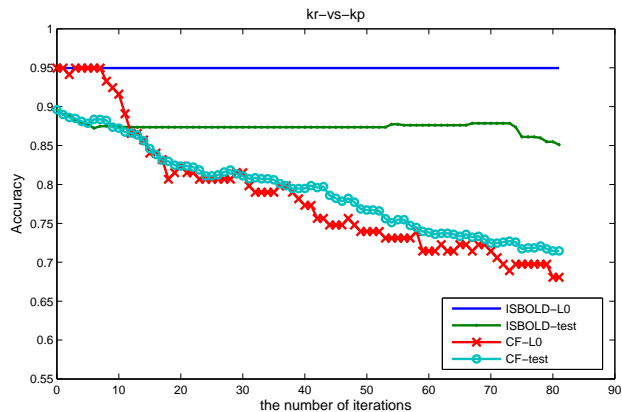


Figure 4.4: Learning curves on the $kr-vs-kp$ dataset

4.3 Discussion

In standard self-training and co-training, if the current classifier has poor performance and wrongly assigns labels to some self-labeled instances, the final performance will be jeopardized due to the accumulation of mislabeled data. It is a general problem for the methods based on the classifier performance on the expanded data, including the original labeled data and the self-labeled data. Since the originally labeled instances are generally more reliable than self-labeled instances, the performance on the former instances alone is more critical. ISBOLD considers both the prediction confidence of the current classifier on the self-labeled data and the accuracy on the original labeled data only. The effectiveness of ISBOLD algorithm confirmed our conjecture that, the classifier should have a good performance on the original labeled data if it wants to have good prediction performance on future data. More precisely, when the accuracy of the classifier evaluated on the

original labeled data decreases, the accuracy on the future testing set generally degrades as well. Hence, utilizing the accuracy on the original labeled data to select more reliable unlabeled instances seems crucial to the final performance of semi-supervised learning.

4.4 Summary

In this chapter, a new instance selection method, ISBOLD, is presented to improve the performance of self-training and co-training when using Naive Bayes as the underlying classifier. During each iteration, after selecting a number of unlabeled instances that the current classifier has high classification confidence, we use the change of accuracy of the current classifier on the original labeled data to decide whether to accept the selected unlabeled instances to expand the labeled training set in the next iteration. Experiments on 26 UCI datasets show that ISBOLD can significantly improve the performance of self-training and co-training on many datasets. The learning curve analysis gives a vivid demonstration and experimentally proves the feasibility of our method.

Chapter 5

Cost-Sensitive Semi-Supervised Bayesian Learning Methods

In this chapter, we deal with the application of Naive Bayes in the problem that the labeled data is scarce and different misclassification errors result in different costs. A special case that datasets have imbalanced data distribution is introduced at the beginning, which is natural to be handled by cost-sensitive learning methods if assigning different misclassification costs to the classes. Hence, three methods on cost-sensitive learning in semi-supervised learning scenario are presented in the chapter: a single-model cost-sensitive self-training method and two cost-sensitive self-training methods using multiple classifiers. We focus on binary datasets here, and the proposed methods might be extended on multi-class datasets in the future.

5.1 A Study of Semi-Supervised Learning on Class-Imbalanced Data

Section 2.2 introduces that learning classifiers on data with skewed class distributions may cause bad predictions on the minority class. Here, we conduct experiments on some special cases to confirm the phenomenon in self-training and co-training. Given the percentage of positive instances in the labeled data, we randomly sample a new dataset with the same size of the original data set with different class distributions. Then self-training and co-training are applied on the new dataset. By changing the percentage of positive instances in the labeled data from 0 to 1, we observe the classification accuracy on the test set that has the same class distribution as the labeled data.

Figure 5.1 shows the average results of true positive rate (TPR) and true negative rate (TNR) of 10 runs of 10-fold cross-validation on datasets sampled from the “credit-g” dataset when the ratio of labeled data to unlabeled data is fixed to 10% : 90% in self-training. Computation methods for TPR and TNR are described in Appendix B.

The curves show that, when there are fewer positive instances in the training labeled data, most test instances are predicted by the learn classifier to negative class because TPR is close to 0 and TNR is close to 1.0; when the percentage of positive instances is close to 1.0, most test instances are predicted as positive class. Hence, when the class distribution is not balanced,

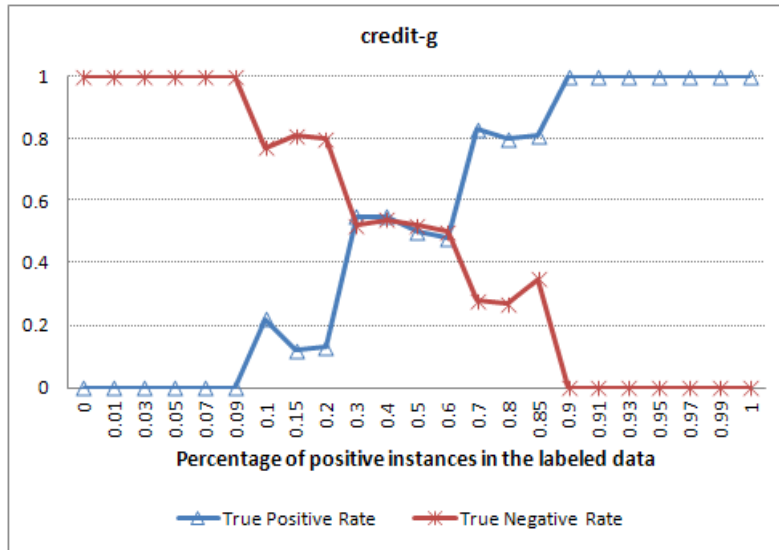


Figure 5.1: Results on class-imbalanced data

the learned classifier tends to classify more instances to the majority class which might ignore the minority class when the percentage of minority class in the training data is very small.

Cost-sensitive learning is a very natural method to apply on the class-imbalanced datasets if we consider that different classes have unequal misclassification costs. In the following sections, we will introduce three cost-sensitive semi-supervised learning methods.

5.2 Analysis of Cost-Sensitive Learning

Cost-sensitive learning deals with the problem that misclassifying different classes are associated with different costs. A cost matrix for binary-class

datasets is shown in Figure 5.2. Suppose the class with lower misclassification cost is represented as positive (P), and the class with higher misclassification cost as negative (N). “ cfp ” is the cost of wrongly classifying a negative instance to be positive. “ cfn ” is the cost of misclassifying a positive instance to be negative. “ ctp ” and “ ctn ” are the costs of correctly classifying a positive instance and a negative instance, respectively. Usually, $ctp=ctn=0$, and $cfp > cfn > 0$.

		Predicted class	
		P	N
Actual class	P	CTP	CFN
	N	CFP	CTN

Figure 5.2: Cost matrix for binary-class datasets

Here, we use the confusion matrix to analyze the cost-sensitive problem. Descriptions of the confusion matrix for binary-class datasets are given in Appendix B. Three confusion matrixes are shown in Figure 5.3. Sub-graph (c) represents the general case that there are some misclassified positive instances and negative instances when applying a classifier on a testing dataset. The numbers of true positive instances, false positive instances, true negative instances, and false negative instances are p_1 , n_1 , n_2 , and p_2 , respectively. Sub-graph (a) shows the confusion matrix that all the instances are predicted as “positive”. Sub-graph (b) displays the confusion matrix that all the instances are predicted as “negative”. By computing the total cost, we have $cost_a = (p_1 + p_2) * ctp + 0 * cfn + (n_1 + n_2) * cfp + 0 * ctn = (n_1 + n_2) * cfp$,

$cost_b = 0 * ctp + (p_1 + p_2) * cfn + 0 * cfp + (n_1 + n_2) * ctn = (p_1 + p_2) * cfn$,
 and $cost_c = p_1 * ctp + p_2 * cfn + n_1 * cfp + n_2 * ctn = p_2 * cfn + n_1 * cfp$. By
 comparing the above three equations, we obtain following results:

1. If $\frac{cfp}{cfn} < \frac{p_2}{n_2}$, then $cost_a < cost_c$ which means that, classifying all instances as “positive” can get smaller misclassification cost than using some specific classifier.
2. If $\frac{cfp}{cfn} > \frac{p_1}{n_1}$, then $cost_b < cost_c$ which means that, classifying all instances as “negative” can get smaller misclassification cost than using some specific classifier.
3. If $\frac{cfp}{cfn} < \frac{p_1+p_2}{n_1+n_2}$, then $cost_a < cost_b$ and classifying all instances as “positive” has smaller misclassification cost than classifying them as “negative”; otherwise, classifying all as “negative” has smaller cost than classifying as “positive”.

		Predicted class	
		P	N
Actual class	P	p_1+p_2	0
	N	n_1+n_2	0

(a)

		Predicted class	
		P	N
Actual class	P	0	p_1+p_2
	N	0	n_1+n_2

(b)

		Predicted class	
		P	N
Actual class	P	p_1	p_2
	N	n_1	n_2

(c)

Figure 5.3: Three confusion matrixes

The two cases in sub-graph (a) and sub-graph (b) are two extreme cases, which make sense in real-world applications when we want to correctly find instances of one particular class and ignore the other class. The ideal situation is that a good cost-sensitive classifier is able to correctly classify instances

for both classes, that is, $p_2 \rightarrow 0$ and $n_1 \rightarrow 0$. It is difficult to design a classifier to meet the ideal situation. An alternative way to get lower misclassification costs is to design classifiers to get high values of n_1 while sacrificing the whole accuracy to a small degree.

5.3 A Single-Model Cost-Sensitive Self-Training Method

In this section, a new cost-sensitive self-training method CS-ST is presented. The main idea of CS-ST is to consider the expected cost when selecting and labeling the unlabeled instances so as to adapt the self-training algorithm to the cost-sensitive learning problem. The degree of change of the average misclassification cost is used as a further selection criterion to decide whether to add the selected instances into the training data.

5.3.1 Algorithm Description

The cost matrix described in Section 5.2 is used in our algorithm. We set $cfn = 1$ and $cfp > 1$ because misclassifying a negative instance is associated with a larger cost. By fixing the value of cfn to 1, the value of cfp reflects the ratio of the misclassification costs of the two classes. The average misclassification

cost on a testing dataset with m instances is formulated as:

$$AC = \frac{\sum_{i=1}^m C(\text{predictedclass}_i, \text{actualclass}_i)}{m} \quad (5.1)$$

where predictedclass_i is the predicted class label of the i -th testing instance, actualclass_i is the actual class label of the i -th testing instance, and $C(\text{predictedclass}_i, \text{actualclass}_i)$ is the cost of misclassifying the instance of the actual class label to the predicted class label.

The CS-ST algorithm is described in Figure 5.4. Initially, a Naive Bayes classifier C_0 is learned from L_0 . In iteration t , L_t is updated and a new Naive Bayes classifier C_t is built from L_t . The unlabeled instances are selected and labeled according to the expected costs. The change of the average misclassification cost is used to further decide whether to use the selected instances to expand the training data. Since the real labels of the unlabeled data are unknown to the algorithm, it is not feasible to compute the actual misclassification cost of the each classifier in each iteration. Therefore, we estimate the average misclassification cost of C_t , denoted as \widetilde{AC}_t , on a small dataset with real labels (L_0). The stopping criterion is that the maximum number of iterations is reached or there is no unlabeled instances left in U . Compared to the standard self-training, the misclassification cost is considered in three places:

- Selection (step 4(a)): after current classifier C_t produces the prediction probability for each unlabeled instance, the expected cost is computed

-
1. Set the iteration counter t to 0.
 2. Build a Naive Bayes classifier C_0 on the labeled data L_0 only.
 3. Compute \widetilde{AC}_t .
 4. While the stopping criteria are not satisfied,
 - (a) Select m unlabeled instances from U_t that classifier C_t has the smallest expected cost.
 - (b) Assign each selected unlabeled instance an “optimum” label with the smallest expected cost computed by Equation 2.1.
 - (c) Form L_{t+1} as the union of L_t and the selected instances.
 - (d) Form U_{t+1} by deleting the selected instances from U_t .
 - (e) Build a Naive Bayes classifier C_{t+1} on L_{t+1} .
 - (f) Compute \widetilde{AC}_{t+1} .
 - (g) If $\widetilde{AC}_{t+1} > \widetilde{AC}_t$, $L_{t+1} = L_t$ and $C_{t+1} = C_t$.
 - (h) Increase t by 1.
 5. Return the final classifier.
-

Figure 5.4: The CS-ST algorithm

using Equation 2.1. The m unlabeled instances with the smallest expected cost will be selected.

- Labeling (step 4(b)): for each of the m selected instance, assign it the “optimum” class that has the smallest expected cost. This step is similar to the labeling step used in the CS-EM algorithm [55]. The difference is that, in CS-EM, it computes a value (between 0 and 1) as the weight of the instance after assigning the “optimum” class; while all unlabeled instances have the same weight 1.0 in CS-ST.
- Whether to accept the m instances to expand the labeled data in the next iteration (steps 4(f-g)): if $\widetilde{AC}_{t+1} > \widetilde{AC}_t$, discard the m selected instances; otherwise, use them in the next iteration.

5.3.2 Experiments and Results

In this section, CS-ST is compared with following algorithms:

- (1) **SelfTrain**: it is the standard self-training method using Naive Bayes as the underlying classifier.
- (2) **SL**: it is a Naive Bayes classifier trained on the original labeled data only.

