DOMAIN ADAPTATION IN NATURAL LANGUAGE PROCESSING

BY

JING JIANG

B.S., Stanford University, 2002
M.S., Stanford University, 2003

DISSERTATION

Submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy in Computer Science
in the Graduate College of the
University of Illinois at Urbana-Champaign, 2008

Urbana, Illinois

Doctoral Committee:

Assistant Professor ChengXiang Zhai, Chair
Professor Jiawei Han
Professor Dan Roth
Professor Bruce Schatz
Associate Professor Hwee Tou Ng, National University of Singapore

Abstract

With the fast growth of the amount of digitalized texts in recent years, text information management becomes increasingly important in people's daily life. Natural language processing provides the foundation of many modern text information management technologies. For many natural language processing tasks, the state-of-the-art solutions are based on supervised statistical machine learning methods, which require large manually annotated corpora. However, the variations of text in vocabulary, format, style, etc. in different domains and the large amount of human efforts needed to create labeled training data make it practically infeasible to directly apply supervised machine learning methods to natural language processing tasks in new domains. There is therefore a great need to develop special learning algorithms and techniques to adapt classifiers trained on some old domains to a different but related new domain.

This thesis aims at understanding the domain adaptation problem and developing general learning techniques for solving the problem. To understand domain adaptation, a formal analysis is conducted from different perspectives. First, we look at the intrinsic distributional difference between two domains, which leads to an instance weighting solution to domain adaptation. Second, we look at the the extrinsic functional difference between the optimal classifiers for two domains, which leads to a feature selection solution to domain adaptation. Third, we distinguish the domain difference that comes from the old training domain from the difference that comes from the new test domain, and accordingly propose that domain adaptation should consist of two stages.

The instance weighting and feature selection solutions are formally developed into two general and principled frameworks for domain adaptation. Both frameworks modify the objective function of the standard risk minimization framework for supervised learning, and include standard super-

vised learning and semi-supervised learning as special cases. Evaluation of the two frameworks on a number of natural language processing tasks using real data sets demonstrates the effectiveness of the domain adaptation techniques incorporated in the frameworks compared with standard supervised and semi-supervised learning.

Observing that the effectiveness of different domain adaptation techniques varies from data set to data set, we also study different types of domain adaptation and their associations with different domain adaptation techniques. Using perturbed real data sets, we are able to show that different types of domain difference indeed require different domain adaptation techniques. This analysis deepens our understanding of domain adaptation, and potentially helps us select the appropriate techniques for particular domain adaptation problems.

Although we focus on domain adaptation in natural language processing in this thesis, most of the analysis of the problem and the proposed domain adaptation techniques are not restricted to natural language processing problems but can be generally applied to most classification tasks when the training and the test domains differ.

to Kai and my parents

Acknowledgments

It is a great pleasure to express my respect to the many people who have supported me throughout my doctoral study at UIUC.

First of all, I would like to express my deep and sincere gratitude to my advisor, Professor ChengXiang Zhai, whose enthusiasm and vision in research has greatly inspired me, and whose constant guidance and encouragement has made this work possible. Ever since I started working with Cheng, he has been treating me as a peer, pointing me to important research questions while also giving me enough freedom to explore my own interests. Cheng has generously and patiently spent much time discussing research ideas with me. A large part of the solutions presented in this thesis came from those stimulating discussions with him. Most importantly, I learned from Cheng not only how to conduct high-quality research but also how to identify and aim at research problems that will make real impact to this field and the society. In the field of text information management, where theory meets practice, Cheng's research vision and approach has greatly influenced me.

I would also like to thank all the other thesis committee members, Professor Dan Roth, Professor Jiawei Han, Professor Bruce Schatz, and Professor Hwee Tou Ng, for their generous time and commitment. Their constructive comments and suggestions made this thesis complete. Special thanks to Professor Hwee Tou Ng from the National University of Singapore, who made a special trip to UIUC for my final defense. I also want to thank Professor Feng Liang from the Department of Statistics of UIUC for her insightful feedback on my thesis research.

During my doctoral study, I have received help from many colleagues and friends at UIUC. I would like to thank many of the members of the TIMan Group for the various valuable discussions

I had with them as well as the joyful time we spent together in the office: Hui Fang, Tao Tao, Xuehua Shen, Azadeh Shakery, Xu Ling, Qiaozhu Mei, Xuanhui Wang, Xin He, Bin Tan, Younhee Ko, Yue Lu, Duo Zhang, Alex Kotov, Maryam Karimzadehgan, and V.G.Vinod Vydiswaran. I would also like to thank many other members of the DAIS Group, especially Hong Cheng, Jing Gao, and Deng Cai. I also want to thank my roommates Priscilla To and Yaning Yang for sharing our graduate life together at UIUC.

My thesis research would not be possible without the financial support from the BeeSpace project and the MIAS project at UIUC.

Finally, I am grateful to my husband Kai and my parents, whose unceasing love and support has helped me go through all the difficulties to complete this venture. I dedicate this thesis to them.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

The past few decades witnessed an explosion of digitalized textual data in many fields, including scientific, clinical, enterprise, legal, and personal information management. Information retrieval (IR) systems have been deployed in many areas and shown to be very useful in helping users find relevant documents. The most prominent example is the success of Web search engines. However, natural language processing (NLP) techniques can provide users with even more sophisticated services. For example, automatic text summarization, information extraction (IE), and question answering (QA) systems can help users quickly digest the huge amount of relevant information and locate specific information nuggets.

Recent advances in natural language processing have shown that statistical machine learning plays a critical role (Manning and Schütze, 1999). In particular, supervised statistical learning has been successfully applied to many NLP problems, including part-of-speech tagging, chunking, full parsing, dependency parsing, word sense disambiguation, semantic role labeling, named entity recognition, relation extraction, etc., and achieved the state-of-the-art performance. Supervised statistical learning relies on a sufficient amount of manually labeled data for training. For this reason, computational linguists have devoted a large amount of human efforts to creating annotated corpora for different purposes, including the Penn Treebank for part-of-speech tagging and syntactic parsing (Marcus et al., 1993), the Proposition Bank for semantic role labeling (Palmer et al., 2005), a series of data sets from the ACE (Automatic Content Extraction) program for information extraction[1], etc.

A fundamental assumption in supervised statistical learning is that the labeled training data is

[1] http://www.nist.gov/speech/tests/ace/

an independent and identically distributed (i.i.d.) sample drawn from the data distribution where the learned classifier will be applied later to make predictions; otherwise, good performance on the test data cannot be guaranteed even if the training error is low. When dealing with textual data, this assumption usually amounts to requiring the training and the test data to be from the same domain. For example, if one wants to build a named entity recognizer for personal Web logs, ideally she should also train on Web log entries annotated with named entities rather than on, say, financial news articles, because common named entities in Web logs and in financial news can have different properties. However, existing annotated corpora are usually biased toward certain domains, especially the news domain. For example, the Penn Treebank corpus is dominated by financial news from the Wall Street Journal. The Proposition Bank uses the Penn Treebank corpus, and thus has the same bias. The ACE data sets also contain mostly newspaper articles; only recently have texts from Web logs and Web forums been added to the ACE data sets. It has generally been observed that when an NLP classifier (e.g. a part-of-speech tagger, a parser, a word sense disambiguation classifier, or a named entity recognizer) trained on a certain domain is applied to a different domain, the performance of the classifier often substantially degrades (Blitzer et al., 2006; McClosky et al., 2006; Chan and Ng, 2006; Vilain et al., 2007).

An immediate solution to the problem is to create annotated data representative of the new domains. For example, the University of Pennsylvania has developed a corpus of biomedical text that contains a Treebank with syntactic structures, a Propbank with predicate-argument structures, and annotated entities and relations (Kulick et al., 2004). Such annotated corpora in special domains can clearly help computational linguists develop NLP tools tailored for the corresponding domains. However, human annotation is usually very labor-intensive. For example, for the first phase of the Penn Treebank project, it took three years to annotated a 4.5-million-word corpus with part-of-speech information and skeletal syntactic structures (Marcus et al., 1993). Clearly, it is infeasible to create new annotated corpora for all domains. The insufficiency of labeled data is therefore a bottleneck problem that hinders the progress of developing effective NLP tools in various domains.

On the other hand, existing annotated corpora are not entirely useless even if we are not working on the same domain; when working on a special domain such as the biomedical literature domain where labeled data is lacking, it would be a big waste to ignore existing annotated corpora from other domains such as the news domain, because there is still a certain degree of similarity between other domains and the biomedical literature domain. Therefore, in order to address the bottleneck problem with insufficient labeled data, a critical and interesting research question is how to leverage existing labeled data from related domains to help train classifiers in the domain we are interested in. We refer to the domain which does not have sufficient labeled data but which we are interested in as the *target* domain, and the domain(s) where sufficient labeled data is available as the *source* domain(s). We call this problem of exploiting labeled data from the source domains to help train classifiers in the target domain the *domain adaptation* problem.

Despite its importance, the domain adaptation problem only gained much attention in the machine learning and natural language processing research communities very recently. There have been a number of studies on solving special kinds of domain adaptation problems or addressing the problem from different perspectives. For example, Chelba and Acero (2004) studied adaptation of maximum entropy classifiers from one domain to another for the task of recovering capitalization, and proposed to construct a Bayesian prior from the source domain to be used in the target domain. Chan and Ng (2006) studied domain adaptation for word sense disambiguation (WSD), where the domain difference is mainly caused by different class prior probabilities. Daumé III and Marcu (2006) used mixture models and labeled data from both the source and the target domains to discover a common mixture component as well as two domain-specific mixture components, one for each domain. Blitzer et al. (2006) proposed a structural corresponding learning (SCL) algorithm for domain adaptation, which maps domain-specific features to common "pivot" features shared by domains, and to learn in the projected feature space. The SCL algorithm was applied to the task of part-of-speech tagging (Blitzer et al., 2006) and sentiment analysis (Blitzer et al., 2007). Ben-David et al. (2007) and Blitzer et al. (2008) theoretically analyzed the domain adaptation problem based on statistical learning theory, and gave error bounds for the problem.

However, the domain adaptation problem is not yet completely understood, and no single standard method has been identified that is guaranteed to work well for any arbitrary domain adaptation problem. In this thesis, we aim at a deep and thorough understanding of the domain adaptation problem by identifying the causes of domain difference and designing domain adaptation methods accordingly. Because much of the existing work on domain adaptation appeared around the same time as this thesis research was conducted, we postpone a detailed discussion of related work until Chapter 6 at the end of the thesis so that we can relate our work with other existing work and lay out a big picture of the current research on this topic.

To understand domain adaptation, we first observe that the essential difference between two domains lies in the joint distribution of the observed object or instance X and the class label Y. Because statistical learning theory requires the labeled training data to be drawn from the same underlying distribution as the data to be classified, training on a domain that has a different data distribution than the test domain fundamentally violates this basic assumption of statistical learning theory. To alleviate this problem, we observe that one solution is to weigh the training instances based on the difference between the two data distributions of the two domains. To further study how to assign the instance weights, we decompose the joint probability of X and Y into two components, the marginal probability of the observed instance X, and the conditional probability of the class label Y given X. This view of domain adaptation from the perspective of the distributional difference between the two domains leads to our instance weighting framework for domain adaptation. In this instance weighting framework, we make use of all the instances available to us, including both labeled instances from the source domain and labeled and unlabeled instances from the target domain. Applying the instance weighting framework boils down to setting various parameters in the framework. We propose several heuristics to set the parameters. Evaluation of the framework on a number of natural language processing tasks shows that instance weighting is effective for domain adaptation.