For each method, after the classifier is built, testing instances are assigned labels according to the predicted probabilities of the classifier, and the av-

erage misclassification cost is computed thereafter based on the number of misclassified instances and the corresponding costs.

5.3.2.1 Datasets

Two kinds of datasets are used to compare the performance of the methods. The first set is 13 datasets that appear in several papers of cost-sensitive learning [14] [57] [15] [55] [63]. They can be downloaded from UCI repository [1]. In our experiments, the datasets are pre-processed in Weka [67]. Missing values are replaced by the “*ReplaceMissing*” filter. Numeric values are discretized by the ten-bin discretization filter. The dataset “hypothyroid” is changed to a binary class dataset by selecting the two most frequent class values. The second set is three text datasets “oh0”, “oh5”, and “oh10” that are used in Qin [55].

Table 5.1: Experimental Datasets

Dataset	Size	#Attr	#Pos	#Neg	%Neg	#Pos/#Neg
kr-vs-kp	3196	37	1669	1527	47.78%	1.1
clean1	476	167	269	207	43.49%	1.3
bupa	345	7	200	145	42.03%	1.4
spambase	4601	58	2788	1813	39.40%	1.5
vote	435	17	267	168	38.62%	1.6
wdbc	569	31	357	212	37.26%	1.7
pima-indians	768	9	500	268	34.90%	1.9
tic-tac-toe	958	10	626	332	34.66%	1.9
breast-w	699	10	458	241	34.48%	1.9
credit-g	1000	21	700	300	30.00%	2.3
breast-cancer	286	10	201	85	29.72%	2.4
sick	3772	30	3541	231	6.12%	15.3
hypothyroid	3675	30	3481	194	5.28%	17.9
oh0	1003	3183	809	194	19.34%	4.2
oh5	918	3013	769	149	16.23%	5.2
oh10	1050	3239	885	165	15.71%	5.4

To be consistent, the order of class values in some datasets are changed so

that the majority (positive) class is the first class value while the minority (negative) class is the second class value. The modified datasets include “tic-tac-toe”, “bupa”, “breast-cancer”, “breast-w”, and “vote”.

The details of the data sets are displayed in Table 5.1. “#Attr” is the number of attributes in each dataset. Columns “#Pos” and “#Neg” show the number of instances belong to positive class and negative class, respectively, in each dataset. Column “%Neg” depicts the percentage of negative instances in each dataset. And column “#Pos/#Neg” is the ratio of the number of positive instances to the number of negative instances in each dataset.

5.3.2.2 Experimental Settings

On each dataset, ten runs of five-fold cross-validation are conducted and the average results are reported. The labeled percentage lp (following the notation in previous chapters) is set to be 1%. Hence, in each fold, 20% data is kept as the testing set, and the other 80% data is then randomly split into labeled data (1% of the 80% data) and unlabeled data (99% of the 80% data). The class distribution in the labeled data is kept the same as that in the whole dataset.

We implemented CS-ST and self-training in Weka, and utilized the code for *NaiveBayes* and *NaiveBayesMultinomial* in Weka. For the 13 UCI datasets, *NaiveBayes* is used as the underlying classifier in all the methods. For the three text datasets, *NaiveBayesMultinomial* classifier is used as the underlying classifier because it is suitable for dealing with text datasets.

The cost of misclassifying a negative instance to be positive (cfp) is set to 2, 5, and 10, respectively, to observe the performance of the three methods in different situations. The average misclassification cost is used as the performance measurement.

5.3.2.3 Results on Thirteen UCI Datasets

Comparison results of the methods when using different cfp values are shown in the sub-tables of Table 5.2. Each value in front of “ \pm ” is the average value of the average misclassification costs computed from the ten runs of five-fold cross-validation, followed by the corresponding standard deviation after “ \pm ”. Row “Mean” depicts the mean value of the average misclassification cost computed over all the datasets of the corresponding column. Row “CS-ST: w/t/l” represents that CS-ST wins on w datasets (marked by \bullet), ties on t datasets, and loses on l datasets (marked by \circ) against the corresponding method, under a two-tailed pair-wise t-test at 95% significance level. Please note that, lower average cost implies better performance.

It can be seen that, CS-ST generally gets significantly smaller or equal average misclassification cost than **SelfTrain** when cfp changes from 2 to 10. It significantly outperforms **SelfTrain** on nine datasets when cfp is 10. Moreover, CS-ST generally obtains much smaller average cost than **SL** except on the “hypothyroid” dataset. The advantage of CS-ST over **SL** is more obvious when cfp is 5 and 10.

To compare the classifiers’ abilities to identify negative instances, the com-

Table 5.2: Average results of AC on 13 UCI datasets(a) $cfp=2$

Dataset	CS-ST	SelfTrain	SL
breast-cancer	0.54 ± 0.15	0.53 ± 0.16	0.57 ± 0.12
breast-w	0.03 ± 0.01	0.03 ± 0.01	0.22 ± 0.06 ●
bupa	0.69 ± 0.12	0.70 ± 0.11	0.72 ± 0.13
clean1	0.70 ± 0.10	0.68 ± 0.15	0.67 ± 0.12
credit-g	0.57 ± 0.05	0.56 ± 0.05	0.58 ± 0.04
hypothyroid	0.17 ± 0.03	0.17 ± 0.01	0.11 ± 0.00 ○
kr-vs-kp	0.50 ± 0.11	0.73 ± 0.03 ●	0.44 ± 0.08
pima-indians	0.51 ± 0.12	0.51 ± 0.12	0.61 ± 0.06
sick	0.13 ± 0.06	0.15 ± 0.06	0.12 ± 0.00
tic-tac-toe	0.65 ± 0.08	0.64 ± 0.10	0.60 ± 0.08
vote	0.17 ± 0.05	0.19 ± 0.16	0.18 ± 0.12
wdbc	0.14 ± 0.17	0.13 ± 0.16	0.33 ± 0.09 ●
spambase	0.28 ± 0.03	0.29 ± 0.11	0.78 ± 0.01 ●
Mean	0.39	0.41	0.46
CS-ST: w/t/l	-	1/12/0	3/9/1

(b) $cfp=5$

Dataset	CS-ST	SelfTrain	SL
breast-cancer	0.97 ± 0.30	1.04 ± 0.30	1.16 ± 0.20 ●
breast-w	0.04 ± 0.03	0.04 ± 0.02	0.52 ± 0.15 ●
bupa	1.16 ± 0.34	1.29 ± 0.31 ●	1.43 ± 0.40 ●
clean1	1.32 ± 0.29	1.24 ± 0.43	1.32 ± 0.35
credit-g	1.13 ± 0.13	1.18 ± 0.10 ●	1.35 ± 0.11 ●
hypothyroid	0.33 ± 0.03	0.33 ± 0.01	0.26 ± 0.00 ○
kr-vs-kp	0.88 ± 0.24	1.44 ± 0.08 ●	0.92 ± 0.21
pima-indians	0.95 ± 0.27	1.00 ± 0.25	1.38 ± 0.15 ●
sick	0.21 ± 0.12	0.28 ± 0.11 ●	0.31 ± 0.00 ●
tic-tac-toe	1.24 ± 0.20	1.30 ± 0.21	1.18 ± 0.19
vote	0.31 ± 0.21	0.36 ± 0.31	0.29 ± 0.25
wdbc	0.24 ± 0.32	0.21 ± 0.31	0.77 ± 0.24 ●
spambase	0.55 ± 0.06	0.63 ± 0.23 ●	1.94 ± 0.03 ●
Mean	0.72	0.80	0.99
CS-ST: w/t/l	-	5/8/0	8/4/1

(c) $cfp=10$

Dataset	CS-ST	SelfTrain	SL
breast-cancer	1.58 ± 0.56	1.88 ± 0.55 ●	2.14 ± 0.41 ●
breast-w	0.07 ± 0.05	0.06 ± 0.04	1.04 ± 0.30 ●
bupa2	1.80 ± 0.59	2.28 ± 0.64 ●	2.62 ± 0.87 ●
clean1	2.36 ± 0.63	2.18 ± 0.89	2.39 ± 0.76
credit-g	1.99 ± 0.25	2.20 ± 0.19 ●	2.62 ± 0.25 ●
hypothyroid	0.57 ± 0.04	0.58 ± 0.02	0.53 ± 0.01 ○
kr-vs-kp	1.53 ± 0.44	2.62 ± 0.15 ●	1.70 ± 0.44 ●
pima-indians	1.51 ± 0.52	1.82 ± 0.48 ●	2.67 ± 0.31 ●
sick	0.33 ± 0.23	0.51 ± 0.19 ●	0.61 ± 0.01 ●
tic-tac-toe	2.17 ± 0.42	2.40 ± 0.41 ●	2.15 ± 0.41
vote	0.45 ± 0.23	0.64 ± 0.58 ●	0.49 ± 0.46
wdbc	0.40 ± 0.58	0.34 ± 0.57	1.51 ± 0.49 ●
spambase	1.00 ± 0.14	1.18 ± 0.43 ●	3.88 ± 0.06 ●
Mean	1.21	1.44	1.87
CS-ST: w/t/l	-	9/4/0	9/3/1

Table 5.3: Average results of TNR on 13 UCI datasets(a) $cfp=2$

Dataset	CS-ST	SelfTrain	SL
breast-cancer	0.46 \pm 0.18	0.44 \pm 0.17	0.34 \pm 0.15 ●
breast-w	0.99 \pm 0.01	0.99 \pm 0.01	0.70 \pm 0.09 ●
bupa	0.56 \pm 0.16	0.53 \pm 0.16	0.43 \pm 0.23 ●
clean1	0.51 \pm 0.16	0.57 \pm 0.22 ○	0.50 \pm 0.19
credit-g	0.32 \pm 0.08	0.32 \pm 0.06	0.15 \pm 0.09 ●
hypothyroid	0.06 \pm 0.07	0.03 \pm 0.03 ●	0.00 \pm 0.01 ●
kr-vs-kp	0.69 \pm 0.09	0.51 \pm 0.03 ●	0.67 \pm 0.10
pima-indians	0.55 \pm 0.13	0.53 \pm 0.13	0.26 \pm 0.10 ●
sick	0.49 \pm 0.34	0.25 \pm 0.28 ●	0.00 \pm 0.01 ●
tic-tac-toe	0.38 \pm 0.12	0.37 \pm 0.12	0.44 \pm 0.13 ○
vote	0.89 \pm 0.07	0.86 \pm 0.14	0.90 \pm 0.11
wdbc	0.91 \pm 0.14	0.93 \pm 0.14 ○	0.60 \pm 0.14 ●
spambase	0.74 \pm 0.04	0.72 \pm 0.10	0.02 \pm 0.01 ●
Mean	0.58	0.54	0.39
CS-ST: w/t/l	-	3/8/2	9/3/1

(b) $cfp=5$

Dataset	CS-ST	SelfTrain	SL
breast-cancer	0.50 \pm 0.17	0.44 \pm 0.17 ●	0.34 \pm 0.15 ●
breast-w	0.99 \pm 0.01	0.99 \pm 0.01	0.70 \pm 0.09 ●
bupa	0.62 \pm 0.18	0.53 \pm 0.16 ●	0.43 \pm 0.23 ●
clean1	0.52 \pm 0.15	0.57 \pm 0.22	0.50 \pm 0.19
credit-g	0.38 \pm 0.09	0.32 \pm 0.06 ●	0.15 \pm 0.09 ●
hypothyroid	0.06 \pm 0.08	0.03 \pm 0.03 ●	0.00 \pm 0.01 ●
kr-vs-kp	0.73 \pm 0.10	0.51 \pm 0.03 ●	0.67 \pm 0.10 ●
pima-indians	0.58 \pm 0.14	0.53 \pm 0.13 ●	0.26 \pm 0.10 ●
sick	0.55 \pm 0.35	0.25 \pm 0.28 ●	0.00 \pm 0.01 ●
tic-tac-toe	0.42 \pm 0.12	0.37 \pm 0.12 ●	0.44 \pm 0.13
vote	0.88 \pm 0.11	0.86 \pm 0.14	0.90 \pm 0.11
wdbc	0.91 \pm 0.14	0.93 \pm 0.14	0.60 \pm 0.14 ●
spambase	0.77 \pm 0.04	0.72 \pm 0.10 ●	0.02 \pm 0.01 ●
Mean	0.61	0.54	0.39
CS-ST: w/t/l	-	9/4/0	10/3/0

(c) $cfp=10$

Dataset	CS-ST	SelfTrain	SL
breast-cancer	0.56 \pm 0.18	0.44 \pm 0.17 ●	0.34 \pm 0.15 ●
breast-w	0.99 \pm 0.01	0.99 \pm 0.01	0.70 \pm 0.09 ●
bupa	0.66 \pm 0.15	0.53 \pm 0.16 ●	0.43 \pm 0.23 ●
clean1	0.52 \pm 0.15	0.57 \pm 0.22	0.50 \pm 0.19
credit-g	0.41 \pm 0.09	0.32 \pm 0.06 ●	0.15 \pm 0.09 ●
hypothyroid	0.07 \pm 0.09	0.03 \pm 0.03 ●	0.00 \pm 0.01 ●
kr-vs-kp	0.73 \pm 0.09	0.51 \pm 0.03 ●	0.67 \pm 0.10 ●
pima-indians	0.64 \pm 0.14	0.53 \pm 0.13 ●	0.26 \pm 0.10 ●
sick	0.60 \pm 0.35	0.25 \pm 0.28 ●	0.00 \pm 0.01 ●
tic-tac-toe	0.45 \pm 0.13	0.37 \pm 0.12 ●	0.44 \pm 0.13
vote	0.91 \pm 0.06	0.86 \pm 0.14 ●	0.90 \pm 0.11
wdbc	0.91 \pm 0.14	0.93 \pm 0.14	0.60 \pm 0.14 ●
spambase	0.77 \pm 0.04	0.72 \pm 0.10 ●	0.02 \pm 0.01 ●
Mean	0.63	0.54	0.39
CS-ST: w/t/l	-	10/3/0	10/3/0

parison results on True Negative Rate (TNR) are shown in Table 5.3. TNR is the ratio of the number of correctly classified negative instances over the total number of negative instances. Higher TNR means that the classifier identifies more negative instances, which is beneficial to reduce the misclassifying cost. Because **SelfTrain** does not consider misclassification cost during classifier learning process, the classifier is the same when cfp changes and hence TNR values are not affected by using different cfp values. The situation is the same for **SL**. It is observed from the table that, when cfp is small, CS-ST wins the other two methods on nine and three datasets, respectively, while loses on one and two datasets, respectively. However, when cfp is 5 or 10, CS-ST can significantly outperform **SelfTrain** and **SL** on nine or ten datasets in terms of TNR .

To summarize the analysis, on the 13 UCI datasets, CS-ST generally has much better performance than **SelfTrain** and **SL** on most of the datasets concerning the average misclassification cost and the true negative rate, when cfp is 2, 5 or 10.

5.3.2.4 Results on Three Text Datasets

In [55], compared to a decision tree classifier built on the labeled data only and a direct-EM method, the presented method CS-EM (using a decision tree classifier with smoothing as the underlying classifier) shows better average misclassification cost only on “*oh0*” while obtaining similar results on ‘*oh5*’ and “*oh10*”, when different cfp values are used. Here, we use the three text

Table 5.4: Average results of AC on three text datasets(a) $cfp=2$

Dataset	CS-ST	SelfTrain	SL
oh0	0.08 \pm 0.08	0.23 \pm 0.11 •	0.21 \pm 0.06 •
oh5	0.37 \pm 0.22	0.49 \pm 0.11 •	0.30 \pm 0.08 ◦
oh10	0.26 \pm 0.09	0.30 \pm 0.08 •	0.27 \pm 0.05
Mean	0.24	0.34	0.26
CS-ST:w/t/l	-	3/0/0	1/1/1

(b) $cfp=5$

Dataset	CS-ST	SelfTrain	SL
oh0	0.12 \pm 0.15	0.28 \pm 0.16 •	0.38 \pm 0.15 •
oh5	0.56 \pm 0.38	0.69 \pm 0.22 •	0.58 \pm 0.14
oh10	0.31 \pm 0.19	0.35 \pm 0.18	0.54 \pm 0.14 •
Mean	0.33	0.44	0.50
CS-ST:w/t/l	-	2/1/0	2/1/0

(c) $cfp=10$

Dataset	CS-ST	SelfTrain	SL
oh0	0.19 \pm 0.27	0.36 \pm 0.28 •	0.64 \pm 0.31 •
oh5	0.88 \pm 0.67	1.00 \pm 0.44	1.04 \pm 0.27
oh10	0.40 \pm 0.37	0.42 \pm 0.36	1.00 \pm 0.29 •
Mean	0.49	0.60	0.89
CS-ST:w/t/l	-	1/2/0	2/1/0

datasets to examine the performance of CS-ST. The comparison results on the average misclassification cost are shown in Table 5.4, when cfp is 2, 5, and 10, respectively.

It is observed that, when cfp is 2, CS-ST significantly outperforms **SelfTrain** on all the three dataset. When cfp is larger, CS-ST significantly outperforms **SelfTrain** on one to two datasets, while having equal performance on the other datasets. While CS-ST wins on one dataset and loses on one dataset over **SL** when cfp is 2, the former significantly outperforms the latter on two datasets and ties on one dataset when cfp is 5 and 10. In other words, when

cfp is larger, CS-ST has more effect to reduce the misclassification cost than the other two methods.

In each row, the lowest average misclassification cost obtained on the dataset is shown in bold font. It is observed that, CS-ST generally obtains the lowest average misclassification cost among the three methods except on “oh5” when cfp is 2. Moreover, CS-ST has much lower mean values on the three datasets than the other two methods. The difference is more obvious when cfp is 10. Therefore, on the three text datasets, CS-ST also generally outperforms **Self-Train** and **SL** on the average misclassification cost when cfp is 2, 5 or 10. The superior performance is obviously observed when cfp is larger.

5.3.3 Summary

In this section, a cost-sensitive self-training method CS-ST is proposed to deal with the situation that the number of labeled data is small and different misclassification errors incur different costs. Naive Bayes is used as the underlying classifier. The expected cost is considered when selecting and labeling unlabeled instances in each iteration of self-training. In order to prevent possible performance degradation, the change of performance on average misclassification cost on the original labeled data is applied to decide whether to add the selected instances to expand the training data.

Our experimental results on 13 UCI datasets and three text datasets show that, CS-ST generally outperforms two base methods in terms of the average misclassification cost and the true negative rate. The advantage of CS-ST is

more obvious when cfp has a larger value.

5.4 A Cost-Sensitive Semi-Supervised Learning Method Based on Bagging

Bagging, also called bootstrap aggregating, is a method to learn multiple versions of a model and use them to get an aggregated model [3]. It is also used for cost-sensitive learning, such as the MetaCost algorithm [14]. Given a training dataset D of size n , the basic steps of applying bagging technique in supervised learning are listed as follows.

1. Generate m new datasets D_i of size $n' \geq n$ by sampling from the original datasets uniformly and with replacement;
2. Build m classifiers on the sampled datasets;
3. For an unseen data instance, combine the predictions of the m classifiers by voting.

When sampling with replacement, an instances may appear multiple times in a new dataset. For a large value of n , D_i may have 63.2% of the distinct instances from D , while the other instances are duplicates [3].

In this section, a new cost-sensitive method “BaggingCSSSL” is presented in semi-supervised learning scenario that uses the bagging technique. A set of semi-supervised learning models are learned from the sampled datasets

and combined for classification when different misclassification errors are associated with different costs.

5.4.1 Algorithm Description

In semi-supervised learning, the number of labeled data L_0 is small. Hence, to generate sampling instances from L_0 might not contribute to the learning of a better classifier. Hence, we obtain sampling instances from the unlabeled data U and combine the sampled dataset with L_0 to learn semi-supervised learning classifiers.

The BaggingCSSSL algorithm is described in Figure 5.5. Firstly, n new datasets U_{S_i} ($i = 1, \dots, n$) are randomly sampled from the unlabeled data U with replacement. U_{S_i} and U have the same size. Secondly, given each combination of the original labeled data L_0 and a unlabeled data U_{S_i} as the input, a cost-sensitive classifier C_{S_i} is leaned by using the $CSAllU$ method that is described in steps 2(a-d). The set of classifiers are output as the final classifiers. To better illustrate the idea of the “BaggingCSSSL” method, its structure is described in Figure 5.6.

In the $CSAllU$ method (step (2) of Figure 5.5), all the unlabeled instances are iteratively assigned labels and used to expand the labeled data. A Naive Bayes classifier is initially built from the original labeled data. For each instance, the prediction probability given by the Naive Bayes classifier is adopted as an estimate of the real conditional probability so as to compute the expected cost. Then the method finds the label that has the smallest

Input: the labeled data L_0 ; the unlabeled data U ; the number n .

Output: a set of classifiers C_{S_i} ($i = 1, 2, \dots, n$).

1. Generate n new datasets U_{S_i} by sampling from U uniformly and with replacement.
2. For $i = 1$ to n ,
 - Apply the following steps of the *CSAllU* method to build a cost-sensitive classifier C_{S_i} from the labeled data L_0 and the unlabeled data U_{S_i} :
 - (a) Initialize $t = 0$.
 - (b) Build a Naive Bayes classifier C_0 from L_0 .
 - (c) While t has not reached the maximum number of iterations,
 - Form a new dataset U' by copying all instances from U_{S_i} .
 - Assign each instance in U' an “optimum” label that has the smallest expected cost computed by Equation 2.1.
 - Form a new training dataset L_{t+1} by combining L_0 and U' .
 - Build a new Naive Bayes classifier C_{t+1} from L_{t+1} .
 - Set $t=t+1$.
 - (d) $C_{S_i} = C_{t-1}$.
3. Return the set of classifiers C_{S_i} .

Figure 5.5: The BaggingCSSSL algorithm

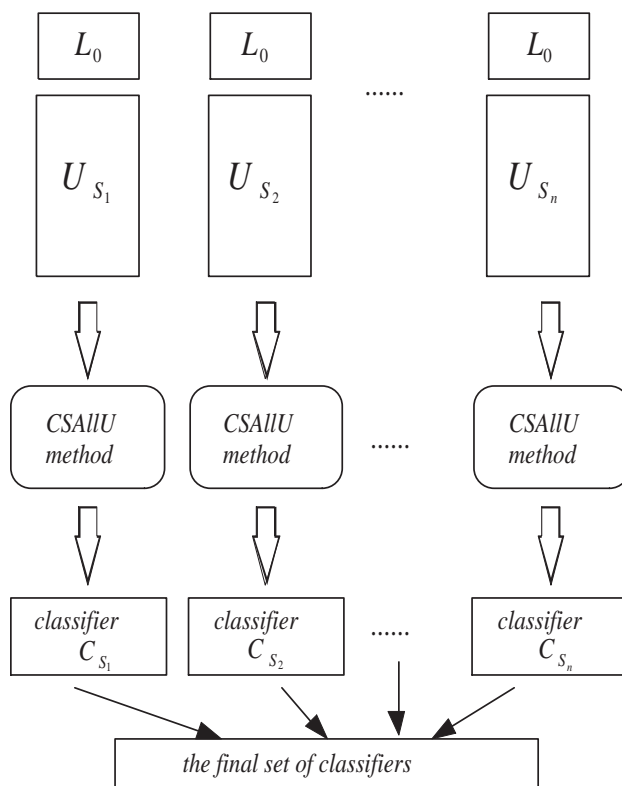


Figure 5.6: The structure of the BaggingCSSSL algorithm

expected misclassification cost for each unlabeled instance to form a new dataset U' . A new Naive Bayes classifier is built from the combination of the original labeled data and U' . The method iterates with updating the labels in U' and building a new classifier until the stopping criterion is satisfied.

After obtaining the set of classifiers, we use them to predict class memberships for unseen testing data instances. For example, suppose the dataset has two classes “+1” and “-1”; and we have three classifiers C_{S_1} , C_{S_2} and C_{S_3} output by the algorithm. For a test instance X , the pair of prediction

probabilities on the two classes generated by the three classifiers are (0.8, 0.2), (0.9, 0.1) and (0.4, 0.6), respectively. Then, when we use the set of classifiers to do prediction on X , $P(+1|X) = \frac{0.8+0.9+0.4}{(0.8+0.9+0.4)+(0.2+0.1+0.6)} = 0.7$, and $P(-1|X) = \frac{0.2+0.1+0.6}{(0.8+0.9+0.4)+(0.2+0.1+0.6)} = 0.3$.

5.4.2 Experiments and Results Analysis

We conduct similar experiments as shown in Section 5.3.2.2. n is set to 10 which is a recommended value for the number of Bootstrap replicates in bagging [3]. Hence, there are 10 classifiers returned by BaggingCSSSL method: C_{S_1}, C_{S_2}, \dots , and $C_{S_{10}}$. The results on 13 UCI datasets and three text datasets are summarized in Table 5.5, when cfp is set to different values. Row “vs CS-ST: w/t/l” shows the t-test results of comparing BaggingCSSSL to CS-ST when using the cfp value displayed in the corresponding column. Row “vs SelfTrain: w/t/l” summarizes the t-test results of comparing BaggingCSSSL to SelfTrain when using the cfp value displayed in the corresponding column. Likewise, row “vs SL: w/t/l” indicates the t-test results of comparing BaggingCSSSL to SL when cfp is set to the value in the corresponding column.