We also study domain adaptation from another perspective: we look at how the optimal classification functions for the two domains can be different. In particular, we focus on linear classifiers

because they are widely used in natural language processing and shown to be effective. A natural way to connect the linear classifiers for two domains is to see for which features the weights in the two linear classifiers are close and for which features the weights are far apart from each other. In another word, we hypothesize that two domains may have a set of *generalizable* features that have similar weights in the linear classifiers for the two domains, but there may also be a set of *domain-specific* features that have very different weights in the two linear classifiers. This view of domain adaptation from the perspective of the similarity and dissimilarity between the weight vectors of the two linear classifiers for two domains leads to our feature selection framework for domain adaptation. The feature selection framework allows two domains to have the same weights for the generalizable features but different weights for the domain-specific features. We generalize the framework to the case where we have more than one source domains. We then propose heuristic ways of identifying generalizable features across the multiple training domains. We also use a penalization parameter to force the domain-specific features that are not useful for the target domain to have low weights, in order to shift the focus to those generalizable features. Semi-supervised learning can also be incorporated into the feature selection framework in order for the model to learn weights for those domain-specific features in the target domain. The feature selection framework is evaluated on a number of natural language processing tasks and shown to be effective.

The experiment results from applying the two frameworks show that the effectiveness of difference domain adaptation techniques varies from data set to data set. Motivated by this observation, we then study whether there are different types of domain difference and whether these types require different techniques. We consider domain difference from two perspectives. First, we separate domain difference in $P(X)$ from domain difference in $P(Y|X)$. Second, we separate domain difference caused by "novel" characteristics in the test domain from that caused by "noisy" characteristics in the training domain. These two perspectives give us four different types of domain difference. We then analytically associate a few representative domain adaptation techniques with the domain difference types, and use perturbed real data sets to verify the associations.

Our findings confirm that the different types based on different domain-specific characteristics have strong associations with certain domain adaptation techniques. However, there is no strong evidence to suggest that domain difference in $P(X)$ and domain difference in $P(Y|X)$ require different domain adaptation techniques, probably because difference in $P(X)$ also leads to difference in $P(Y|X)$. The analysis on domain difference types and their associations with domain adaptation techniques has particular practical meaning because it can potentially help us choose the appropriate domain adaptation techniques for specific domain adaptation problems.

The rest of the thesis is organized as follows. We first formally analyze the domain adaptation problem in Chapter 2, from both an intrinsic view and an extrinsic view, leading to the two general solutions based on instance weighting and feature selection, respectively. We then present the two general frameworks in Chapter 3 and Chapter 4 in detail and evaluate them on real data sets. In Chapter 5, we study different types of domain difference and their associations with different domain adaptation techniques. We give a detailed discussion on related work in Chapter 6 with our work included, which summarizes the state-of-the-art solutions to this problem. Finally, in Chapter 7, we conclude our work and point out future directions to explore.

Chapter 2

Understanding Domain Adaptation

In this chapter, we try to analyze and understand the domain adaptation problem in a principled way. Such an analysis allows us to identify the underlying causes of domain difference and to design methods that can directly address the difference. We first introduce some notations and give the setup of the domain adaptation problem in Section 2.1. We then briefly review some basics of statistical learning theory that will be used in the rest of this thesis in Section 2.2. We formally analyze the domain adaptation problem in Section 2.3 from three perspectives. We first discuss domain adaptation from an intrinsic view and an extrinsic view. These two views naturally lead to two solutions to domain adaptation, instance weighting and feature selection, which we will discuss in detail in Chapter 3 and Chapter 4, respectively. We also present a third view which is based on whether the domain difference mainly comes from special characteristics of the target domain or from the source domain. This third view forms the basis of our analysis of domain difference types in Chapter 5.

2.1 Notations and Problem Setup

We first introduce some notations that are used in the rest of the thesis. Recall that we refer to the training domain where labeled data is abundant as the source domain, and the test domain where labeled data is not available or very little as the target domain. Let X denote the input variable (i.e. an observation) and Y the output variable (i.e. a class label). We use $P(X, Y)$ to denote the true underlying joint distribution of X and Y, which is unknown. In domain adaptation, this joint distribution in the target domain differs from that in the source domain. We therefore use

$P_t(X, Y)$ to denote the true underlying joint distribution in the target domain, and $P_s(X, Y)$ to denote that in the source domain. We use $P_t(Y)$, $P_s(Y)$, $P_t(X)$ and $P_s(X)$ to denote the true marginal distributions of Y and X in the target and the source domains, respectively. Similarly, we use $P_t(X|Y)$, $P_s(X|Y)$, $P_t(Y|X)$ and $P_s(Y|X)$ to denote the true conditional distributions in the two domains. We use lowercase x to denote a specific value of X, and lowercase y to denote a specific class label. A specific x is also referred to as an observation, an unlabeled instance or simply an instance. A pair (x, y) is referred to as a labeled instance. Here, $x \in \mathcal{X}$, where $\mathcal{X}$ is the input space, i.e. the set of all possible observations. Similarly, $y \in \mathcal{Y}$, where $\mathcal{Y}$ is the class label set. Without any ambiguity, $P(X = x, Y = y)$ or simply $P(x, y)$ should refer to the joint probability of $X = x$ and $Y = y$. Similarly, $P(X = x)$ (or $P(x)$), $P(Y = y)$ (or $P(y)$), $P(X = x|Y = y)$ (or $P(x|y)$) and $P(Y = y|X = x)$ (or $P(y|x)$) also refer to probabilities rather than distributions.

We assume that there is always a relatively large amount of *labeled* data available in the source domain. We use $D_s = \{(x_i^s, y_i^s)\}_{i=1}^{N_s}$ to denote this set of labeled instances in the source domain. In the target domain, we assume that we always have access to a large amount of *unlabeled* data, and we use $D_{t,u} = \{x_i^{t,u}\}_{i=1}^{N_{t,u}}$ to denote this set of unlabeled instances. Sometimes, we may also have a small amount of labeled data from the target domain, which is denoted as $D_{t,l} = \{(x_i^{t,l}, y_i^{t,l})\}_{i=1}^{N_{t,l}}$. In the case when $D_{t,l}$ is not available, i.e. $N_{t,l} = 0$, we refer to the problem as *unsupervised domain adaptation*, while when $D_{t,l}$ is available, we refer to the problem as *supervised domain adaptation*.

It is worth pointing out that the unlabeled instances $D_{t,u}$ from the target domain are not necessarily the instances on which we need to make predictions later. In another word, we consider an *inductive* learning setting, where the learned classifier can be applied to any unseen instance, rather than a *transductive* learning setting, where we only care about predictions on a pre-specified set of unlabeled instances (Vapnik, 1998; Zhu, 2005). Our ultimate goal is to learn a classifier that works well for instances drawn i.i.d. from the target domain distribution.

While we only consider a single target domain, sometimes, we may have labeled training data from more than one source domains. In this case, exploiting the domain difference in the training

data may also be useful. Suppose there are K ($K > 1$) source domains. We use $P_k(X, Y)$ ($k = 1, 2, \ldots, K$) to denote the joint distribution of X and Y in the k'th source domain. Similarly, we use $P_k(X)$ and $P_k(Y)$ to denote the marginal distributions of X and Y in the source domains, and $P_k(Y|X)$ and $P_k(X|Y)$ to denote the conditional distributions in the source domains. In each source domain, we have a set of labeled instances $D_k = \{(x_i^k, y_i^k)\}_{i=1}^{N_k}$ ($N_k > 0$), drawn i.i.d. from $P_k(X, Y)$. We will see later that having multiple source domains is especially useful in unsupervised domain adaptation, i.e. when $N_{t,l} = 0$.

2.2 Some Basics of Statistical Learning Theory

Before we begin the discussion on the analysis of domain adaptation, let us first briefly review some basics of statistical learning theory that will be used in this thesis. To simplify the notations and the discussion, let us first assume a binary classification setting, i.e. $\mathcal{Y} = \{-1, +1\}$. This simplification should not affect our analysis of domain adaptation. Indeed, multiclass classification problems can always be solved by binary classifiers if we first transform the multiclass classification problem into a number of binary classification problems (Hsu and Lin, 2002; Har-Peled et al., 2003). Furthermore, some statistical learning algorithms can naturally handle multiclass classification, e.g. logistic regression classifiers.

For classification problems, our ultimate goal is to find the functional dependency between the input X and the output Y. We often also seek a prediction confidence value. Let $f : \mathcal{X} \to \mathbb{R}$ be a function that returns the confidence of assigning the class label $+1$ to x. We assign $+1$ to x if $f(x) > 0$, i.e. the classification function is $y = \text{sign}(f(x))$. To find a good function f, we usually first choose a hypothesis space $\mathcal{H}$, which is the set of candidate functions we consider. The optimal choice f^* should then minimize the expected loss with respect to the true distribution $P(X, Y)$:

$$f^* = \arg\min_{f \in \mathcal{H}} \sum_{(x,y) \in \mathcal{X} \times \mathcal{Y}} P(x, y) L(x, y, f),$$

where $L(x, y, f)$ is a loss function. For example, we can use the 0-1 loss function defined as follows:

$$L(x, y, f) = \begin{cases} 1 & \text{if } y = \text{sign}(f(x)) \\ 0 & \text{otherwise} \end{cases} .$$

Another commonly used loss function is the logistic loss function:

$$L(x, y, f) = \ln(1 + \exp(-yf(x))). \tag{2.1}$$

Because the true distribution $P(X, Y)$ is unknown, in practice, we find $\hat{f}$ that minimizes the empirical loss:

$$\begin{aligned} \hat{f} &= \arg\min_{f \in \mathcal{H}} \sum_{(x,y) \in \mathcal{X} \times \mathcal{Y}} \tilde{P}(x, y) L(x, y, f) \\ &= \arg\min_{f \in \mathcal{H}} \frac{1}{N} \sum_{i=1}^{N} L(x_i, y_i, f). \end{aligned}$$

$\tilde{P}(X, Y)$ denotes the empirical joint distribution of X and Y, which can be estimated from a set of labeled instances $\{(x_i, y_i)\}_{i=1}^{N}$ drawn i.i.d. from $P(X, Y)$.

To avoid overfitting, we often add a regularization term $R(f)$ to the objective function to control the smoothness or the complexity of f:

$$\hat{f} = \arg\min_{f \in \mathcal{H}} \left[\frac{1}{N} \sum_{i=1}^{N} L(x_i, y_i, f) + \lambda R(f) \right].$$

The parameter λ is used to balance the tradeoff between the empirical loss and the function complexity, and can be set by cross validation.

Different classification algorithms differ in their choices of the hypothesis space $\mathcal{H}$, the loss function L, and the search strategies for $\hat{f}$. In this thesis, when we need to refer to a specific choice of the hypothesis space, we always choose the hypothesis space that contains linear classification

functions. This choice is reasonable because many classification problems in natural language processing can be solved by linear classifiers. In the meantime we assume that the input variable X is represented by a p-dimensional vector in $\mathbb{R}^p$, i.e. $\mathcal{X} = \mathbb{R}^p$, and we use boldface $\mathbf{X}$ and $\mathbf{x}$ to denote X and x, respectively. In this case, for any $f \in \mathcal{H}$, f is of the following form:

$$f(\mathbf{x}) \quad = \quad \mathbf{w}^T\mathbf{x},$$

where $\mathbf{w} \in \mathbb{R}^p$ is a weight vector, and $\mathbf{w}^T\mathbf{x}$ is the inner product of $\mathbf{w}$ and $\mathbf{x}$. Note that usually we also have a threshold $b \in \mathbb{R}$, that is, $f(\mathbf{x}) = \mathbf{w}^T\mathbf{x} + b$. Here to simplify the notation, we assume that $\mathbf{X}$ has a *bias* feature that is always set to 1. b is then encoded as the weight for the bias feature.