It can be seen from Table 5.5 (a) that, when cfp is small ($cfp=2$), BaggingCSSSL wins CS-ST on two datasets while loses on five datasets, and BaggingCSSSL has equal performance as SelfTrain and SL. However, when cfp increases to 5 and 10, BaggingCSSSL wins CS-ST on six datasets, ties on four datasets, and loses on three datasets. Compared with SelfTrain and

Table 5.5: Average AC of *BaggingCSSSL* method

(a) on UCI datasets

Datasets	<i>BaggingCSSSL</i>		
	$cfp = 2$	$cfp = 5$	$cfp = 10$
breast-cancer	0.56 ± 0.15	0.74 ± 0.10	0.72 ± 0.10
breast-w	0.03 ± 0.01	0.04 ± 0.02	0.06 ± 0.05
bupa	0.64 ± 0.10	0.61 ± 0.05	0.63 ± 0.09
clean1	0.69 ± 0.14	1.32 ± 0.42	2.30 ± 0.86
credit-g	0.59 ± 0.07	0.88 ± 0.11	0.84 ± 0.17
hypothyroid	0.21 ± 0.02	0.50 ± 0.06	0.86 ± 0.09
kr-vs-kp	0.66 ± 0.08	0.81 ± 0.21	0.90 ± 0.15
pima-indians	0.49 ± 0.11	0.64 ± 0.10	0.67 ± 0.03
sick	0.22 ± 0.03	0.51 ± 0.05	0.86 ± 0.05
tic-tac-toe	0.64 ± 0.03	0.65 ± 0.00	0.65 ± 0.00
vote	0.17 ± 0.16	0.25 ± 0.30	0.38 ± 0.52
wdbc	0.10 ± 0.04	0.22 ± 0.09	0.39 ± 0.18
spambase	0.81 ± 0.04	1.46 ± 0.69	0.95 ± 1.11
Mean	0.45	0.66	0.79
vs CS-ST: w/t/l	2/6/5	6/4/3	7/4/2
vs SelfTrain: w/t/l	4/5/4	7/3/3	7/4/2
vs SL: w/t/l	4/4/5	9/2/2	10/1/2

(b) on text datasets

Datasets	<i>BaggingCSSSL</i>		
	$cfp = 2$	$cfp = 5$	$cfp = 10$
oh0	0.19 ± 0.10	0.24 ± 0.12	0.30 ± 0.14
oh5	0.42 ± 0.13	0.55 ± 0.18	0.78 ± 0.36
oh10	0.35 ± 0.06	0.39 ± 0.16	0.47 ± 0.31
Mean	0.32	0.40	0.52
vs CS-ST: w/t/l	0/0/3	0/1/2	0/3/0
vs SelfTrain: w/t/l	2/0/1	1/1/1	0/3/0
vs SL: w/t/l	1/0/2	2/1/0	2/1/0

Table 5.6: Average TNR of *BaggingCSSSL* method on 13 UCI datasets

Datasets	<i>BaggingCSSSL</i>		
	$cfp = 2$	$cfp = 5$	$cfp = 10$
breast-cancer	0.64 ± 0.22	0.96 ± 0.11	0.99 ± 0.03
breast-w	0.99 ± 0.01	0.99 ± 0.01	0.99 ± 0.01
bupa	0.78 ± 0.26	0.98 ± 0.02	0.99 ± 0.02
clean1	0.53 ± 0.22	0.53 ± 0.22	0.54 ± 0.21
credit-g	0.43 ± 0.16	0.80 ± 0.11	0.95 ± 0.06
hypothyroid	0.06 ± 0.04	0.19 ± 0.07	0.79 ± 0.24
kr-vs-kp	0.62 ± 0.13	0.82 ± 0.10	0.90 ± 0.03
pima-indians	0.72 ± 0.15	0.98 ± 0.07	0.99 ± 0.01
sick	0.07 ± 0.06	0.20 ± 0.21	0.92 ± 0.12
tic-tac-toe	0.92 ± 0.16	1.00 ± 0.00	1.00 ± 0.00
vote	0.93 ± 0.13	0.93 ± 0.12	0.93 ± 0.12
wdbc	0.89 ± 0.05	0.90 ± 0.05	0.90 ± 0.05
spambase	0.01 ± 0.05	0.29 ± 0.38	0.86 ± 0.32
Mean	0.58	0.74	0.90
vs CS-ST: w/t/l	6/4/3	8/3/2	10/3/0
vs SelfTrain: w/t/l	8/3/2	8/4/1	10/3/0
vs SL: w/t/l	10/2/1	12/1/0	12/1/0

SL, BaggingCSSSL gets lower average misclassification costs on seven or nine datasets. When cfp is 10, BaggingCSSSL reduces the misclassification costs on more datasets than the other three methods. Therefore, BaggingCSSSL shows better performance when cfp increases.

In terms of finding true negative instances, the advantage of BaggingCSSSL is observed in Table 5.6. The results indicate that, generally, BaggingCSSSL significantly predicts more negative instances than CS-ST, SelfTrain, and SL. When cfp is 10, the TNR values of BaggingCSSSL computed on most of the 13 UCI datasets are close to 1.0, and the average result of TNR on the 13 UCI datasets is 0.90. On the “tic-tac-toe” dataset, BaggingCSSSL correctly predicts all the negative instances ($TNR = 1.0$) when cfp is 5 and 10.

As displayed in Table 5.5 (b), the comparison results on the three text datasets indicate that, BaggingCSSSL gets equal or worse performance than

CS-ST with regards to the average misclassification cost. Compared to Self-Train, using bagging technique has better performance on one or two datasets while losing on one dataset when cfp is 2 and 5. Compared to SL, the performance of BaggingCSSSL is improved as cfp increases. BaggingCSSSL has more advantage to get a low average misclassification cost among the four methods when cfp is 10.

5.4.3 Summary

A bagging based cost-sensitive semi-supervised learning algorithm is presented in this section. A set of classifiers are learned and saved for prediction on unseen data. Since each classifier is trained from the combination of the labeled data and a different set of unlabeled data, diversity is introduced into the final set of classifiers. The experimental results show that, BaggingCSSSL has smaller misclassification cost than the single-model method and the two base methods. Moreover, the bagging based method identifies more negative instances that have a small portion in the dataset.

5.5 A Cost-Sensitive Semi-Supervised Learning Method Using Multiple Classifiers with Randomly Selected Unlabeled Instances

In this section, a method “MultiRSCSSSL” is presented to use multiple cost-sensitive self-training classifiers when randomly selecting the unlabeled instances.

5.5.1 Algorithm Description

Similar to BaggingCSSSL, MultiRSCSSSL generates an ensemble of classifiers to predict probabilities for the unseen test data. The MultiRSCSSSL algorithm is described in Figure 5.7. In step 1, the CSRanSelU method described in steps 1(a-d) is applied n times on the combination of L_0 and U to learn an ensemble of cost-sensitive classifiers. For an unseen test instance, the prediction probabilities given by the set of classifiers are computed using the same method for BaggingCSSSL in Section 5.4.1. To vividly show the idea of the method, the structure of the “MultiRSCSSSL” method is drawn in Figure 5.8.

In the CSRanSelU method, if we use the EM method or the self-training method with confidence selection for instances, C_{E_i} would be the same in the ensemble. Please recall that, the experimental study in Chapter 3 shows that, using random selection is not necessarily inferior to using confidence

Input: the labeled data L_0 ; the unlabeled data U ; the number n .

Output: an ensemble of classifiers C_{E_i} ($i = 1, 2, \dots, n$).

1. For $i = 1$ to n ,
 - Apply the following steps of the *CSRanSelU* method to build a cost-sensitive classifier C_{E_i} from the labeled data L_0 and the unlabeled data U :
 - (a) Initialize $t = 0$.
 - (b) Build a Naive Bayes classifier C_0 from L_0 .
 - (c) While t has not reached the maximum number of iterations,
 - Randomly select m instances from U and move them out of U .
 - Assign each selected instance an “optimum” label that has the smallest expected cost computed by Equation 2.1.
 - Generate L_{t+1} as the combination of L_0 and the m selected instances.
 - Build a new Naive Bayes classifier C_{t+1} from L_{t+1} .
 - Set $t=t+1$.
 - (d) $C_{E_i} = C_{t-1}$.
2. Return the ensemble of classifiers C_{E_i} .

Figure 5.7: The MultiRSCSSSL algorithm

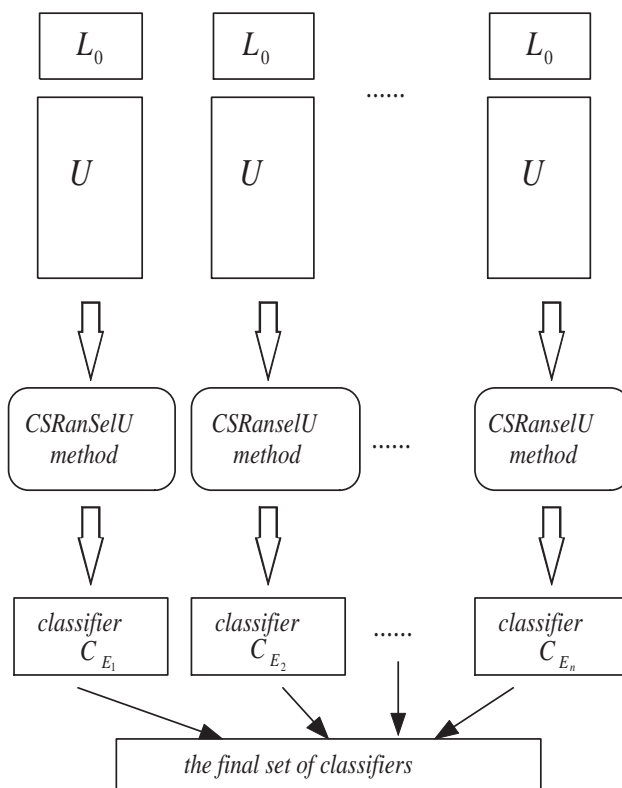


Figure 5.8: The structure of the MultiRSCSSL algorithm

selection in self-training. In order to add more diversity into the ensemble classifiers, we randomly select some unlabeled instances in each iteration of self-training and assign them the labels that have the minimum expected cost.

There are two major differences between BaggingCSSSL and MultiRSCSSL: (1) In MultiRSCSSL, each classifier C_{E_i} is built from the original labeled data L_0 and the unlabeled data U ; while each classifier C_{S_i} in BaggingCSSSL is learned from L_0 and an unlabeled dataset U_{S_i} sampled from U . (2) In

MultiRSCSSSL, each classifier is built by calling the CSRanSelU method that randomly select unlabeled instances in self-training; in BaggingCSSSL, each classifier is obtained by calling the CSAllU method that employs uses all the unlabeled instances in each iteration of training.

5.5.2 Experiments and Results Analysis

We conduct experiments on the same datasets as described in Section 5.3.2.2. n is set to 10 here as well. The results on 13 UCI datasets and three text datasets are summarized in Table 5.7, when cfp is set to different values. Figures in the table have similar meanings as introduced in previous sections. The results shown in Table 5.7 (a) indicate that, MultiRSCSSSL generally outperforms SL on five to eight UCI datasets when cfp changes from 2 to 10, although it get worse performance on three datasets when cfp is 2. Compared to SelfTrain that does not consider the misclassification cost when building the classifier, MultiRSCSSSL get significant smaller average misclassification cost on four and six datasets when cfp is set to different values. Hence, it is concluded that, MultiRSCSSSL generally outperforms the two base performances.

It is also observed that, MultiRSCSSSL wins CS-ST on two datasets and loses on three datasets when cfp is 2; and wins on three datasets while losing on one dataset when cfp is 5 or 10. The advantage of ensemble learning is shown only when cfp is set to a larger value.

When comparing MultiRSCSSSL to BaggingCSSSL, the former method sig-

Table 5.7: Average AC of *MultiRSCSSSL* method

(a) on UCI datasets

Datasets	<i>MultiRSCSSSL</i>		
	$cfp = 2$	$cfp = 5$	$cfp = 10$
breast-cancer	0.54 ± 0.16	0.92 ± 0.29	1.13 ± 0.34
breast-w	0.03 ± 0.01	0.04 ± 0.02	0.06 ± 0.04
bupa	0.64 ± 0.10	0.75 ± 0.16	0.75 ± 0.17
clean1	0.68 ± 0.14	1.30 ± 0.42	2.28 ± 0.83
credit-g	0.57 ± 0.04	1.16 ± 0.10	1.83 ± 0.23
hypothyroid	0.17 ± 0.01	0.33 ± 0.01	0.59 ± 0.02
kr-vs-kp	0.65 ± 0.09	1.08 ± 0.22	1.44 ± 0.40
pima-indians	0.49 ± 0.10	0.71 ± 0.19	0.89 ± 0.41
sick	0.18 ± 0.03	0.36 ± 0.05	0.60 ± 0.12
tic-tac-toe	0.62 ± 0.09	0.85 ± 0.14	0.82 ± 0.18
vote	0.16 ± 0.16	0.22 ± 0.12	0.32 ± 0.15
wdbc	0.10 ± 0.04	0.21 ± 0.09	0.38 ± 0.18
spambase	0.32 ± 0.02	0.58 ± 0.04	0.90 ± 0.07
Mean	0.40	0.65	0.92
vs BaggingCSSSL: w/t/l	5/8/0	3/7/3	2/8/3
vs CS-ST: w/t/l	2/8/3	3/9/1	3/9/1
vs SelfTrain: w/t/l	4/7/2	4/9/0	6/7/0
vs SL: w/t/l	5/5/3	7/5/1	8/4/1

(b) on text datasets

Datasets	<i>MultiRSCSSSL</i>		
	$cfp = 2$	$cfp = 5$	$cfp = 10$
oh0	0.15 ± 0.15	0.29 ± 0.35	0.48 ± 0.65
oh5	0.48 ± 0.10	0.80 ± 0.18	1.32 ± 0.34
oh10	0.28 ± 0.06	0.31 ± 0.10	0.38 ± 0.23
Mean	0.31	0.47	0.73
vs BaggingCSSSL: w/t/l	1/2/0	0/2/1	0/2/1
vs CS-ST: w/t/l	0/1/2	0/1/2	0/1/2
vs SelfTrain: w/t/l	0/3/0	0/3/0	0/3/0
vs SL: w/t/l	1/1/1	2/0/1	2/0/1

nificantly gets smaller average misclassification cost on five datasets and ties on eight datasets, if cfp is 2. When cfp increases to 5 or 10, the two methods get equivalent performance.

Results recorded in Table 5.7 (a) show that, on the text datasets, MultiRSCSSSL wins SL on one or two datasets and loses on one dataset, while MultiRSCSSSL has similar performance as SelfTrain. The better performance of MultiRSCSSSL over CS-ST on UCI datasets is not feasible on the text datasets: CS-ST outperforms MultiRSCSSSL on two datasets when cfp is 2, 5, and 10, respectively. Compared to BaggingCSSSL, MultiRSCSSSL wins on the “on10” dataset when cfp is 2, but loses on the “oh5” dataset when cfp is 5 and 10. Therefore, on the three text datasets, CS-ST should be applied instead of BaggingCSSSL or MultiRSCSSSL.

5.5.3 Summary

In this section, MultiRSCSSSL, is presented to use multiple self-training classifiers that unlabeled instances are randomly selected and misclassification costs are incorporated during the training process. By randomly selecting unlabeled instances to expand the labeled data part, diversity is introduced into the learned self-training classifiers. Results on the 13 UCI datasets show that MultiRSCSSSL generally outperforms the NB classifier built on the labeled data only and the self-training classifier, concerning the misclassification costs evaluated on the testing data. Moreover, MultiRSCSSSL gets smaller average misclassification costs than CS-ST when cfp is set to

5 or 10. The two methods MultiRSCSSSL and BaggingCSSSL have similar performance on most datasets when cfp is large.

Chapter 6

Learning Bayesian Network Structures from Labeled and Unlabeled Data

In previous chapters, we have conducted research to improve the performance of self-training and co-training when Naive Bayes is used as the underlying classifiers, as well as methods to deal with the cost-sensitive problem. However, the conditional independence assumption of Naive Bayes may not be satisfied in real-world applications. In addition, although Naive Bayes has good classification accuracy, the graph structure of Naive Bayes is less powerful than Bayesian networks to represent the relationship among variables. In this chapter, we study the structure learning of Bayesian networks from labeled data and unlabeled data. The discovered Bayesian networks not only

have good classification performance, but also have good graph structures to represent the relationship among variables.

6.1 Two Definitions of Bayesian Networks

As introduced in Section 2.3, a Bayesian network has a directed acyclic graph (DAG) representing the relationships among variables (nodes) and a conditional probabilities table for each node that represents the conditional probabilities of the node given its parents.

In order to get an understanding of the details of our presented methods, some definitions are described here:

Definition 6.1.1. *In a given network structure, the Markov blanket of a variable X consists of X 's parents, X 's children, and the parents of X 's children [21].*

Conditioned on the set of nodes in the Markov blanket of X , X is independent of all other nodes in the graph. Therefore, for classification tasks, the class node is conditionally independent of all other nodes given its Markov blanket.

Definition 6.1.2. *A v-structure in a directed acyclic graph G is an ordered triple of nodes (x,y,z) such that the directed edges $x \rightarrow y$ and $z \rightarrow y$ are contained in G but there is no adjacent edge between x and z in G [9].*

The skeleton of a DAG is the corresponding undirected graph with the same nodes and edges. The only difference is that the edges are undirected in

the skeleton. Two DAGs are considered equivalent if and only if they have the same skeletons and have the same v-structures that are found from the graphs [65]. A detailed method of finding equivalence class is given in [8].

6.2 Algorithm Description

Our new algorithm named “SSLBN” is depicted in Figure 6.1. In the SSLBN method, we utilize the framework of self-training method to use both the labeled data and the unlabeled data. The main idea is to iterate between structure learning and parameter learning for Bayesian networks. Greedy hill-climbing method is used to search for a graph structure from a labeled training dataset. Firstly, a BN classifier C_t is built on the labeled data L_t . Then, an EM procedure is called to update parameters for C_t , based on the labeled data L_t and the unlabeled data U_t . In other words, we treat the class labels of unlabeled instances as missing values and use EM method to update the parameters of the newly built Bayesian network. And then, the classifier C_t with its new parameters is applied in instance selection of the self-training method. The most confident instances are selected, labeled and moved to the labeled data. We call these steps as one iteration. The process iterates until the maximum number of iterations is reached or there is no unlabeled instances left in U_t .

In step 3(e), the greedy hill-climbing search method starts from the neighborhood structures of C_t to find a better candidate structure C_{t+1} , aiming

Input: labeled data L_0 and unlabeled data U

Output: a Bayesian network classifier C

1. Set t , the iteration counter, to 0.
2. Use the hill climbing search method to learn a Bayesian network C_0 on the labeled data L_0 only.
3. While the stopping criteria are not satisfied,
 - (a) Apply the **EM** method to update parameters for the class node of C_t .
 - (b) Select m unlabeled instances from U_t that C_t has the highest prediction confidence.
 - (c) Use C_t to assign a label to each selected instance.
 - (d) Form L_{t+1} as the combination of L_t and the m selected instances.
 - (e) Use the hill climbing search method to find a new Bayesian Network classifier C_{t+1} on L_{t+1} .
 - (f) Increase t by 1.
4. Return C_{t-1} as the final classifier.

Figure 6.1: The SSLBN algorithm

to keep good structures that have been learned so far. The neighborhood structures of C_t consists of all the BN structures that can be obtained from C_t by one operation of adding, removing, or reversing a directed edge locally. These structures are evaluated by a scoring metric. Bayesian metric is used in the algorithm¹. The candidate structure that can mostly improve the score of the previous structure is adopted as the new starting structure. Then the neighborhood structures of the new starting structure are evaluated. The process iterates until the current structure is not significantly different from the previous structure. The final structure is adopted as C_{t+1} . In step 2, the greedy hill-climbing method works in a similar way except that it initially starts with a randomly generated structure.

In step 3(a), an EM method is applied to update the parameters of the class node in C_t . The EM algorithm is described in Figure 6.2. There are two main steps in the procedure: the E-step and the M-step. In the E-step, sufficient statistics are counted or estimated using probabilistic inference based on classifier C_t and datasets L_t and U_t . Given a dataset with complete information, parameters of a Bayesian network can be computed using Equation 2.3 introduced in Section 2.3. In this section, the labels of instances in U_t are unknown to us. Therefore we need to use some inference method to estimate the parameters on U_t . For an unlabeled instance \tilde{u} , the expected value of

¹The description is available at <http://weka.sourceforge.net/manuals/weka.bn.pdf>. Other scoring metrics can also be used instead.

N_{ijk} is computed as:

$$E(N_{ijk}) = \sum_{\tilde{u}} E(N_{ijk}|\tilde{u}) \quad (6.1)$$

where

$$E(N_{ijk}|\tilde{u}) = p(X_i = k, \Pi_{X_i} = j|\tilde{u}) \quad (6.2)$$

When the values of X_i and Π_{X_i} are not missing in \tilde{u} , the expected value is counted as either 1 if $X_i = k$ and $\Pi_{X_i} = j$ are satisfied in \tilde{u} , or 0 otherwise. If either value is missing in \tilde{u} , $p(X_i = k, \Pi_{X_i} = j|\tilde{u})$ is calculated as $p(X_i = k, \Pi_{X_i} = j|u^*)$ that can be solved by Bayesian inference algorithms such as Junction Tree Inference method [28], where u^* represents the set of all the available values in \tilde{u} . The inference program is usually complex to apply, especially when the number of variables is large. To make it simply here, we use the class membership generated by C_t on u^* as an estimation of $p(X_i = k, \Pi_{X_i} = j|\tilde{u})$. In the M-step, the parameter θ_{ijk} is updated by

$$\theta_{ijk} = \frac{E(N_{ijk})}{\sum_k E(N_{ijk})} \quad (6.3)$$

The two steps iterate until the maximum number of EM iterations has been reached, or the difference of the log-likelihood values of two adjacent iterations is smaller than a pre-defined threshold ε . The second criterion can be formalized as

$$|ll(w) - ll(w - 1)| < \varepsilon \quad (6.4)$$

ε is set to 10^{-4} in SSLBN method. and $ll(w)$ is the log-likelihood of classifier

Input: a Bayesian network classifier C_t

Output: C_t with updated parameters values

1. Set the counter w to 0.
2. While the stopping criteria are not satisfied,
 - (a) E-step: Compute $E(N_{ijk})$ on L_t and U_t using Equation 6.1.
 - (b) M-step: Compute parameter values using Equation 6.3.
 - (c) Compute the log-likelihood $l(w)$ by Equation 6.5.
 - (d) $w = w + 1$.
3. Return C_t with the new parameter values.

Figure 6.2: The EM method for updating parameters

C_t in the w -th iteration of EM, which is computed by

$$l(w) = \sum_{i,j,k} E(N_{ijk}) \times \log(\theta_{ijk}) \quad (6.5)$$

6.3 Experiments on Datasets Generated from Known Bayesian Networks

Given a Bayesian network structure with known parameter values, random instances can be simulated from the distribution represented by the BN. To examine the performance of our method, we generate datasets from known

Bayesian network structures by simulation. After learning Bayesian network classifiers from the generated datasets, we can check the structure differences between the discovered BNs with the original BNs. Here we focus on binary-class datasets with balanced class distributions. Two kinds of known Bayesian networks are used in the experiments: some benchmark Bayesian networks and some artificial Bayesian networks with different numbers of nodes and edges.

6.3.1 Two Kinds of Bayesian Networks

6.3.1.1 Benchmark Bayesian Networks

Table 6.1 shows the information of selected benchmark Bayesian networks and the datasets generated from them. The first column shows the name of each Bayesian network. The second, third and fourth column represent the number of nodes and edges and the edge density in each BN, respectively. For a structure with $|v|$ nodes and $|E|$ edges, the edge density $e = \frac{|E|}{|v| \times (|v|-1)/2}$. The remaining three columns display the name of the dataset, the number of instances in the dataset, and the class attribute, respectively.

Table 6.1: Datasets generated from benchmark Bayesian networks

BN	$ v $	$ E $	e	Dataset	<i>instances</i>	class node
Hailfinder	56	66	0.04	Hailfinder-500	500	WindFieldMt
				Hailfinder-1000	1000	WindFieldMt
Insurance	27	52	0.15	Insurance-AntiTheft-1000	1000	AntiTheft
Win95pts	76	112	0.04	Win95pts-1000	1000	PrtData

“Hailfinder” is obtained from JavaBayes software, given in the examples available at <http://www.cs.cmu.edu/javabayes/JavaBayes-0.346.tar.gz>, and we selected the node “WindFieldMt” as the class node because it has balanced marginal probabilities (0.5276, 0.4723). Concerning “Insurance”², node “Airbag” has balanced marginal distributions thus is used as the class node in the sampled dataset. Similarly, node “PrtData” on the “Win95pts” BN structure is used as the class node in the “Win95pts-1000” dataset.

6.3.1.2 Artificial Bayesian Networks

Although above three Bayesian Networks have large $|v|$ and $|E|$, e is very small for each BN. In order to observe the performance to reconstruct Bayesian network with different edge densities, we randomly generate artificial Bayesian networks with different numbers of nodes and edges and then conducted experiments on datasets generated from them.

Table 6.2: Datasets generated from artificial Bayesian networks

BN	$ v $	$ E $	e	Dataset
c3v10E9	10	9	0.2	c3v10E9-1000
c3v10E18	10	18	0.4	c3v10E18-1000
c3v10E27	10	27	0.6	c3v10E27-1000

We use the open-source java software WEKA to generate artificial BNs. After a BN is generated, the node that has the maximum in-degree (the number of incoming edges) is selected as the class node. Each non-class node has

²“Insurance” and “Win95pts” are obtained from <http://www.cs.huji.ac.il/labs/compbio/Repository/networks.html>

3 states, and the class node has two states. Random values are assigned to the conditional probability tables for each node on one condition that the marginal probability distribution of the class node is close to (0.5, 0.5). Then a data set with 1000 instances is randomly generated from the artificial Bayesian network. Table 6.2 shows the information of the generated Bayesian networks and datasets. When e is too large, most nodes have directed edges to the children of its children which makes the graph too complex. Hence, we only generated BN with very large e for $|v| = 10$.

6.3.2 Measurements

When the underlying Bayesian network structure is known, the performance of a discovered Bayesian network classifier on a given dataset is usually examined from two aspects: (1) the quality of the graph structure; and (2) the classification performance.

Three kinds of measurements are used to examine the performance of the presented method here:

1. The structure differences between the original Bayesian network G_o and the discovered Bayesian network G_d [69] [64]:
 - *Missing*: the number of missing edges that exist on G_o but do not exist on G_d ;
 - *Extra*: the number of extra edges that do not exist on G_o but appear on G_d ;

- *Reversed*: the number of reversed edges that have opposite directions on G_o and G_d ;
 - *SD*: the total number of different edges, which is the summarization of *Missing*, *Extra* and *Reversed*;
 - *SHD*: the number of different edges on the equivalent classes of the two BNs.
2. The structure difference of the Markov blankets of the class node on the original Bayesian network G_o and the discovered Bayesian network G_d : the number of missing edges (*MB_Missing*), the number of extra edges (*MB_Extra*), the number of reversed edges (*MB_Reversed*), and total different edges (*MB_SD*).
 3. Classification measures on separate test sets: *Accuracy*, *True Positive rate (TPR)*, *True Negative rate (TNR)*, *F-measure*, and *AUC* [13]. These measures are described in Appendix B.