A number of classification algorithms output linear classifiers, including support vector machines (SVMs), logistic regression classifiers (a.k.a. conditional maximum entropy classifiers), perceptrons, etc. In this thesis, when we need to refer to a specific classification algorithm, we always choose to work with logistic regression classifiers, although the general approach to domain adaptation should also apply to other classification algorithms. For logistic regression classifiers, the loss function is chosen to be the logistic loss as defined in Equation (2.1). Thus, with logistic regression, given a sample of labeled instances $\{(\mathbf{x}_i, y_i)\}_{i=1}^N$, the following linear classifier is chosen to be the optimal decision boundary:

$$\hat{\mathbf{w}} \quad = \quad \arg\min_{\mathbf{w}} \left[\frac{1}{N} \sum_{i=1}^{N} \ln(1 + \exp(-y_i(\mathbf{w}^T\mathbf{x}_i))) + \lambda \|\mathbf{w}\|^2 \right], \tag{2.2}$$

where $\lambda\|\mathbf{w}\|^2$ is the regularization term. Equation (2.2) is a convex optimization problem, and can be solved by standard convex optimization procedures.

Logistic regression classifiers can naturally handle multiclass classification. For multiclass classification, there are $|\mathcal{Y}|$ weight vectors associated with a logistic regression classifier, each corresponding to one class label. We use $\mathbf{w}_y$ to denote the weight vector associated with the class label y. Then logistic regression classifiers assume the following parametric model to approximate

$P(Y|\mathbf{X})$:

$$P(y|\mathbf{x}, \mathbf{w}) = \frac{\exp(\mathbf{w}_y^T \mathbf{x})}{\sum_{y' \in \mathcal{Y}} \exp(\mathbf{w}_{y'}^T \mathbf{x})}.$$

When learning the optimal classifier, $-\ln P(y|\mathbf{x}, \mathbf{w})$ is used as the loss function, i.e. risk minimization is equivalent to maximization of data likelihood. The objective function for learning a multiclass logistic regression classifier is then

$$\hat{\mathbf{w}} = \arg\min_{\mathbf{w}} \left[-\frac{1}{N} \sum_{i=1}^{N} \ln \frac{\exp(\mathbf{w}_y^T \mathbf{x})}{\sum_{y' \in \mathcal{Y}} \exp(\mathbf{w}_{y'}^T \mathbf{x})} + \lambda \|\mathbf{w}\|^2 \right],$$

where $\|\mathbf{w}\|^2 = \sum_{y \in \mathcal{Y}} \|\mathbf{w}_y\|^2$.

2.3 Analysis of Domain Adaptation

In this section, we analyze the domain adaptation problem from three perspectives. We first try to understand the intrinsic problem by considering the difference between the joint distributions of X and Y in the source and the target domains, which is the essential cause of the problem. In this intrinsic view, we do not refer to any specific choice of the hypothesis space or the loss function, and therefore the analysis is general to any statistical learning algorithm that minimizes the (regularized) empirical loss. Next, we take an extrinsic view, and directly consider the possible difference between the optimal classification functions for the two domains. In this extrinsic view, we have to refer to a specific hypothesis space. The hypothesis space we consider here is the set of linear classification functions as described above. However, we do not refer to a specific loss function, and therefore the analysis is still applicable to a number of linear classification algorithms such as support vector machines and logistic regression classifiers. The third angle from which we analyze domain adaptation is to see whether the domain difference comes from special characteristics in the target domain or those in the source domain. This analysis suggests that there are two stages in domain adaptation, a *generalization* stage where we want to remove "noise" from

the source domain, and an *adaptation* stage where we want to pick up "novel" characteristics in the target domain.

2.3.1 An Intrinsic View—Joint Distribution

As we have pointed out earlier, the essential cause of the domain adaptation problem is that the joint distribution of X and Y in the target domain is different from that in the source domain, that is, $P_t(X, Y) \neq P_s(X, Y)$. How does this difference affect learning a good classification function for the target domain? Recall that the optimal function f_t^* for the target domain minimizes the expected loss with respect to $P_t(X, Y)$:

$$f_t^* = \arg\min_{f \in \mathcal{H}} \sum_{(x,y) \in \mathcal{X} \times \mathcal{Y}} P_t(x, y) L(x, y, f). \tag{2.3}$$

Usually, we can use $\tilde{P}(X, Y)$ to approximate $P(X, Y)$. Here since we do not have a sufficient amount of labeled instances drawn i.i.d. from $P_t(X, Y)$, we cannot obtain a $\tilde{P}_t(X, Y)$ that is a good approximation of $P_t(X, Y)$.

We have, however, an i.i.d. sample from $P_s(X, Y)$, i.e. a set of labeled instances from the source domain. Since these instances are drawn from $P_s(X, Y)$, the empirical distribution estimated from these instances cannot help us approximate $P_t(X, Y)$. In order to make use of these labeled instances, we can rewrite Equation (2.3) as follows:

$$\begin{aligned} f_t^* &= \arg\min_{f \in \mathcal{H}} \sum_{(x,y) \in \mathcal{X} \times \mathcal{Y}} \frac{P_t(x, y)}{P_s(x, y)} P_s(x, y) L(x, y, f) \\ &\approx \arg\min_{f \in \mathcal{H}} \sum_{(x,y) \in \mathcal{X} \times \mathcal{Y}} \frac{P_t(x, y)}{P_s(x, y)} \tilde{P}_s(x, y) L(x, y, f) \\ &= \arg\min_{f \in \mathcal{H}} \frac{1}{N_s} \sum_{i=1}^{N_s} \frac{P_t(x_i^s, y_i^s)}{P_s(x_i^s, y_i^s)} L(x_i^s, y_i^s, f). \end{aligned} \tag{2.4}$$

As can be seen from Equation (2.4), in order to make use of the labeled instances from the source domain for the purpose of learning a good classification function for the target domain, we need to

weigh each labeled instance (x_i^s, y_i^s) with $\frac{P_t(x_i^s,y_i^s)}{P_s(x_i^s,y_i^s)}$.

Intuitively, $\frac{P_t(x,y)}{P_s(x,y)}$ indicates how different the two domains are at (x, y). This probability ratio cannot be directly computed, however, because estimating the probability $P(x, y)$ is a challenging task, especially in the target domain when labels are not available. In order to simplify the analysis a little, we decompose $P(X, Y)$ into $P(X)P(Y|X)$. $\frac{P_t(x,y)}{P_s(x,y)}$ is then decomposed into $\frac{P_t(x)}{P_s(x)} \cdot \frac{P_t(y|x)}{P_s(y|x)}$. We can now deal with these two ratios separately.

- **Instance Difference**

  The first probability ratio component is $\frac{P_t(x)}{P_s(x)}$. If $P_t(x) \neq P_s(x)$, then $\frac{P_t(x)}{P_s(x)} \neq 1$. Intuitively, an instance that is in a dense region in the source domain but in a sparse region in the target domain should be down-weighted, because this instance is not so representative of the target domain. This case corresponds to $\frac{P_t(x)}{P_s(x)} < 1$. Likewise, an instance that is in a sparse region in the source domain but in a dense region in the target domain should be weighted more, corresponding to $\frac{P_t(x)}{P_s(x)} > 1$. Since Y is not involved in this probability ratio, ideally we can estimate this ratio using only unlabeled instances, that is, we can estimate this ratio in *unsupervised* domain adaptation. We refer to this distributional difference as *instance difference*.

- **Labeling Difference**

  The second probability ratio component is $\frac{P_t(y|x)}{P_s(y|x)}$. If $P_t(y|x) \neq P_s(y|x)$, then $\frac{P_t(y|x)}{P_s(y|x)} \neq 1$, which indicates that the distribution of labels are different in the two domains at x. Since this ratio is related to Y, without any labeled instances from the target domain, it is hard to estimate $P_t(y|x)$ and thus hard to estimate this ratio. Therefore, we can only estimate this ratio with relatively high accuracy in *supervised* domain adaptation. We refer to this distributional difference as *labeling difference*.

Alternatively, we can also decompose $P(X, Y)$ into $P(Y)P(X|Y)$, and analyze $\frac{P_t(x,y)}{P_s(x,y)}$ accordingly. However, we choose not to follow this decomposition for our analysis here for two

reasons. First, if we decompose the joint distribution in this way, then both components, $P(Y)$ and $P(X|Y)$, are related to Y. However, we have little observation of Y in the target domain. As a result, both $\frac{P_t(y)}{P_s(y)}$ and $\frac{P_t(x|y)}{P_s(x|y)}$ will be hard to estimate with only unlabeled instances from the target domain. Second, it is generally observed that discriminative models work better than generative models for classification. In discriminative models, we are interested in $P(Y|X)$ rather than $P(X|Y)$. Therefore, decomposing $P(X,Y)$ into $P(X)P(Y|X)$ makes it easier to place the decomposition in the discriminative model based learning paradigm.

The analysis above naturally leads to an instance weighting approach to domain adaptation, as shown below:

$$f_t^* \approx \arg\min_{f \in \mathcal{H}} \sum_{i=1}^{N_s} \frac{P_t(x_i^s)}{P_s(x_i^s)} \frac{P_t(y_i^s|x_i^s)}{P_s(y_i^s|x_i^s)} L(x_i^s, y_i^s, f). \tag{2.5}$$

This instance weighting approach looks attractive because it points out how we should correctly make use of the labeled instances from the source domain in a principled way. There are, however, a number of challenges when following this instance weighting approach:

1. Estimating $P_t(x)$ and $P_s(x)$ may not be as easy as we expect. One solution is to assume a generative model $P(X|\pi)$ for X, and estimate the parameters π_s and π_t for the two domains using unlabeled instances. However, it may be hard to find a suitable generative model for X, especially when the features are not independent, as is the case in most natural language processing problems. Another solution is to use non-parametric density estimation. However, since we usually work with high-dimensional data in natural language processing problems, non-parametric density estimation may become inaccurate here. In Chapter 3, we will present two other more feasible methods for estimating $\frac{P_t(x)}{P_s(x)}$.

2. Estimating $P_t(y|x)$ is hard without any labeled instances from the target domain. Indeed, our original goal is to estimate $P_t(Y|X)$. The difference is that in the original problem we want to find a parametric model $P(Y|X,\theta)$ to approximate $P_t(Y|X=x)$ for all x, while here we only need to estimate $P_t(y_i^s|x_i^s)$ for $i = 1, 2, \ldots, N_s$. Still, without any labeled

instances from the target domain, we cannot expect to know $P_t(y_i^s|x_i^s)$ because it can be arbitrarily different from $P_s(y_i^s|x_i^s)$. Therefore, we only consider using the ratio $\frac{P_t(y|x)}{P_s(y|x)}$ in the supervised domain adaptation setting. In Chapter 3, we will discuss how to estimate the ratio $\frac{P_t(y|x)}{P_s(y|x)}$ and use it to weigh the source domain instances in detail.