For the two structure difference measurements, smaller values imply better performance. For the classification measurements, larger values indicate better performance.

6.3.3 Experimental Settings

Our method is implemented in WEKA, and all the comparison experiments are conducted in WEKA.

Following the symbols used in previous chapters, lp is used again to represent the percentage of labeled data in the training set when generating the labeled data, unlabeled data, and testing data in each fold. For each dataset, we use 10 runs of 10-fold cross-validation. In each fold, 10% of the dataset is saved aside to be the testing set, lp portion of the other 90% data is used as labeled data, and the remaining $(1-lp)$ portion of the 90% data is used as unlabeled data. At the end of each iteration of self-training, the discovered BN is saved as the start candidate structure of hill-climbing search in the next iteration. The maximum number of iterations I is set to be 100. When the algorithm stops, the final BN is evaluated on the test set by computing above measurements.

For datasets generated from “Hailfinder”, we set lp to be 5%, 10% and 15% respectively. For all other datasets, we set lp to be 10%. However, if the accuracy of supervised learning classifier when lp is 10% is very close to that of when lp is 100%, we set lp to be 1%. The reason of doing this is to make sure that there is room for improvement by utilizing unlabeled data.

To examine whether using unlabeled data can really help find better Bayesian networks, we recorded the results on supervised learning which use only the labeled data to train a classifier. These results are represented as $slBN$ for simplicity. We also recorded the results of supervised learning when lp is 100% (which means the amount of unlabeled data is zero and all training data are used as the labeled data), represented as $slBN_{100\%}$, to get the upper bound of the classification performance of semi-supervised learning.

6.3.4 Results and Analysis

6.3.4.1 On Benchmark Bayesian Network Structures

Table 6.3 and Table 6.4 show the average results of 10 runs of 10-fold cross-validation on datasets generated from “Hailfinder” with different sizes and different values of lp are used. The second column (“slBN_100%”) records the results of classifiers that are learned from all the training data (that is, lp is 100%). This column is put here to show the best performance that the classifier can get if all the training data are available to be used. The third column (“slBN”) represents the results of BN classifiers learned from the labeled data L_0 only by using the hill-climbing search method, when lp is 5%. The fourth column (“SSLBN”) represents the results of BN classifiers learned from L_0 and U by applying the SSLBN method, when lp is 5%. Values in the remaining four columns have similar meanings but with different values of lp . Better performance of SSLBN over the classifier learned from the labeled data only (that is, higher values on classification performance measures and lower values on structure qualify measures) are marked in bold font. The contents in other tables in this sub-section have similar meanings and thus we will not explain each of them in the following paragraphs.

It is observed from Table 6.3 that, when lp is 5%, by utilizing the unlabeled data, SSLBN gets higher classification performance in terms of *accuracy*, *TPR*, *F-measure*, and *AUC*; and it obtains higher quality of Bayesian network structures with smaller structure difference concerning *Missing*, *Extra*,

Table 6.3: Average results of 10 runs of 10-fold cross-validation on “Hailfinder-500”

Measures	sIBN_100%	$lp=5\%$		$lp=10\%$		$lp=15\%$	
		sIBN	SSLBN	sIBN	SSLBN	sIBN	SSLBN
<i>Accuracy</i>	73.04%	53.70%	55.12%	55.26%	58.58%	58.14%	60.08%
<i>TPR</i>	0.88	0.57	0.61	0.60	0.69	0.65	0.70
<i>TNR</i>	0.55	0.49	0.49	0.50	0.46	0.51	0.48
<i>F-measure</i>	0.78	0.57	0.59	0.59	0.64	0.62	0.65
<i>AUC</i>	0.77	0.55	0.56	0.58	0.60	0.60	0.63
<i>Missing</i>	15.32	41.45	20.39	30.46	18.97	26.31	17.85
<i>Extra</i>	29.80	104.80	43.03	97.40	43.38	73.40	40.60
<i>Reversed</i>	11.84	10.93	12.43	12.76	11.99	13.01	12.16
<i>SD</i>	56.96	157.18	75.85	140.62	74.34	112.72	70.61
<i>SHD</i>	65.32	158.11	86.84	145.89	86.33	121.13	82.19
<i>MB_Missing</i>	0.04	0.85	0.67	0.50	0.40	0.18	0.14
<i>MB_Extra</i>	2.19	8.68	7.2	9.56	3.79	8.51	4.24
<i>MB_Reversed</i>	0	0	0	0	0	0	0
<i>MB_SD</i>	2.23	9.53	7.87	10.06	4.19	8.69	4.38

Table 6.4: Average results of 10 runs of 10-fold cross-validation on “Hailfinder-1000”

Measures	sIBN_100%	$lp=5\%$		$lp=10\%$		$lp=15\%$	
		sIBN	SSLBN	sIBN	SSLBN	sIBN	SSLBN
<i>Accuracy</i>	70.30%	54.61%	57.24%	55.26%	58.16%	58.06%	59.05%
<i>TPR</i>	0.79	0.57	0.64	0.60	0.70	0.69	0.73
<i>TNR</i>	0.62	0.52	0.50	0.50	0.46	0.47	0.45
<i>F-measure</i>	0.73	0.56	0.60	0.57	0.63	0.62	0.64
<i>AUC</i>	0.73	0.57	0.60	0.58	0.61	0.61	0.62
<i>Missing</i>	15.67	31.40	18.50	24.93	16.62	22.63	15.28
<i>Extra</i>	19.11	98.96	46.18	58.74	45.97	45.13	44.60
<i>Reversed</i>	8.96	13.30	11.58	12.31	11.12	13.29	11.00
<i>SD</i>	43.74	143.66	76.26	95.98	73.71	81.05	70.88
<i>SHD</i>	43.85	148.13	88.36	104.91	86.30	91.09	83.30
<i>MB_Missing</i>	0.00	0.44	0.36	0.12	0.07	0.05	0.04
<i>MB_Extra</i>	2.10	9.51	5.68	7.71	3.41	5.06	3.37
<i>MB_Reversed</i>	0.00	0.00	0.00	0.00	0.00	0.00	0.00
<i>MB_SD</i>	2.10	9.95	6.04	7.83	3.48	5.11	3.41

Table 6.5: Average results of 10 runs of 10-fold cross-validation on “Win95pts-1000” when $lp = 10\%$

Measures	slBN_100%	slBN	SSLBN
<i>Accuracy</i>	99.90%	99.86%	99.57%
<i>TPR</i>	1.00	1.00	1.00
<i>TNR</i>	1.00	1.00	0.99
<i>F-measure</i>	1.00	1.00	1.00
<i>AUC</i>	1.00	1.00	1.00
<i>Missing</i>	26.97	63.35	40.23
<i>Extra</i>	84.47	73.12	86.35
<i>Reversed</i>	18.18	14.46	17.25
<i>SD</i>	129.62	150.93	143.83
<i>SHD</i>	129.90	150.97	143.72
<i>MB_Missing</i>	3.80	3.30	3.79
<i>MB_Extra</i>	2.29	0.40	2.65
<i>MB_Reversed</i>	0	0	0.2
<i>MB_SD</i>	6.09	3.70	6.64

Reversed, *SD*, *SHD*, *MB_Missing*, *MB_Extra*, and *MB_SD*. SSLBN and slBN have equal performance on *MB_Reversed* which is 0. The same observations are found when lp is 10% and 15%. The performance of SSLBN generally improves when lp increases except *TNR*, if we carefully compare the values in the table. Same findings are obtained from Table 6.4 on “Hailfinder-1000”. Hence, on “Hailfinder-500” and “Hailfinder-1000”, SSLBN outperforms slBN by finding Bayesian network structures with smaller structure difference and higher classification performance. And the performance increases when more original labeled data are available.

Table 6.5 shows the average results of 10 runs of 10-fold cross-validation on “Win95pts-1000” with lp equal to 10%. Values in column “slBN_100%” and “slBN” are the measures of the supervised learning Bayesian network classifiers built on the corresponding original labeled data, respectively. It can

Table 6.6: Average results of 10 runs of 10-fold cross-validation on “Insurance-AntiTheft-1000” when $lp = 1\%$

Measures	sIBN_100%	sIBN	SSLBN
<i>Accuracy</i>	89.9%	65.01%	64.42%
<i>TPR</i>	0.81	0.62	0.60
<i>TNR</i>	0.96	0.67	0.68
<i>F-measure</i>	0.87	0.59	0.58
<i>AUC</i>	0.94	0.68	0.65
<i>Missing</i>	16.81	33	25.07
<i>Extra</i>	11.53	52.67	18.75
<i>Reversed</i>	10.68	10.67	9.55
<i>SD</i>	39.02	96.34	53.37
<i>SHD</i>	49.25	99.16	57.37
<i>MB_Missing</i>	2.01	4.19	4.26
<i>MB_Extra</i>	0.04	3.78	4.21
<i>MB_Reversed</i>	0.99	0.27	0.14
<i>MB_SD</i>	3.04	8.24	8.61

be found that, SSLBN gets equivalent classification performance as “sIBN” and “sIBN_100%” on accuracy, *TPR*, *TNR*, *F-measure*, and AUC. Although SSLBN gets a larger value of *MB_SD*, the classification accuracy is not seriously affected. The average accuracy values obtained by “sIBN_100%”, “sIBN”, and SSLBN are close to 100%, which means that the three methods correctly predict almost all the instances in the testing sets. Besides, SSLBN gets higher values of *SD* and *SHD* than “sIBN” does. Hence, on the “Win95pts-1000” dataset, by using the unlabeled data, SSLBN can get better Bayesian network structures while keeping the good classification performance.

Table 6.6 shows the average results of 10 runs of 10-fold cross-validation on “Insurance-AntiTheft-1000” with lp equal to 1%. It can be observed that, SSLBN can generally get much smaller values on *SD* and *SHD* than

“slBN”, and close results on the structure difference of Markov Blanket of the class node. For the five classification performance, the results of “slBN” and SSLBN are close to each other. Therefore, on “Insurance-AntiTheft-1000”, SSLBN finds better Bayesian networks structures with a slight lose on the classification accuracy.

6.3.4.2 On Artificial Bayesian Network Structures

Table 6.7 shows the average results of 10 runs of 10-fold cross-validation on the three datasets generated from artificial Bayesian networks with 10 nodes and different numbers of edges, and lp is set to 10%. It is obviously observed from results on “C3V10E9-1000” that, SSLBN has higher values on accuracy, TPR , F -measure, and AUC than “slBN”, while it obtains lower values on SD , SHD , and MB -Missing. Similar observations are found on “C3V10E18-1000” and “C3v10E27-1000”. These observations indicate that, when the labeled data is scarce, SSLBN has better classification performance and discovers Bayesian network structures of higher quality than the classifier learned from the original labeled data only.

When comparing the results on the three datasets generated from artificial Bayesian network structures with the same number of nodes but different number of edges, we can find that, the performance on accuracy and AUC decreases as the number of edges increases, and the structure difference increases when there are more edges in the graph. The performance degradation is probably caused by the increased complexity of the graph structures

Table 6.7: Average results of 10 runs of 10-fold cross-validation on three datasets from artificial networks when $lp = 10\%$

Measures	C3V10E9-1000			C3V10E18-1000			C3v10E27-1000		
	sIBN_100%	sIBN	SSLBN	sIBN_100%	sIBN	SSLBN	sIBN_100%	sIBN	SSLBN
<i>Accuracy</i>	75.04%	60.42%	61.55%	72.44%	57.98%	59.01%	73.88%	55.17%	56.81%
<i>TPR</i>	0.75	0.54	0.59	0.74	0.64	0.60	0.72	0.58	0.57
<i>TNR</i>	0.75	0.66	0.64	0.71	0.52	0.58	0.75	0.52	0.57
<i>F-measure</i>	0.75	0.54	0.58	0.73	0.60	0.60	0.73	0.53	0.56
<i>AUC</i>	0.81	0.63	0.65	0.81	0.60	0.62	0.82	0.57	0.59
<i>Missing</i>	0.01	3.54	2.5	3.26	6.84	4.27	3.19	14.89	9.82
<i>Extra</i>	2.65	4.41	4.45	1.97	2.69	3.27	1.22	1.53	2.33
<i>Reversed</i>	2.12	0.97	1.78	5.03	3.66	5.06	3.01	1.97	4.6
<i>SD</i>	4.78	8.92	8.73	10.26	13.19	12.6	7.42	18.39	16.75
<i>SHD</i>	5.14	10.52	10.03	9.26	13.08	12.66	8.29	20.54	18.85
<i>MB_Missing</i>	0.01	3.67	2.86	3.39	5.75	4.10	2.26	8.22	6.24
<i>MB_Extra</i>	2.65	1.94	3.69	5.97	3.10	5.71	0.69	1.65	5.01
<i>MB_Reversed</i>	2.12	0.78	1.25	2.03	0.90	1.71	0.25	0.53	1.56
<i>MB_SD</i>	4.78	6.39	7.8	11.39	9.75	11.52	3.2	10.4	12.81

which poses great challenge in the structure learning methods.