3. When $P_s(x)$ is very small or zero, $\frac{P_t(x)}{P_s(x)}$ may be very large or infinity. Essentially, the problem here is that if there is some region in the input space $\mathcal{X}$ where instances are dense in the target domain but are sparse or have zero probability in the source domain, then we may have only a few or no labeled instances in this region from the labeled training set D_s, although we need many instances in this region to be included in the expected loss function for the estimation to be accurate. One possible solution to this problem is to make use of the unlabeled instances from the target domain.

As we can see, although the instance weighting approach appears attractive because of its clean, principled derivation, the actual execution still remains challenging. In Chapter 3, we will study this instance weighting approach in detail, and address the challenges identified above.

2.3.2 An Extrinsic View—Classification Function

In the intrinsic view discussed above, we do not need to relate to the kind of classification functions we use. We can also directly consider the possible difference between the optimal classification functions for the source and the target domains. Let $f_t^* \in \mathcal{H}$ be the optimal classification function for the target domain, and $f_s^* \in \mathcal{H}$ the optimal classification function for the source domain. If we believe that the two domains are related, then f_t^* must be similar to f_s^* in some way. To model this similarity between f_s^* and f_t^*, we have to refer to the functional form of f_s^* and f_t^*, or in another word, the specific hypothesis space.

Let us consider the space of linear classification functions. To simplify the discussion, we again assume that we deal with binary classification problems. Let $\mathbf{w}_s^*$ and $\mathbf{w}_t^*$ be the parameters

associated with f_s^* and f_t^*, respectively, that is,

$$\begin{aligned} f_s^*(\mathbf{x}) &= \mathbf{w}_s^{*T}\mathbf{x}, \\ f_t^*(\mathbf{x}) &= \mathbf{w}_t^{*T}\mathbf{x}. \end{aligned}$$

There are many different ways to relate $\mathbf{w}_t^*$ with $\mathbf{w}_s^*$. A simple and natural way to connect them is to assume that a subset of the features have very similar wights in $\mathbf{w}_s^*$ and in $\mathbf{w}_t^*$. The intuition behind this assumption is that there are some features whose correlations with Y in the target domain are similar to their correlations with Y in the source domain. On the other hand, there are some other features which have different correlations with Y in the two domains. We formalize this assumption with the following formulas:

$$\mathbf{w}_s^* = A^T\mathbf{v}^* + \mathbf{u}_s^*, \tag{2.6}$$

$$\mathbf{w}_t^* = A^T\mathbf{v}^* + \mathbf{u}_t^*. \tag{2.7}$$

Here A is an $h \times p$ ($h < p$) feature selection matrix. It selects a fixed set of h rows from a p-dimensional vector. For example, the following A matrix select the second and the fifth features from an original 6-dimensional feature vector.

$$A = \begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \end{pmatrix}.$$

What Equation (2.6) and Equation (2.7) mean is that $\mathbf{w}_s^*$ and $\mathbf{w}_t^*$ share a common lower-dimensional weight vector $\mathbf{v}^*$, which can be regarded as the weights for those features that have similar correlations with Y in the two domains. What can be adapted from the source domain to the target domain is exactly this component $\mathbf{v}^*$.

With labeled instances from the source domain, we can obtain an estimated $\mathbf{w}_s^*$. By making the above assumption, intuitively we can use $A\mathbf{w}_s^*$ to approximate $\mathbf{v}^*$, which can then be applied to $A\mathbf{x}$

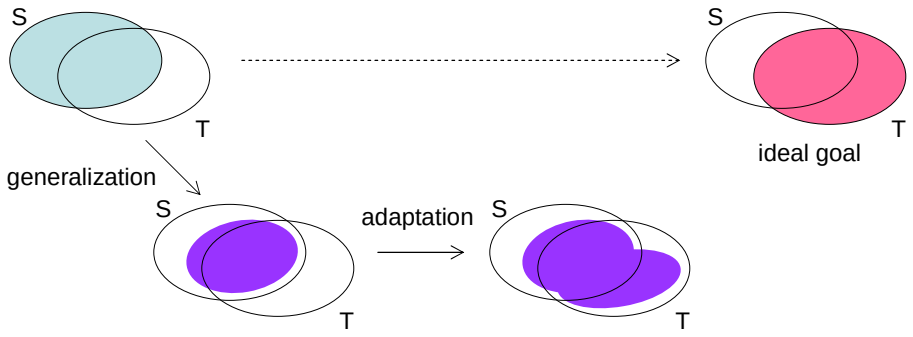

Figure 2.1: An abstract illustration of the two stages of domain adaptation.

to make predictions in either the source or the target domains. The difficulty is how to identify the subset of features that behave similarly in the two domains, i.e. to find the matrix A. In Chapter 4, we will formally present a feature selection framework for domain adaptation with all the details.

2.3.3 Two Stages of Domain Adaptation

Another way to look at how the source and the target domain differ is to separate the difference caused by "novel" characteristics in the target domain from that caused by "noisy" characteristics in the source domain that are not useful in the target domain. This intuition is illustrated in Figure 2.1 in an abstract manner. We use two ovals to represent the source and the target domains, respectively. Because we believe that the two domains are somehow similar, the two ovals have an overlapping region. Because the two domains are still different from each other, there are also two non-overlapping regions in the diagram. Ideally, the optimal classifier for the target domain should cover only the region inside the target oval. However, when training only on the source domain, we obtain a classifier that covers only the source oval. We then need two steps in order to achieve our goal. First, we want to remove the "noise" in the source domain, which can be regarded as identifying the overlapping or the generalizable region between the two domains. Second, we want to adapt to the target domain by reaching out to the non-overlapping region in the target domain. Note that although we separate the two stages in the diagram for illustration purpose, they can also be combined into a single step in learning.

This two-stage idea presented above is fairly abstract. We can relate the idea to the previous two perspectives based on instances and features to make it concrete. In the first perspective, for example, we propose to weigh the source domain instances to address domain adaptation. One challenge we identified is that there may be dense regions in the target domain where we only observe very few or no source domain instances. These regions can be represented by the non-overlapping region in the target domain as shown in Figure 2.1. In order to adapt to the target domain, we then need to include enough target domain instances into the training set at the adaptation stage. In the second perspective, we propose to separate features that are generalizable across domains from features that are domain-specific. The generalizable features can then be regarded as the overlapping region in Figure 2.1, and at the generalization stage, our goal is to identify these generalizable features. In Chapter 3 and Chapter 4, when we fully develop the instance weighting and the feature selection frameworks, we will discuss the details of incorporating the two-stage idea into both frameworks.

Chapter 3

An Instance Weighting Framework for Domain Adaptation

In this chapter, we present an instance weighting framework to address the domain adaptation problem. This framework is motivated by the analysis of domain adaptation in Chapter 2 that looks into the distributional difference between two domains and its effect on empirical risk minimization. Chapter 2 identified some challenges of directly applying the idea of instance weighting as presented in Equation (2.5). In this chapter, we address these challenges by formally developing an instance weighting framework and discuss the details of setting various parameters in this framework. We first review the two kinds of distributional difference between the source and the target domains in Section 3.1. We then formally present the instance weighting framework in Section 3.2. In Section 3.3, we discuss how to set the various weighting parameters in the framework and propose a number of heuristics. We show our experiment results in Section 3.4, and summarize our findings in Section 3.5.

3.1 Two Factors for Domain Adaptation

In Chapter 2, we identified two kinds of distributional difference between the source and the target domains, namely, instance difference and labeling difference. Let us revisit them here. The two kinds of difference are also illustrated in Figure 3.1.

- **Instance Difference**

  The difference between $P_t(X, Y)$ and $P_s(X, Y)$ may come from the difference between $P_t(X)$ and $P_s(X)$. In another word, the source and the target domains may have different dense regions of X. It may first appear that if $P_t(Y|X = x) = P_s(Y|X = x)$ holds for all

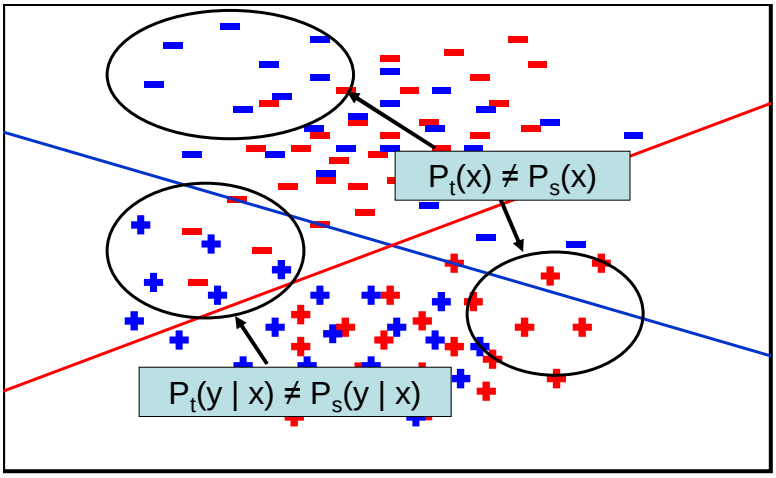

Figure 3.1: An illustration of the two factors for the distributional difference between two domains. The plus signs represent positive instances, and the minus signs represent negative instances. The red signs represent instances from the target domain, while the blue signs represent instances from the source domain. Clearly the two domains have different data distributions, and hence different optimal decision boundaries.

$x \in \mathcal{X}$, then $P_t(X) \neq P_s(X)$ should not cause any problem, because we are only interested in modeling $P(Y|X)$ in classification. However, usually the optimal classification function we choose is only an approximation of $P(Y|X)$. To measure how good a classifier is, we measure its performance averaged over $P(X)$. In another word, we care more about how a classifier performs in dense regions of X than in sparse regions regions of X when choosing the optimal classifier. Therefore, if the target domain has different dense regions of X than the source domain, then the optimal classifier learned from the source domain may no longer be optimal for the target domain.

When the distributional difference between the source and the target domains comes only from $P(X)$, the problem is referred to as *covariate shift* (Shimodaira, 2000) or *sample selection bias* (Heckman, 1979; Zadrozny, 2004). See Chapter 6 for a detailed discussion of related work.

- **Labeling Difference**

The difference between $P_t(X, Y)$ and $P_s(X, Y)$ may also come from the difference between

$P_t(Y|X = x)$ and $P_s(Y|X = x)$ for a considerable number of $x \in \mathcal{X}$. In this case, it is clear that the classifier we learn from the labeled source domain instances may no longer work well in the target domain, especially at those x's where $P_t(Y|X = x)$ is very different from $P_s(Y|X = x)$. Note that unlike the instance difference, labeling difference is hard to detect because we need observations of Y to estimate $P(Y|X)$ but we generally do not have enough labeled instances from the target domain. When we do not have any labeled instance in the target domain, we generally have to make the assumption that $P_t(Y|X = x) = P_s(Y|X = x)$ for all $x \in \mathcal{X}$; when we have a little amount of labeled instances from the target domain, we may be able to detect some difference between $P_t(Y|X)$ and $P_s(Y|X)$.

Note that we can have both kinds of domain difference at the same time. Also note that the degree of either kind of difference may affect the degree of difference between the two domains; if we expect to learn from the source domain for the target domain, neither instance difference nor labeling difference can be large. In another word, the dense regions of X in the two domains should have much overlapping, and there should be a large fraction of $x \in \mathcal{X}$ where $P_t(Y|X = x)$ is very close to $P_s(Y|X = x)$.