6.4 Experiments on Datasets with Unknown Bayesian Networks

In above section, the datasets are generated from known Bayesian network structures. Hence, it is easier to evaluate the performance of the discovered Bayesian network structures. In this section, the SSLBN method is tested on several UCI datasets where the Bayesian network structures are unknown. Five UCI datasets with different number of instances, attributes, and classes are shown in Table 6.8. The numbers are the average accuracy computed from ten runs of four-fold cross-validation on each dataset. The value of lp is

Table 6.8: Average accuracy of 10 runs of 4-fold cross-validation on five UCI datasets when $lp = 1\%$

Datasets	attr	classes	size	<i>sLBN</i>	<i>SSLBN</i>	sLNB	selfNB
colic.ORIG	27	2	368	57.31	59.08	62.58	55.79
iris	5	3	150	37.70	61.60	65.21	88.57
kr-vs-kp	37	2	3196	68.15	66.41	70.17	57.57
sick	30	2	3772	94.00	89.31	93.83	87.18
vowel	14	11	990	10.91	15.74	18.65	16.63

Table 6.9: Average accuracy on two UCI datasets with fixed given test sets

Datasets	attr	classes	$ L $	$ U $	$ test $	<i>sLBN</i>	<i>SSLBN</i>	sLNB	selfNB
shuttle	10	7	100	43400	14500	86.64	91.58	88.38	82.79
satimage	37	6	600	3835	2000	77.10	79.30	81.03	79.08

set to 1%. Column “sLBN” represents the accuracy of the Bayesian Network classifier learned on the labeled data only. Column “sLNB” represents the accuracy of the Naive Bayes classifier learned on labeled data only, while column “selfNB” represents the accuracy of self-training using Naive Bayes as the underlying classifier. Numbers in bold font means a better performance than the corresponding base performance.

From Table 6.8, it can be observed that, when Naive Bayes is used, the self-training method **selfNB** outperforms the corresponding base classifier **sLNB** on one dataset but gets worse accuracy on the other four datasets. However, when learning Bayesian networks as the underlying classifier, the self-training method **SSLBN** outperforms its base classifier **sLBN** on three datasets and loses on two datasets.

Table 6.9 shows the average accuracy on “shuttle” and “satimage” datasets.

For each dataset, a training data and a testing data are given separately in UCI Machine Learning Repository [1]. Hence, we randomly split the training set into labeled part (L) and unlabeled part (U) and then conducted experiments on the datasets. The process is repeated five times for each dataset. From Table 6.9, it is observed that, when learning complex Bayesian Network structures, using unlabeled data can help improve the accuracy on the two datasets (shown in columns “slBN” and “SSLBN”); while using unlabeled data degrade the performance when Naive Bayes is used as the underlying classifier (shown in columns “slNB” and “selfNB”). The results suggest that, when unlabeled data can not work to help get a good performance of self-training classifier using Naive Bayes, it might be promising to change the underlying classifier to a Bayesian network.

6.5 Summary

In this section, we present a method called SSLBN for Bayesian network structure learning from both the labeled and unlabeled data. It iterates between learning better structures and find better parameters for the structures. Experiments on datasets simulated from some known Bayesian network structures show that, our method SSLBN can generally discover Bayesian networks with higher quality of structures and better or similar classification performance, compared to the Bayesian network learned from the original labeled data only. The experiments on some UCI datasets without knowing

the graph structures also show that, SSLBN might be an alternative when unlabeled data can not help improve the performance of self-training that uses Naive Bayes.

Chapter 7

Conclusion and Future Work

The main results of this dissertation are summarized in Section 7.1. In Section 7.2, some interesting directions are pointed out for future research.

7.1 Thesis Summary

- In Chapter 3, the performance of several Bayesian network classifiers in some semi-supervised learning methods are examined by an extensive empirical study, as well as the comparison to a graph-based semi-supervised learning method and the Transductive SVM method. The results show that, using unlabeled data helps to build better classifiers on some datasets. There are some interesting observations as well. In the self-training and co-training framework, selecting the unlabeled instances with high prediction confidence is not necessarily superior to

randomly selecting the unlabeled instances. Besides, unlabeled data might hurt the classification performance on some datasets.

- In Chapter 4, a new method ISBOLD is presented to improve the performance of self-training and co-training when Naive Bayes is used as the underlying classifier. By not adding into the training data some unlabeled instances that hurt the classification accuracy on the original labeled data, the method could prevent degradation of the classification performance. The experimental results demonstrate that ISBOLD outperforms the standard self-training and co-training methods.
- Chapter 5 proposes three learning methods when the labeled data is insufficient and the classes have different misclassification costs. The single-model cost-sensitive self-training method, CS-ST, incorporates the misclassification costs into the training process of self-training and uses the change of average cost on the original labeled data to help select unlabeled instances into the training data. Two methods using multiple classifiers, *BaggingCSSSL* and *MultiRSCSSSL*, build a set of different classifiers to reduce the variance of classification results. Results on UCI datasets show that, the three new methods could get smaller average misclassification cost while correctly predicting more negative (minority) instances that have a higher misclassification cost than the instances of other classes.
- In Chapter 6, we present a method SSLBN to learn Bayesian network

structures in semi-supervised learning scenario. A hill-climbing search method is used in the underlying classifier to discover Bayesian Networks from a dataset. SSLBN utilizes the framework of self-training to expand the training data and employs EM to update parameters in the graph structures. Experiments are conducted on datasets generated from some benchmark structures and artificial structures, and some UCI datasets with unknown structures. Results show that, SSLBN could find better Bayesian networks with smaller structure differences and higher classification accuracy, compared to the base BN classifier that is learned on the original labeled data only.

7.2 Future Work

This section describes some possible future work of the dissertation.

In Chapter 5, the cost-sensitive semi-supervised learning methods presented use Naive Bayes as the underlying classifier. Other classifiers such as decision tree classifiers could be used instead in the methods. More experiments will be conducted to examine the performance of using other classifiers. These cost-sensitive learning methods will also be extended for the situation that cfp is unknown.

In Chapter 6, we use the greedy hill-climbing method to search for Bayesian network structures. Apparently, more sophisticated structure learning methods could be applied in this step. Besides, conditional likelihood could be

used as the scoring metric to evaluate Bayesian network structures. This is one direction for our future work. SSLBN will be extended to different learning models to deal with cost-sensitive problems in our future work. Moreover, concerning the complexity of Bayesian network structures, it might be helpful to firstly learn a graph from the combination of labeled data and unlabeled data without the class attribute, and to fine tune the edges connected to the class node. Another possible way is to search in the space of the equivalence classes instead of directed acyclic graphs. In the future work, we will study how to better utilize the information contained in the unlabeled data. Concerning the real-world applications, the proposed methods may be applied in high-dimensional datasets such as gene expression data.

Bibliography

- [1] A. Asuncion and D. J. Newman, *UCI Machine Learning Repository*, 2007.
- [2] A. Blum and T. Mitchell, *Combing labeled and unlabeled data with co-training*, Proceedings of the 11th annual conference on Computational Learning Theory, 1998, pp. 92–100.
- [3] L. Breiman, *Bagging predictors*, Machine Learning **24** (1996), 123–140.
- [4] O. Chapelle, B. Schölkopf, and A. Zien (eds.), *Semi-supervised learning*, MIT Press, Cambridge, MA, 2006.
- [5] N.V. Chawla and G. Karakoulas, *Learning from labeled and unlabeled data: An empirical study across techniques and domains*, Journal of Artificial Intelligence Research **23** (2005), 331–366.
- [6] J. Cheng, R. Greiner, J. Kelly, D. Bell, and W. Liu, *learning bayesian networks from data: an information-theory based approach*, Artificial Intelligence **137** (2002), 43–90.

- [7] D. Chickering, D. Geiger, and D. Heckerman, *Learning bayesian networks is np-hard*, Tech. report, Microsoft Research, 1994.
- [8] D. M. Chickering, *A transformational characterization of equivalent bayesian network structures*, Proceedings of the Eleventh conference on Uncertainty in artificial intelligence, 1995.
- [9] ———, *Learning equivalence classes of bayesian-network structures*, Journal of Machine Learning Research **2** (2002), 445–498.
- [10] I. Cohen, N. Sebe, F. Cozman, M.C. Cirelo, and T.S. Huang, *Learning bayesian network classifiers for facial expression recognition using both labeled and unlabeled data*, Proceedings of the 2003 IEEE Computer Society conference on Computer Vision and Pattern Recognition, 2003.
- [11] F.G. Cozman and I. Cohen, *Unlabeled data can degrade classification performance of generative classifiers*, Proceedings of the 15th International Florida Artificial Intelligence Research Society Conference, 2002, pp. 327–331.
- [12] F.G. Cozman, I. Cohen, and M.C. Cirelo, *Semi-supervised learning of mixture models and bayesian networks*, Proceedings of the Twentieth International Conference of Machine Learning, 2003.
- [13] A. Djebbari, Z. Liu, S. Phan, and F. Famili, *An ensemble machine learning approach to predict survival in breast cancer*, International Journal of Computational Biology and Drug Design **1** (2008), no. 3, 275–294.

- [14] P. Domingos, *MetaCost: A general method for making classifiers cost-sensitive*, Proceedings of the 5th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 1999, pp. 155–164.
- [15] P. Donmez and J.G. Carbonell, *Proactive learning: Cost-sensitive active learning with multiple imperfect oracles*, Proceedings of the 17th ACM Conference on Information and Knowledge Management, 2008.
- [16] C. Drummond and R.C. Holte, *C4.5, class imbalance, and cost sensitivity: why under-sampling beats over-sampling*, Workshop on learning from imbalanced datasets, II, ICML, 2003, 2003.
- [17] C. Elkan, *The foundations of cost-sensitive learning*, Proceedings of the 17th International Joint Conference on Artificial Intelligence, 2001, pp. 973–978.
- [18] W. Fan, S.J. Stolfo, J. Zhang, and P.K. Chan, *AdaCost: Misclassification cost-sensitive boosting*, Proceedings of the 16th International Conference on Machine Learning, 1999, pp. 97–105.
- [19] T. Fawcett and A. Niculescu-Mizil, *Pav and the roc convex hull*, Machine Learning **68** (2007), no. 1, 97–106.
- [20] N. Friedman, *Learning belief networks in the presence of missing values and hidden variables*, Proceedings of the Fourteenth International Conference on Machine Learning, 1997.

- [21] N. Friedman, D. Geiger, and M. Goldszmidt, *Bayesian network classifiers*, Machine Learning **29** (1997), 131–163.
- [22] N. Friedman, M. Linial, I. Nachman, and D. Pe'er, *Using bayesian network to analyze expression data*, Proceedings of the Fourth Annual International Conference on Computational Molecular Biology, 2000.
- [23] Y. Guo, X. Niu, and H. Zhang, *An extensive empirical study on semi-supervised learning*, The 10th IEEE International Conference on Data Mining, 2010.
- [24] R. Guzmán-Cabrera, M. Montes y Gómez, P. Rosso, and L. Villaseñor Pineda, *Taking advantage of the web for text classification with imbalanced classes*, MICAI2007: Advances in Artificial Intelligence, 2007, pp. 831–838.
- [25] H. He and E.A. Garcia, *Learning from imbalanced data*, IEEE Transactions on Knowledge and Data Engineering **21** (2009), no. 9, 1263–1284.
- [26] D. Heckerman, *A tutorial on learning bayesian networks*, Tech. report, Microsoft Research Adv. Technol. Div., Redmond, WA, 1995.
- [27] D. Heckerman, D. Geiger, and D. M. Chickering, *Learning Bayesian networks: The combination of knowledge and statistical data*, Machine Learning **20** (1995), 197–243.

- [28] C. Huang and A. Darwiche, *Inference in belief networks: a procedural guide*, International Journal of Approximate Reasoning **15** (1996), 225–263.
- [29] J. Huang and C. X. Ling, *Using auc and accuracy in evaluating learning algorithms*, IEEE Transactions on Knowledge and Data Engineering **17** (2005), no. 3, 299–310.
- [30] T.M. Huang and V. Kecman, *Semi-supervised learning from unbalanced labeled data: An improvement*, International Journal of Knowledge-based and Intelligent Engineering Systems **10** (2006), no. 1, 21–27.
- [31] J.V. Hulse, T.M. Khoshgoftaar, and A. Napolitano, *Experimental perspectives on learning from imbalanced data*, Proceedings of the 24th International Conference on Machine Learning, 2007, pp. 935–942.
- [32] N. Japkowicz, *Learning from imbalanced data sets: A comparison of various strategies*, Proceedings of the AAAI’2000 Workshop on Imbalanced Data Sets, 2000.
- [33] T. Jebara, J. Wang, and S. Chang, *Graph construction and b-matching for semi-supervised learning*, Proceedings of the 26th International Conference on Machine Learning, 2009.
- [34] T. Joachims, *Transductive inference for text classification using support vector machines*, Proceedings of the Sixteenth International Conference on Machine Learning, 1999, pp. 200–209.

- [35] R. Kohavi, *Scaling up the accuracy of Naive-Bayes classifiers: A decision-tree hybrid*, Second International Conference on Knowledge Discovery and Data Mining, 1996, pp. 202–207.
- [36] W. Lam and F. Bacchus, *learning bayesian belief networks: an approach based on the mdl principle*, Computational Intelligence **10** (1994), 269–293.
- [37] A.C. Le, A. Shimazu, V.N. Huynh, and L. M. Nguyen, *Semi-supervised learning integrated with classifier combination for word sense disambiguation*, Computer Speech and Language **21** (2008), 330–345.
- [38] J. Li, X. Li, and X. Yao, *Cost-sensitive classification with genetic programming*, 2005 IEEE Congress on Evolutionary Computation, 2005.
- [39] M. Li and Z.H. Zhou, *SETRED: self-training with editing*, Proceeding of Advances in Knowledge Discovery and Data Mining (PAKDD 2005), 2005, pp. 611–621.
- [40] ———, *Tri-Training: Exploiting unlabeled data using three classifiers*, IEEE Transactions on Knowledge and Data Engineering **17** (2005), no. 11, 1529–1541.
- [41] Y.F. Li, J.T. Kwok, and Z.H. Zhou, *Semi-supervised learning using label mean*, Proceedings of the 26th International Conference on Machine Learning, 2009, pp. 633–640.