3.2 A Framework with Instance Weighting for Domain Adaptation

In this section, we formally present the instance weighting framework. Recall that when we set up the domain adaptation problem in Chapter 2, we introduced three data sets, D_s, $D_{t,l}$, and $D_{t,u}$, where $D_{t,l}$ is optional. Our instance weighting framework makes use of both the labeled instances from the source domain (i.e. D_s) and the unlabeled instances from the target domain (i.e. $D_{t,u}$). If there is a small amount of labeled instances from the target domain (i.e. $D_{t,l}$), the framework will also include them. We first show that each of these three data sets alone can be used to approximate the expected loss in the target domain, but each has its limitations. We then propose to combine

the three approximations into a single objective function, and introduce parameters that can control the contributions of each component.

3.2.1 Using D_s

$D_s = \{(x_i^s, y_i^s)\}_{i=1}^{N_s}$ is the set of labeled instances from the source domain. We assume that D_s is an i.i.d. sample from $P_s(X, Y)$. If we ignore the difference between the source and the target domains, and directly apply the standard supervised learning on D_s to obtain a classifier for the target domain, we do the following:

$$\hat{f}_t = \arg\min_{f \in \mathcal{H}} \sum_{i=1}^{N_s} L(x_i^s, y_i^s, f).$$

However, because of the distributional difference between the two domains, the estimation above is not appropriate for the target domain. As we analyzed in Chapter 2 and showed in Equation (2.5), D_s can be used to approximate the expected loss in the target domain if we weigh the instances appropriately. We rewrite the estimation formula here:

$$\begin{aligned} f_t^* &= \arg\min_{f \in \mathcal{H}} \sum_{(x,y) \in \mathcal{X} \times \mathcal{Y}} \frac{P_t(x,y)}{P_s(x,y)} P_s(x,y) L(x,y,f) \\ &\approx \arg\min_{f \in \mathcal{H}} \sum_{i=1}^{N_s} \frac{P_t(x_i^s)}{P_s(x_i^s)} \frac{P_t(y_i^s|x_i^s)}{P_s(y_i^s|x_i^s)} L(x_i^s, y_i^s, f) \\ &= \arg\min_{f \in \mathcal{H}} \sum_{i=1}^{N_s} \alpha_i \beta_i L(x_i^s, y_i^s, f), \end{aligned} \tag{3.1}$$

where α_i and β_i are defined as follows:

$$\begin{aligned} \alpha_i &\propto \frac{P_t(x_i^s)}{P_s(x_i^s)}, \\ \beta_i &\propto \frac{P_t(y_i^s|x_i^s)}{P_s(y_i^s|x_i^s)}, \\ \sum_{i=1}^{N_s} \alpha_i \beta_i &= N_s. \end{aligned}$$

We will show how to estimate α_i and β_i in Section 3.3.

The major limitation of this approximation is that there may be dense regions of X in the target domain where X is sparse in the source domain. In this case, we may have very few or no labeled instances in D_s to represent these regions. For example, in Figure 3.1, the circle on right hand side represents a region where for an x in this region $P_t(x)$ is large but $P_s(x)$ is very small. As a result, D_s may not contain any instance that can represent this region, but this region may be very critical in defining the optimal decision boundary for the target domain, as illustrated in Figure 3.1. As we will see soon, to address this limitation, we need to use instances from the target domain.

3.2.2 Using $D_{t,l}$

If we have a small set of labeled instances from the target domain, $D_{t,l} = \{(x_i^{t,l}, y_i^{t,l})\}_{i=1}^{N_{t,l}}$, then naturally we can learn a model from $D_{t,l}$:

$$f_t^* = \arg\min_{f \in \mathcal{H}} \sum_{(x,y) \in \mathcal{X} \times \mathcal{Y}} P_t(x,y) L(x,y,f) \tag{3.2}$$

$$\approx \arg\min_{f \in \mathcal{H}} \sum_{(x,y) \in \mathcal{X} \times \mathcal{Y}} \tilde{P}_t(x,y) L(x,y,f) \tag{3.3}$$

$$= \arg\min_{f \in \mathcal{H}} \sum_{i=1}^{N_{t,l}} L(x_i^{t,l}, y_i^{t,l}, f), \tag{3.4}$$

where $\tilde{P}_t(x,y)$ is the empirical probability of (x,y) in the target domain. However, Equation (3.3) is a very crude approximation of Equation (3.2) because $\tilde{P}_t(x,y)$ only has non-zero values at the $N_{t,l}$ instances $\{(x_i^{t,l}, y_i^{t,l})\}_{i=1}^{N_{t,l}}$ and $N_{t,l}$ is relatively small. Therefore, usually, using only the small amount of labeled instances from the target domain for training does not give us a classifier with good performance on the target domain.

3.2.3 Using $D_{t,u}$

We always have a large set of unlabeled instances $D_{t,u} = \{x_i^{t,u}\}_{i=1}^{N_{t,u}}$ from the target domain. Recent advances in semi-supervised learning have shown that unlabeled instances may help classification when valid assumptions are made about the data and appropriate methods are applied (Chapelle et al., 2006; Zhu, 2005). We now show how $D_{t,u}$ may possibly help domain adaptation in the instance weighting framework.

$$\begin{aligned} f_t^* &= \arg\min_{f\in\mathcal{H}} \sum_{(x,y)\in\mathcal{X}\times\mathcal{Y}} P_t(x)P_t(y|x)L(x,y,f) \\ &\approx \arg\min_{f\in\mathcal{H}} \sum_{(x,y)\in\mathcal{X}\times\mathcal{Y}} \tilde{P}_t(x)P_t(y|x)L(x,y,f) \\ &= \arg\min_{f\in\mathcal{H}} \sum_{i=1}^{N_{t,u}} \sum_{y\in\mathcal{Y}} P_t(y|x_i^{t,u})L(x_i^{t,u},y,f) \\ &= \arg\min_{f\in\mathcal{H}} \sum_{i=1}^{N_{t,u}} \sum_{y\in\mathcal{Y}} \gamma_i(y)L(x_i^{t,u},y,f), \end{aligned} \tag{3.5}$$

where $\gamma_i(y)$ is define as follows:

$$\gamma_i(y) = P_t(y|x_i^{t,u}). \tag{3.6}$$

Obviously, we cannot accurately estimate $P_t(y|x_i^{t,u})$ for any $x_i^{t,u}$; otherwise, we would have already solved the original classification problem in the target domain. Therefore, Equation (3.5) alone is unlikely to give good estimation of f_t^* when we have a very crude estimation of $\gamma_i(y)$. However, we hope that even a crude estimation of $P_t(y|x_i^{t,u})$ can still improve the estimation of f_t^*, especially if we combine Equation (3.5) with Equation (3.1) and Equation (3.4), because Equation (3.5) allows us to possibly include instances from the dense regions in the target domain. In Section 3.3, we will discuss in detail heuristic ways to set $\gamma_i(y)$.

3.2.4 Combining D_s, $D_{t,l}$ and $D_{t,u}$

As we have pointed out, each of the three ways to approximately estimate f_t^* has its limitations. However, these three approximations can presumably complement each other. We therefore propose to combine them into a single objective function, as shown below:

$$\begin{aligned}\hat{f}_t &= \arg\min_{f\in\mathcal{H}} \Bigg[\lambda_s \sum_{i=1}^{N_s} \alpha_i\beta_i L(x_i^s, y_i^s, f) + \lambda_{t,l} \sum_{i=1}^{N_{t,l}} L(x_i^{t,l}, y_i^{t,l}, f) \\ &\quad +\lambda_{t,u} \sum_{i=1}^{N_{t,u}} \sum_{y\in\mathcal{Y}} \gamma_i(y) L(x_i^{t,u}, y, f) + \lambda R(f)\Bigg], \end{aligned} \tag{3.7}$$

and

$$\alpha_i \geq 0, \beta_i \geq 0, \sum_{i=1}^{N_s} \alpha_i\beta_i = N_s,$$
$$0 \leq \gamma_i(y) \leq 1.$$

$\lambda R(f)$ is a regularization term to control the complexity of f. λ_s, $\lambda_{t,l}$ and $\lambda_{t,u}$ are parameters used to balance the contributions of the three components.

3.3 Setting the Instance Weights

In the last section, we introduced the instance weighting framework with various instance weighting parameters. In this section, we discuss how to set these weighting parameters in Equation (3.7).

3.3.1 Setting α

As we briefly mentioned earlier, people have studied the covariate shift or sample selection bias problem, where the training and the test domains differ only in $P(X)$. There have therefore been a number of approaches proposed to estimate the ratio $\frac{P_t(X)}{P_s(X)}$ (Shimodaira, 2000; Zadrozny, 2004; Huang et al., 2007; Bickel et al., 2007). We list two methods below, and use these two methods in

our experiments.

- **Logistic Regression:** Zadrozny (2004) proposed to transform the estimation of $\frac{P_t(x)}{P_s(x)}$ into the problem of estimating the probability of x coming from the source domain or from the target domain. More specifically, we can rewrite $\frac{P_t(x)}{P_s(x)}$ as follows. First, let d be a variable that denotes the domain.

$$\begin{aligned}
\frac{P_t(x)}{P_s(x)} &= \frac{P(x|d=t)}{P(x|d=s)} \\
&= \frac{P(d=t|x)P(x)}{P(d=t)} \cdot \frac{P(d=s)}{P(d=s|x)P(x)} \\
&= \frac{P(d=s)}{P(d=t)} \cdot \frac{P(d=t|x)}{P(d=s|x)} \qquad (3.8) \\
&\propto \frac{P(d=t|x)}{P(d=s|x)}. \qquad (3.9)
\end{aligned}$$

In Equation (3.8), because the ratio $\frac{P(d=s)}{P(d=t)}$ is a constant for all instances, we drop this factor. We can treat the estimation of $P(d=t|x)$ and $P(d=s|x)$ as a logistic regression problem. Specifically, let $\mathbf{x}$ be the feature vector representing x. We have

$$\begin{aligned}
P(d=t|x) &= \frac{1}{1+\exp(-\theta^T\mathbf{x})}, \\
P(d=s|x) &= 1 - P(d=t|x) \\
&= \frac{1}{1+\exp(\theta^T\mathbf{x})},
\end{aligned}$$

where θ is the logistic regression model we try to learn. To learn this model, we use the available source and target domain instances, i.e. D_s, $D_{t,l}$ and $D_{t,u}$, that is, we treat instances in D_s as belonging to one class, and instances in $D_{t,l}$ and in $D_{t,u}$ as belonging to the other

class.

$$\hat{\theta} = \arg\min_{\theta} \Bigg[\sum_{i=1}^{N_s} \ln(1+\exp(\theta \mathbf{x}_i^s)) + \sum_{i=1}^{N_{t,l}} \ln(1+\exp(-\theta \mathbf{x}_i^{t,l})) + \sum_{i=1}^{N_{t,u}} \ln(1+\exp(-\theta \mathbf{x}_i^{t,u})) + \lambda \|\theta\|^2 \Bigg],$$

where $\lambda\|\theta\|^2$ is a regularization term.

Once this $\hat{\theta}$ is learned from D_s, $D_{t,l}$ and $D_{t,u}$, we set α_i as follows:

$$\begin{aligned} \alpha_i' &= \frac{P(d=t|x_i^s;\hat{\theta})}{P(d=s|x_i^s;\hat{\theta})} \\ &= \frac{1+\exp(\hat{\theta}^T \mathbf{x}_i^s)}{1+\exp(-\hat{\theta}^T \mathbf{x}_i^s)}, \\ C &= \sum_{i=1}^{N_s} \alpha_i', \\ \alpha_i &= \frac{N_s}{C} \alpha_i'. \end{aligned}$$

- **Mean Matching:** Huang et al. (2007) proposed a kernel mean matching method to address the sample selection bias problem. We present below a slightly modified version of their method. The key idea of the method is to select a set of weights so that the difference between the weighted mean of the source domain instances and the mean of the target domain instances is minimized. The mean of a set of instances is the mean of the feature vectors representing the instances. The difference between two mean vectors is measured by the norm of the difference vector.