- [42] ———, *Cost-sensitive semi-supervised support vector machine*, Proceedings of the 24th AAAI Conference on Artificial Intelligence, 2010, pp. 500–505.
- [43] C.X. Ling, J. Du, and Z.H. Zhou, *When does co-training work in real data?*, Proceedings of the 13th Pacific-Asia Conference on Advances in Knowledge Discovery and Data Mining, 2009, pp. 596–603.
- [44] A. Liu, G. Jun, and J. Ghosh, *A self-training approach to cost sensitive uncertainty sampling*, Machine Learning **76** (2009), 257–270.
- [45] M. A. Maloof, *Learning when data sets are imbalanced and when costs are unequal and unknown*, Workshop on Learning from Imbalanced Datasets II, ICML, 2003.
- [46] D. Margineantu, *Active cost-sensitive learning*, Proceedings of the 19th International Joint Conference on Artificial Intelligence, 2005.
- [47] T.M. Mitchell, *Machine learning*, McGraw-Hill Science, 1997.
- [48] I. Muslea, S. Minton, and C.A. Knoblock, *Active + semi-supervised learning = robust multi-view learning*, Proceedings of the 19th International Conference on Machine Learning, 2002.
- [49] R.E. Neapolitan, *Learning bayesian networks*, Prentice Hall, 2003.

- [50] K. Nigam and R. Ghani, *Analyzing the effectiveness and applicability of co-training*, Proceedings of the 9th international conference on information and knowledge management, 2000, pp. 86–93.
- [51] K. Nigam, A.K. McCallum, S. Thrun, and T. Mitchell, *Text classification from labeled and unlabeled documents using EM*, Machine Learning **39** (2000), 103–134.
- [52] D. Nikovski, *Constructing bayesian networks for medical diagnosis from incomplete and partially correct statistics*, IEEE Transactions on Knowledge and Data Engineering **12** (2000), 509–516.
- [53] F. Provost, *Machine learning from imbalanced data sets 101 (extended abstract)*, Proceedings of the AAAI’2000 Workshop on Imbalanced Data Sets, 2000.
- [54] Y. Qi, *A case study of semi-supervised classification methods for imbalanced data set situation*, Tech. report, Carnegie Mellon University, 2004.
- [55] Z. Qin, S. Zhang, L. Liu, and T. Wang, *Cost-sensitive semi-supervised classification using CS-EM*, Proceedings of the 8th IEEE International Conference on Computer and Information Technology, 2008, pp. 131–136.

- [56] M. Seeger, *Learning with labeled and unlabeled data*, Tech. report, Institute for Adaptive and Neural Computation, University of Edinburgh, 2001.
- [57] V.S. Sheng and C.X. Ling, *Thresholding for making classifiers cost-sensitive*, Proceedings of the 21st National Conference on Artificial Intelligence (AAAI-06), 2006.
- [58] M. Singh, *Learning bayesian networks from incomplete data*, AAAI-97, 1997.
- [59] J. Su, J. Sayyad-Shirabad, and S. Matwin, *Large scale text classification using semi-supervised multinomial naive bayes*, Proceedings of the 28th International Conference on Machine Learning, 2011.
- [60] J. Su and H. Zhang, *Full bayesian network classifiers*, Proceedings of the 23rd International Conference on Machine Learning, 2006.
- [61] J. Su, H. Zhang, C.X. Ling, and S. Matwin, *Discriminative parameter learning for Bayesian networks*, Proceedings of the 25th International Conference on Machine Learning, 2008, pp. 1016–1023.
- [62] Y. Sun, M. Kamel, and A. Wong and Y. Wang, *Cost-sensitive boosting for classification of imbalanced data*, Pattern Recognition **40** (2007), 3358–3378.

- [63] K.M. Ting, *A comparative study of cost-sensitive boosting algorithms*, Proceedings of the 17th International Conference on Machine Learning, 2000, pp. 983–990.
- [64] I. Tsamardinos, L. E. Brown, and C. F. Aliferis, *The max-min hill-climbing bayesian network structure learning algorithm*, Machine Learning **65** (2006), 31–78.
- [65] T. Verma and J. Pearl, *Equivalence and synthesis of causal models*, Proceedings of the Sixth conference on Uncertainty in Artificial Intelligence, 1990.
- [66] B. Wang, B. Spencer, C.X. Ling, and H. Zhang, *Semi-supervised self-training for sentence subjectivity classification*, 21st Canadian Conference on Artificial Intelligence, 2008, pp. 344–355.
- [67] I.H. Witten and E. Frank (eds.), *Data mining: Practical machine learning tools and techniques*, 2nd ed., Morgan Kaufmann, 2005.
- [68] M.L. Wong and Y.Y. Guo, *Learning bayesian networks from incomplete databases using a novel evolutionary algorithm*, Decision Support Systems **45** (2008), 368–383.
- [69] M.L. Wong and K.S. Leung, *An efficient data mining method for learning bayesian networks using an evolutionary algorithm-based hybrid approach*, IEEE Transactions on Evolutionary Computation **8** (2004), 378–404.

- [70] J.C. Xue and G.M. Weiss, *Quantification and semi-supervised classification methods for handling changes in class distribution*, Proceedings of the 15th ACM SIGKDD international conference on knowledge discovery and data mining, 2009, pp. 897–906.
- [71] D. Yarowsky, *Unsupervised word sense disambiguation rivaling supervised methods*, Proceedings of the 33rd Annual Meeting of the Association for Computational Linguistics, 1995, pp. 189–196.
- [72] B. Zadrozny and C. Elkan, *Transforming classifier scores into accurate multiclass probability estimates*, Proceedings of the 8th ACM SIGKDD international conference on Knowledge Discovery and Data Mining, 2002, pp. 694–699.
- [73] H. Zhang, L. Jiang, and J. Su, *Hidden Naive Bayes*, Proceedings of the Twentieth National Conference on Artificial Intelligence (AAAI-05), AAAI Press, 2005, pp. 919–924.
- [74] D. Zhou, O.R Bousquet, T.N. Lal, J. Weston, and B. Schölkopf, *Learning with local and global consistency*, Advances in Neural Information Processing Systems 16, MIT Press, 2004, pp. 321–328.
- [75] Z.H. Zhou and M. Li, *Semi-supervised learning by disagreement*, Knowledge and Information Systems **24** (2010), 415–439.
- [76] X. Zhu, *Semi-supervised learning literature survey*, 2008.

- [77] X. Zhu, Z. Ghahramani, and J. Lafferty, *Semi-supervised learning using gaussian fields and harmonic functions*, Proceedings of the 20th International Conference on Machine Learning, 2003.

Appendix A

Datasets

A.1 UCI Datasets

These UCI datasets are downloaded from WEKA [67] and used for classification tasks. A brief description of the properties of the datasets is shown in Table A.1.

A.2 Benchmark Datasets for Semi-Supervised Learning

In Chapter 3, six benchmark datasets from the book “Semi-supervised learning” [4] are used in the experiments. These benchmark datasets are widely used in semi-supervised learning area. A brief description of the properties of the datasets is shown in Table A.2.

Table A.1: Description of 26 UCI datasets used in the experiments

Dataset	size	#attributes	#classes	frequency of each class
balance-scale	625	5	3	288,49,288
breast-cancer	286	10	2	201,85
breast-w	699	10	2	458,241
colic	368	23	2	232,136
colic.ORIG	368	27	2	244,124
credit-a	690	16	2	307,383
credit-g	1000	21	2	700,300
diabetes	768	9	2	500,268
heart-c	303	14	2	165,138
heart-h	294	14	2	188,106
heart-statlog	270	14	2	150,120
hepatitis	155	20	2	32,123
ionosphere	351	35	2	126,225
iris	150	5	3	50,50,50
kr-vs-kp	3196	37	2	1669,1527
labor	57	17	2	20,37
letter	20000	17	26	all around 770
mushroom	8124	23	2	4208,3916
segment	2310	20	7	all 330
sick	3772	30	2	3541,231
sonar	208	61	2	97,111
splice	3190	61	3	767,768,1655
vehicle	846	19	4	212,217,218,199
vote	435	17	2	267,168
vowel	990	14	11	all 90
waveform-5000	5000	41	3	1692,1653,1655

Table A.2: Description of six benchmark datasets used in the experiments

Dataset	size	#attributes	#classes
g241c	1500	241	2
g241d	1500	241	2
Digit1	1500	241	2
USPS	1500	241	2
COIL	1500	241	6
BCI	400	117	2

Appendix B

Measurements

B.1 Classification Performance Measurements

A confusion matrix is usually used to analyze the classification performance of a classifier on a given dataset. The confusion matrix for binary-class datasets is shown in Figure B.1. “*TP*” (true positive) represents the number of correctly classified positive instances; “*TN*” (true negative) is the number of correctly classified negative instances; “*FP*” (false positive) is the number of instances classified as positive while the true class is negative; and “*FN*” (false negative) denotes the number of instances classified as negative but the true class is positive.

By drawing the confusion matrix, we could compute the classifier’s accuracy (*Acc*), true positive rate (*TPR*), true negative rate (*TNR*), false positive rate (*FPR*), false negative rate (*FNR*), and F-measure. The formulas calculating

		Predicted class	
		P	N
Actual class	P	<i>TP</i>	<i>FN</i>
	N	<i>FP</i>	<i>TN</i>

Figure B.1: Confusion matrix for binary-class datasets

these measurements are listed in following equations:

$$\begin{aligned}
 Acc &= \frac{TP+TN}{TP+TN+FP+FN} \\
 TPR &= \frac{TP}{TP+FN} \\
 TNR &= \frac{TN}{TN+FP} \\
 FPR &= \frac{FP}{TN+FP} \\
 FNR &= \frac{FN}{TP+FN} \\
 F - measure &= \frac{2TP}{2TP+FP+FN}
 \end{aligned} \tag{B.1}$$

AUC, the Area under ROC curve, is also a popular performance measure for classification in addition to above measures [29].

Vita

Candidate's full name: Yuanyuan Guo

University attended:

May 2008 - May 2012, University of New Brunswick
PhD Candidate, Computer Science

September 2003 - June 2006, China University of Geosciences
Master, Computer Application Technology

September 1999 - June 2003, China University of Geosciences
Bachelor, Computer Science and Technology

Publications:

Yuanyuan Guo, Harry Zhang, Bruce Spencer, "Cost-Sensitive Self-training", to appear in The 25th Canadian Conference on Artificial Intelligence, 2012.

Yuanyuan Guo, Harry Zhang, Xiaobo Liu, "Instance Selection in Semi-Supervised Learning", The 24th Canadian Conference on Artificial Intelligence, 2011.

Bin Wang, Harry Zhang, Bruce Spencer and Yuanyuan Guo, "The Unsymmetrical-Style Co-training", Proceedings of the 15th Pacific-Asia Conference on Knowledge Discovery and Data Mining (PAKDD'11).

Yuanyuan Guo, Xiaoda Niu, Harry Zhang, “An Extensive Empirical Study on Semi-supervised Learning”, Proceedings of the 2010 IEEE International Conference on Data Mining (ICDM’10).

Yuanyuan Guo, Man Leung Wong, “Mining Bayesian Networks from Direct Marketing Databases with Missing Values”, Intelligent and Evolutionary Systems: Studies in Computational Intelligence, Springer, **187** (2009), 13-35.

Kwang Mong Sim, Yuanyuan Guo, Benyun Shi, “BLGAN: Bayesian learning and genetic algorithm for supporting negotiation with incomplete information”, IEEE Transactions on Systems, Man, and Cybernetics (Part B), **39** (2009), 198-211.

Man Leung Wong, Yuan Yuan Guo, “Learning Bayesian networks from incomplete databases using a novel evolutionary algorithm”, Decision Support Systems, **45** (2008), 368-383.

Kwang Mong Sim, Yuanyuan Guo, Benyun Shi, “Adaptive bargaining agents that negotiate optimally and rapidly”, Proceedings of IEEE Congress on Evolutionary Computation 2007 (CEC’07).

Man Leung Wong, Yuan Yuan Guo, “Discover Bayesian Networks from Incomplete Data Using a Hybrid Evolutionary Algorithm”, Proceedings of the 6th IEEE International Conference on Data Mining (ICDM’06).

Yuan-Yuan Guo, Man-Leung Wong, Zhi-Hua Cai, “A Novel Hybrid Evolutionary Algorithm for Learning Bayesian Networks from Incomplete Data”, Proceedings of IEEE Congress on Evolutionary Computation 2006 (CEC’06).

Cai Zhihua, Jiang Siwei, Zhu Li, GuoYuanyuan, “A Novel Algorithm of Gene Expression Programming Based on Simulated Annealing”, 2005 International Symposium on Intelligence Computation and Applications (ISICA’2005).