  We can therefore solve the following optimization problem in order to obtain a set of weights $\hat{\alpha}_i$:

$$\{\hat{\alpha}_i\}_{i=1}^{N_s} = \arg\min_{\alpha_i} \| \frac{1}{N_s} \sum_{i=1}^{N_s} \alpha_i \mathbf{x}_i^s - \frac{1}{N_{t,l}+N_{t,u}} \Big(\sum_{i=1}^{N_{t,l}} \mathbf{x}_i^{t,l} + \sum_{i=1}^{N_{t,u}} \mathbf{x}_i^{t,u} \Big) \|^2,$$

subject to

$$\begin{aligned} \alpha_i &\geq 0, \\ \sum_{i=1}^{N_s} \alpha_i &= N_s. \end{aligned}$$

Here, $\frac{1}{N_{t,l}+N_{t,u}}\left(\sum_{i=1}^{N_{t,l}} \mathbf{x}_i^{t,l} + \sum_{i=1}^{N_{t,u}} \mathbf{x}_i^{t,u}\right)$ is the mean of the feature vectors representing the target domain instances, and $\frac{1}{N_s}\sum_{i=1}^{N_s} \alpha_i \mathbf{x}_i^s$ is the weighted mean of the features vectors representing the source domain instances. The goal is to find a set of weights α_i that minimize the difference between the two mean feature vectors.

This optimization problem is a quadratic programming problem and can be solved by any standard quadratic program solver.

3.3.2 Setting β

β_i should ideally be set to $\frac{P_t(y_i^s|x_i^s)}{P_s(y_i^s|x_i^s)}$. However, while $P_s(y_i^s|x_i^s)$ can be estimated from the empirical data because we have labeled instances from the source domain, estimation of $P_t(y_i^s|x_i^s)$ is much harder. We thus consider two cases here. If $N_{t,l} = 0$, that is, we are dealing with unsupervised domain adaptation, then we assume that $P_t(y|x) = P_s(y|x)$ for all $(x, y) \in \mathcal{X} \times \mathcal{Y}$. With this assumption, β_i is always set to 1 for all i. Note that this assumption is the best we can do because without any labeled instance from the target domain, we simply do not know anything about the labeling difference unless we have some prior knowledge about the target domain. When we do have a small amount of labeled instances $D_{t,l}$ from the target domain, we propose two strategies to set β_i. In both strategies, we first learn a logistic regression model $P_t(Y|\mathbf{X}, \hat{\theta}_t)$ from $D_{t,l}$, i.e.

$$\hat{\theta}_t = \arg\min_{\theta_t} \left[-\sum_{i=1}^{N_{t,l}} \ln P(y_i^{t,l}|\mathbf{x}_i^{t,l}, \theta_t) + \lambda\|\theta_t\|^2 \right],$$

where

$$P(y|\mathbf{x},\theta_t) = \frac{\exp(\theta_{t,y}^T\mathbf{x})}{\sum_{y'\in\mathcal{Y}}\exp(\theta_{t,y'}^T\mathbf{x})}.$$

We can then predict $P_t(y_i^s|x_i^s)$ using this trained model:

$$\begin{aligned} P_t(y_i^s|x_i^s) &\approx P(y_i^s|\mathbf{x}_i^s,\hat{\theta}_t) \\ &= \frac{\exp(\hat{\theta}_{t,y_i^s}^T\mathbf{x}_i^s)}{\sum_{y'\in\mathcal{Y}}\exp(\hat{\theta}_{t,y'}^T\mathbf{x}_i^s)}. \end{aligned} \tag{3.10}$$

We now describe the two strategies:

- **Direct Estimation:** We can directly use the estimated $P_t(y_i^s|x_i^s)$ as in Equation (3.10) to estimate β_i. In order to do this, we also train another logistic regression model $\hat{\theta}_s$ using the labeled source domain instances from D_s, and estimate $P_s(y_i^s|x_i^s)$ using $P(y_i^s|\mathbf{x}_i^s,\hat{\theta}_s)$. We then set β_i as follows:

$$\begin{aligned} \beta_i' &= \frac{P(y_i^s|\mathbf{x}_i^s,\hat{\theta}_t)}{P(y_i^s|\mathbf{x}_i^s,\hat{\theta}_s)}, \\ C &= \sum_{i=1}^{N_s}\beta_i', \\ \beta_i &= \frac{\beta_i'}{C}. \end{aligned} \tag{3.11}$$

- **Instance Pruning:** The model $P(Y|\mathbf{X},\hat{\theta}_t)$ may not be accurate because the set of labeled instances $D_{t,l}$ is small. We can also choose to only trust the estimated $P(y_i^s|x_i^s;\hat{\theta}_t)$ if it is relatively small. The reason is that when this estimated probability is small, we know that the label for this instance in the source domain cannot be trusted. We thus use the following heuristic to set β_i: if an instance x_i^s is predicted incorrectly by $\hat{\theta}_t$, i.e. $P(y_i^s|\mathbf{x}_i^s,\hat{\theta}_t)$ is very small, then we set β_i to 0, which means that (x_i^s,y_i^s) is essentially removed from the training set; otherwise, β_i is set to 1. In our experiments, we order the misclassified instances in D_s

in increasing order of $P(y_i^s|\mathbf{x}_i^s, \hat{\theta}_t)$, and gradually remove them.

3.3.3 Setting γ

$\gamma_i(y)$ ideally should be set to $P_t(y|x_i^{t,u})$. We clearly do not know this probability. However, we borrow some semi-supervised learning methods, and consider the following two strategies of setting γ.

- **Bootstrapping:** The first method comes from bootstrapping or self-training (Zhu, 2005). Different heuristic version of bootstrapping have been applied to many natural language processing problems, e.g. information extraction (Collins and Singer, 1999; Riloff and Jones, 1999). Here we present a general version of bootstrapping. First, we learn a base classifier from the source domain instances in D_s. If we have some labeled instances from the target domain, i.e. $D_{t,l}$, we can also include these instances in the training data. Using the learned classifier, we can predict the labels for the unlabeled instances in the target domain. We then set $\gamma_i(y)$ to 1 if the instance $x_i^{t,u}$ is predicted to have the label y with high confidence, and to 0 otherwise. Alternative, we can set $\gamma_i(y)$ to 1 for the top m instances that are predicted with the highest confidence. This process can be done iteratively until a certain number of unlabeled target domain instances are added to the training set.

- **Entropy Minimization:** The second method comes from entropy minimization based semi-supervised learning (Grandvalet and Bengio, 2005), which has also been applied to natural language processing problems (Jiao et al., 2006). In this method, we can incorporate the estimation of $\gamma_i(y)$ into the optimization problem itself. First, we assume that we use logistic regression classifiers. We then rewrite Equation (3.7) as follows:

$$\begin{aligned}\hat{\mathbf{w}}_t &= \arg\min_{\mathbf{w}} \Bigg[-\lambda_s \sum_{i=1}^{N_s} \alpha_i \beta_i \ln P(y_i^s|\mathbf{x}_i^s, \mathbf{w}) - \lambda_{t,l} \sum_{i=1}^{N_{t,l}} \ln P(y_i^{t,l}|\mathbf{x}_i^{t,l}, \mathbf{w}) \\ &\quad -\lambda_{t,u} \sum_{i=1}^{N_{t,u}} \sum_{y \in \mathcal{Y}} P(y|\mathbf{x}_i^{t,u}, \mathbf{w}) \ln P(y|\mathbf{x}_i^{t,u}, \mathbf{w}) + \lambda \|\mathbf{w}\|^2 \Bigg], \end{aligned} \tag{3.12}$$

where

$$P(y|\mathbf{x}, \mathbf{w}) \;=\; \frac{\mathbf{w}_y^T \mathbf{x}}{\sum_{y' \in \mathcal{Y}} \mathbf{w}_{y'}^T \mathbf{x}}.$$

We can see that $\gamma_i(y)$ is replaced by $P(y|\mathbf{x}_i^{t,u}, \mathbf{w})$, which is a function of $\mathbf{w}$. Therefore, in this case, $\gamma_i(y)$ is not set in advance before we solve the optimization problem, but rather is inside the optimization function.

Because the objective function in Equation (3.12) is no longer convex, gradient descent only gives us a local optimal solution. We use the following strategy. We start with a small value of $\lambda_{t,u}$, and gradually increase it until it reaches the pre-specified value. At each iteration, with a fixed value of $\lambda_{t,u}$, we apply gradient descent to find $\hat{\mathbf{w}}$, where the starting point of $\mathbf{w}$ is from the solution at the last iteration.

3.3.4 Setting λ's

The three parameters λ_s, $\lambda_{t,l}$ and $\lambda_{t,u}$ are used to control the contribution of using each of the three data sets in the objective function. Usually, if we apply standard supervised learning, then we always set λ_s to 1, $\lambda_{t,l}$ to 1, and $\lambda_{t,u}$ to 0, which means that labeled instances from the source domains are treated the same as those from the target domains, and unlabeled instances are not used. If we apply some kind of standard semi-supervised learning, then $\lambda_{t,u}$ is set to a positive value. For example, in standard bootstrapping, $\lambda_{t,u}$ is set to 1 so that an unlabeled instance with its predicted label is added to the training set and weighted the same as a correctly labeled instance.

For domain adaptation, we may consider different settings of the λ parameters. For example, if the source domain is very different from the target domain, we may want to increase $\lambda_{t,l}$ and/or $\lambda_{t,u}$, or decrease λ_s, or even set λ_s to 0. The relative settings of these parameters are thus related to the degree with which we believe the two domains are similar. In general, because the source domain is different from the target domain and therefore contains noise with respect to the target domain, we believe that the target domain instances are more reliable than the source domain

instances. Therefore, we propose to set $\lambda_{t,l}$ and $\lambda_{t,u}$ to some values greater than 1.

3.4 Experiments

In this section, we present the experiment results from exploring the various ways of setting the instance weighting parameters in the framework. For each task, there are two settings: unsupervised domain adaptation and supervised domain adaptation.

3.4.1 Tasks and Data Sets

We choose three different tasks in natural language processing to evaluate the instance weighting framework for domain adaptation.

- **Part-of-speech tagging:** The first task is part-of-speech (POS) tagging. The goal is to assign part-of-speech tags to each word in a corpus. We use 6166 Wall Street Journal sentences from sections 00 and 01 of the Penn Treebank corpus as the labeled source domain data, and 2730 PubMed sentences from the Oncology section of the PennBioIE corpus as the unlabeled target domain data[1]. These instances are used for the unsupervised setting. For supervised domain adaptation, we use an additional 300 Oncology sentences as the labeled target domain data. We use standard features including the word itself, its surrounding words, its prefix and suffix, capitalization, etc.

- **Entity type classification:** The second task is entity type classification. The setup is very similar to that in (Daumé III and Marcu, 2006). We assume that the entity boundaries have been correctly identified, and we want to classify the types of the entities. We use the ACE 2005 training data for this task[2]. For the source domain, we use 10,000 entities from the newswire collection as labeled data. For the target domains, we use 4,800 entities from the conversational telephone speech (CTS) collection and 5,000 entities from the Web log (WL)

[1] http://bioie.ldc.upenn.edu/publications/latest_release/data/
[2] http://www.nist.gov/speech/tests/ace/ace05/

collection. We therefore have two target domains. For unsupervised domain adaptation, all 4,800 CTS entities and 5,000 WL entities are treated as unlabeled instances. For supervised domain adaptation, we randomly pick 1,000 entities in each target domain as labeled instances, and the remaining instances (3,800 for CTS and 4,000 for WL) as unlabeled. We again use standard features including words, part-of-speech tags, prefix and suffix, capitalization, etc.

- **Spam filtering:** The third task is personalized spam filtering. We use the ECML/PKDD 2006 discovery challenge task A data set[3]. The source domain contains 4,000 spam and ham emails from publicly available sources, and the target domains are three individual users' inboxes, each containing 2,500 emails. For unsupervised domain adaptation, we use all 2,500 personal emails for each user as unlabeled data. For supervised domain adaptation, we randomly pick 500 personal emails for each user as labeled data, and the remaining 2,000 emails as unlabeled data.

The three tasks are referred to as *POS*, *NE Type* and *Spam*, respectively. In all our experiments, we use classification accuracy (the percentage of correctly classified test instances) as our performance measure.

3.4.2 The Effect of α

In this set of experiments, we test the effect of using α only to weigh the source domain instances. We therefore consider the setting of *unsupervised* domain adaptation.

Table 3.1 compares the performance of instance weighting using α only and that of not using instance weighting. *BL* refers to the baseline method, in which all labeled source domain instances in D_s are weighted equally in training. *LR* refers to the method of estimating α using logistic regression, and *MM* refers to the estimation of α using mean matching. For the task of entity type classification, we use two different sizes of D_s: 10,000 and 5,000. Note that for part-of-speech

[3]http://www.ecmlpkdd2006.org/challenge.html

Task	POS	NE Type				Spam		
Src Domain	WSJ	Newswire (10K)		Newswire (5K)		Public Email Collection		
Trgt Domain	Oncology	CTS	WL	CTS	WL	User-1	User-2	User-3
BL	0.8613	0.7819	0.7274	0.7463	0.6832	0.6480	0.6929	0.8028
LR	0.8622	0.8438	0.7218	0.7927	0.6850	0.6580	0.6960	0.8108
MM	–	–	–	0.7238	0.6862	0.6932	0.7260	0.8400

Table 3.1: The effect of using α to weigh the source domain instances in unsupervised domain adaptation.

tagging and for entity type classification when $N_s = 10,000$, we do not show the performance of MM. It is because when N_s is too big, we find that the quadratic program solver we use cannot handle the problem. In all the experiments, we set $\lambda_s = 1$, $\lambda_{t,u} = 0$, and $\lambda = 1$.

From Table 3.1, we see that first, in most cases except for 10K newswire to Web logs with logistic regression and for 5K newswire to conversational telephone speech with mean matching, instance weighting with α gives better performance than the baseline method. The improvement is mostly significant for newswire to conversational telephone speech with logistic regression and for spam filtering with mean matching. This shows that instance weighting using α only does help transform the labeled source domain instances to make them more useful for the target domain.

Between the two estimation methods for α, we can see that for the spam filtering task, mean matching performs better than logistic regression, but for entity type classification, adaptation to conversational telephone speech suggests the opposite. An advantage of logistic regression over mean matching is that it is more efficient to compute.

3.4.3 The Effect of β

In this set of experiments, we use only β to weigh the source domain instances. To obtain estimation of β, we need some labeled target domain instances. We therefore consider the *supervised* domain adaptation setting in this section.

We first run the following experiments. We choose $N_{t,l}$ labeled target domain instances so that $N_{t,l}$ is roughly $\frac{1}{20}$ of N_s. This setting gives us $N_{t,l} \approx 8,000$ for part-of-speech tagging, $N_{t,l} = 500$ for entity type classification, and $N_{t,l} = 200$ for spam filtering. With these labeled target domain

Task	POS	NE Type		Spam		
Src Domain	WSJ	Newswire (10K)		Public Email Collection		
Trgt Domain	Oncology	CTS	WL	User-1	User-2	User-3
BL	0.8613	0.7787	0.7330	0.6435	0.6925	0.8005
IW-α	0.8627	0.8403	0.7310	0.6845	0.7255	0.8410
IW-β	0.8740	0.8732	0.7330	0.7245	0.7808	0.8610

Table 3.2: Comparison between instance weighting using α only and using β only. In all experiments, $N_{t,l}$ is roughly $\frac{1}{20}$ of N_s.

instances, we can estimate β for each source domain instance using Equation (3.11). We then use these β to weigh the labeled source domain instances, and use only these weighted source domain instances to train a classifier for the target domain. In another word, we use $D_{t,l}$ only to help estimate β, but we set $\lambda_{t,l}$ to 0 so that we do not include $D_{t,l}$ in the training data. The reason we exclude $D_{t,l}$ in the training set is that we want to evaluate the pure effect of instance weighting on the source domain using β, and compare it with no instance weighting as well as instance weighting using only α.

Table 3.2 shows the comparison between the three methods: *BL* is the method where no instance weighting is performed, and only D_s is used for training the classifier; *IW-*α is the method where instance weighting using α is used, and again only the weighted D_s is used for training the classifier; *IW-*β is the method where instance weighting using β is used, and still, only the weighted D_s is used for training. For IW-α, for part-of-speech tagging and entity type classification, we use logistic regression method to set α, and for spam filtering, we use mean matching to set α. Note that the performance of BL and of IW-α in Table 3.2 is different from that in Table 3.1. The reason is that we use different $D_{t,u}$ in supervised domain adaptation and in unsupervised domain adaptation, as we explained in Section 3.4.1.

As we can see from the table, instance weighting using β outperforms the baseline method in all cases except for adaptation to Web logs; for adaptation to Web logs, IW-β gives the same accuracy as BL, and better than IW-α. The comparison shows that using estimated $\frac{P_t(y|x)}{P_s(y|x)}$ to weigh the labeled source domain instances can indeed help transform the source domain distribution to make it closer to that of the target domain. Furthermore, we can see from the table that instance

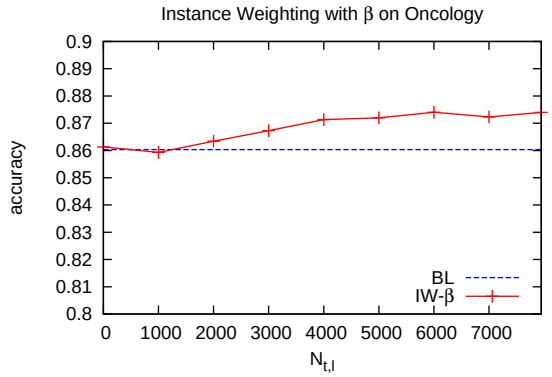

Figure 3.2: The effect of instance weighting using β with different $N_{t,l}$ for part-of-speech tagging.

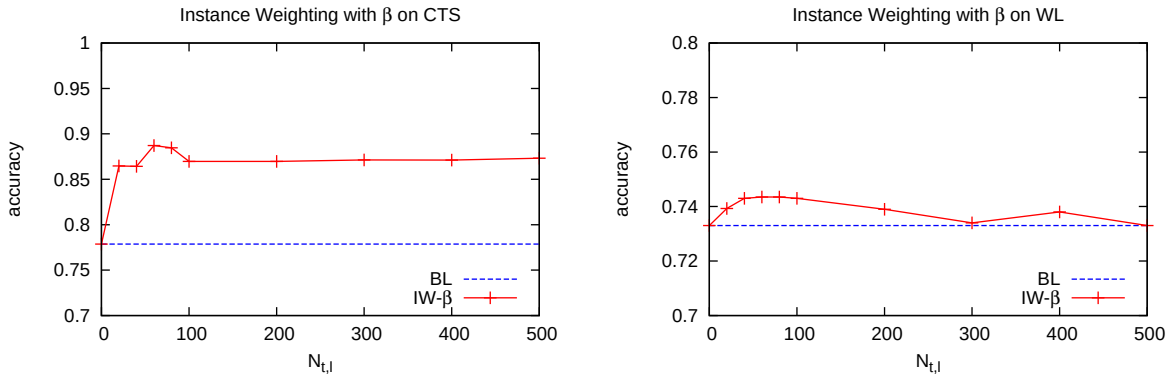

Figure 3.3: The effect of instance weighting using β with different $N_{t,l}$ for entity type classification.

weighting using β is more effective than instance weighting using α. This comparison suggests that for the three NLP tasks we consider here, the domain difference that causes the classification performance to degrade comes more from the difference in $P(Y|X)$ than from $P(X)$.

Because estimation of β depends on the availability of $D_{t,l}$, we expect that the size of $D_{t,l}$, i.e. $N_{t,l}$, has some effect on the estimation accuracy of β and in turn on the instance weighting performance. In order to see how $N_{t,l}$ affects instance weighting, we run another set of experiments. We try different values of $N_{t,l}$, and evaluate the performance of instance weighting using β where β is estimated using these $N_{t,l}$ labeled target domain instances. Again, these labeled target domain instances are not used in the final training of classifiers, but only used for estimation of β, in order for us to see the pure effect of instance weighting using β.

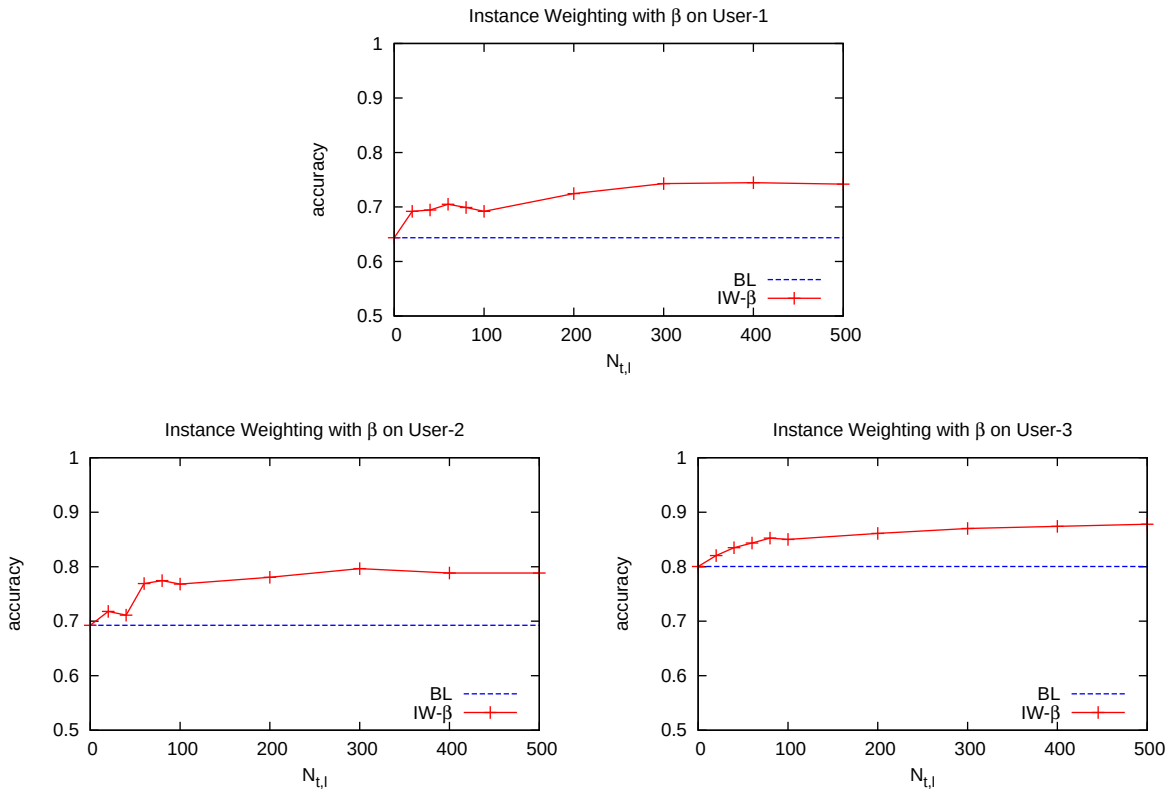

Figure 3.4: The effect of instance weighting using β with different $N_{t,l}$ for spam filtering.

The experiment results are shown in Figure 3.2, Figure 3.3 and Figure 3.4. As we can see from the figures, in general, larger $N_{t,l}$ gives better instance weighting performance, confirming that better estimation of β is obtained with larger $N_{t,l}$. We can also see that the instance weighting performance becomes stable as $N_{t,l}$ is larger than some threshold, and this threshold is usually smaller than $\frac{1}{20}$ of N_s. For example, for the spam filtering task, when $N_{t,l}$ is above roughly 1% of N_s, the instance weighting performance becomes relatively stable. This suggests that we can obtain good instance weighting performance with a relatively small number of labeled target domain instances.

In the experiments above, we use $D_{t,l}$ only to estimation β. We still set $\lambda_{t,l}$ to 0 in order to separate the effect from instance weighting on the source domain using β and the effect from inclusion of labeled target domain instances. In practice, for supervised domain adaptation, i.e. when we have some labeled target domain instances, we naturally want to include them in the

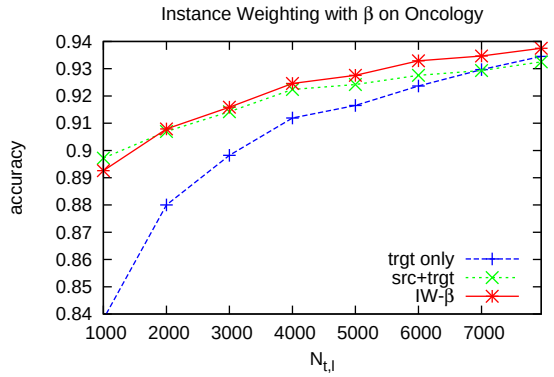

Figure 3.5: The effect of instance weighting using β and inclusion of $D_{t,l}$ in the training set for part-of-speech tagging.

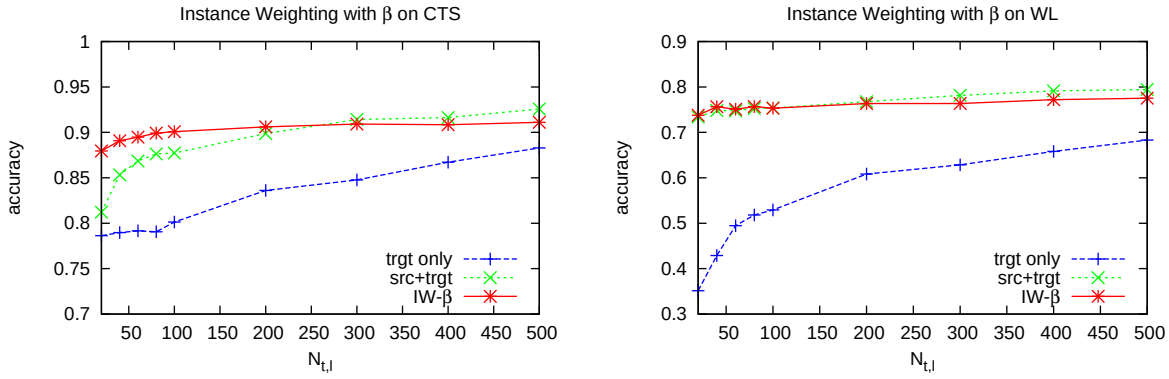

Figure 3.6: The effect of instance weighting using β and inclusion of $D_{t,l}$ in the training set for entity type classification.

training set. We therefore also make the following comparison. First of all, we consider two baseline methods. In the first baseline method, we use only $D_{t,l}$ to train a classifier, i.e. we set $\lambda_{t,l} = 1$ and $\lambda_s = 0$. We call this method *trgt only*. In the second baseline method, we use both D_s and $D_{t,l}$ to train a classifier, i.e. we set $\lambda_s = 1$ and $\lambda_{t,l} = 1$. But we do not perform instance weighting on D_s in this second baseline. We call this method *src+trgt*. We then compare these two baselines with a third method where we use $D_{t,l}$ to estimate β, weigh the source domain instances with β, and combine the weighted D_s with $D_{t,l}$ to train a classifier. Again, we try a range of different $N_{t,l}$ for the comparison of these three methods. Figure 3.5, Figure 3.6 and Figure 3.7 show the comparison.

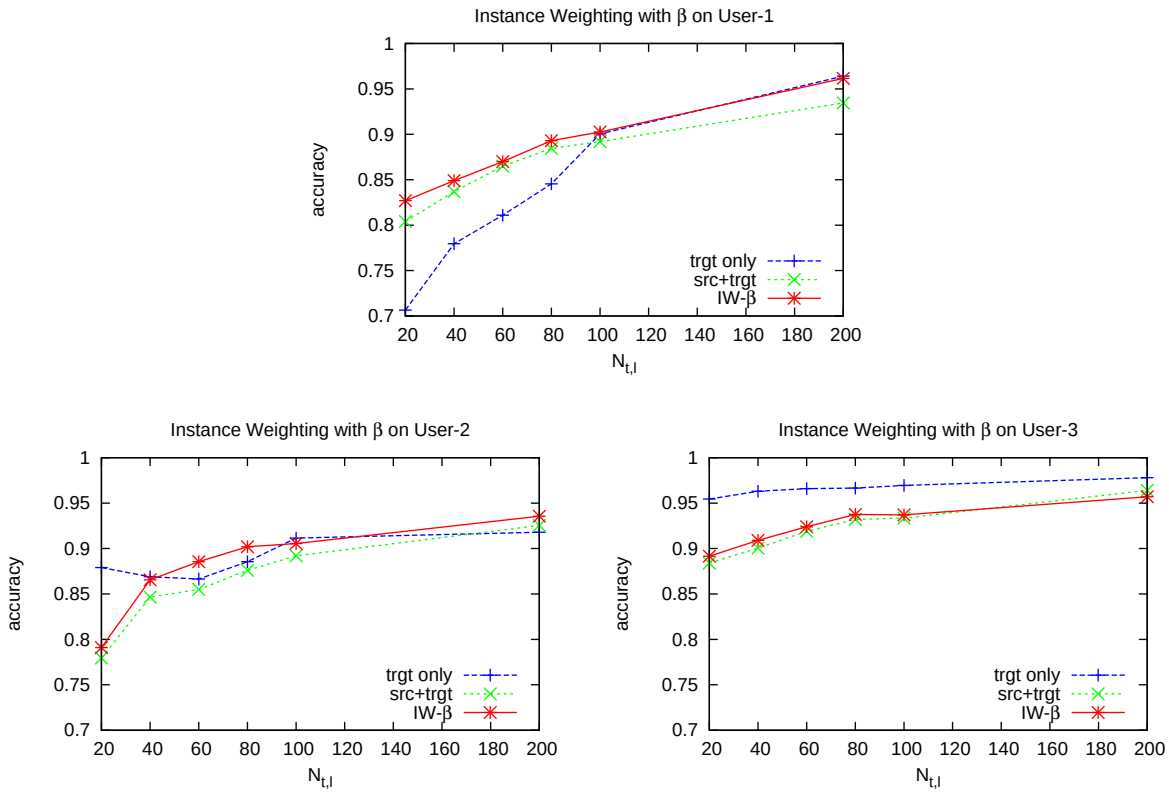

Figure 3.7: The effect of instance weighting using β and inclusion of $D_{t,l}$ in the training set for spam filtering.

For the task of part-of-speech tagging and entity type classification, we see that *src+trgt* clearly outperforms *trgt only*, which means that for these two tasks, the source domain instances are very useful for the target domain. Comparing instance weighting using β with *src+trgt*, we see that for the part-of-speech tagging task, when $N_{t,l}$ is above 4,000, instance weighting performs better than *src+trgt*. For adaptation from newswire to conversational telephone speech, when $N_{t,l}$ is below 200, instance weighting also ourperforms *src+trgt*. In the other ranges of $N_{t,l}$, instance weighting may perform slightly worse than *src+trgt*, but the difference is small. For adaptation to Web logs, instance weighting using β does not show obvious advantage.

For the spam filtering task, the observation is different. First of all, there is no absolute advantage of *src+trgt* over *trgt only*. For User-1, only when $N_{t,l}$ is below 100 is *src+trgt* better than *trgt only*. For the other two users, *src+trgt* does not show much advantage. For User-3, there

is clear advantage of *trgt only* over *src+trgt*. This observation can be explained as follows. The learning curves of *trgt only* show that spam filtering for these three users is an easier task than part-of-speech tagging and entity type classification shown above, in the sense that good performance can be achieved with a relatively small number of training instances. This is especially true for User-3, as we can see that 20 labeled instances from the target domain can already give an accuracy of above 95%. Therefore, when labeled target domain instances are available, the usefulness of labeled source domain instances becomes small, and the noise in the source domain instances may become more obvious, causing *trgt only* to be better than *src+trgt*. In this case, even if we try to weigh the source domain instances, the effect is not obvious because it is overshadowed by the effect of the inclusion of the labeled target domain instances. Nevertheless, if we compare instance weighting using β and *src+trgt*, we see that for User-1 and User-2, IW-β still outperforms *src+trgt*, and for User-3, IW-β does not hurt *src+trgt*.

The above experiment results suggest that it is generally safe to apply instance weighting using β on the labeled source domain instances before combining them with the labeled target domain instances. We can generally expect the performance to be better than simply combining D_s and $D_{t,l}$.

3.4.4 Instance Pruning

As we discussed in Section 3.3.2, besides directly estimating β by estimating $P_t(y|x)$ and $P_s(y|x)$, we can also prune the instances from the source domain. We first train a logistic regression classifier using $D_{t,l}$, and then classify the source domain instances in D_s using this classifier. For those instances whose predicted labels differ from their true labels, we consider them to be "misleading," and remove them from the training data set. We can gradually remove these "misleading" instances in decreasing order of the prediction confidence. In another word, we first remove instances that are classified by the classifier trained from $D_{t,l}$ with the highest confidence but are classified *incorrectly*.

Figure 3.8, Figure 3.9 and Figure 3.10 show the performance on the three tasks as we gradu-

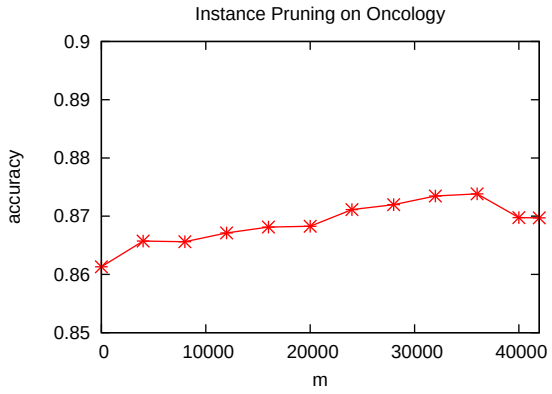

Figure 3.8: The effect of instance pruning for part-of-speech tagging.

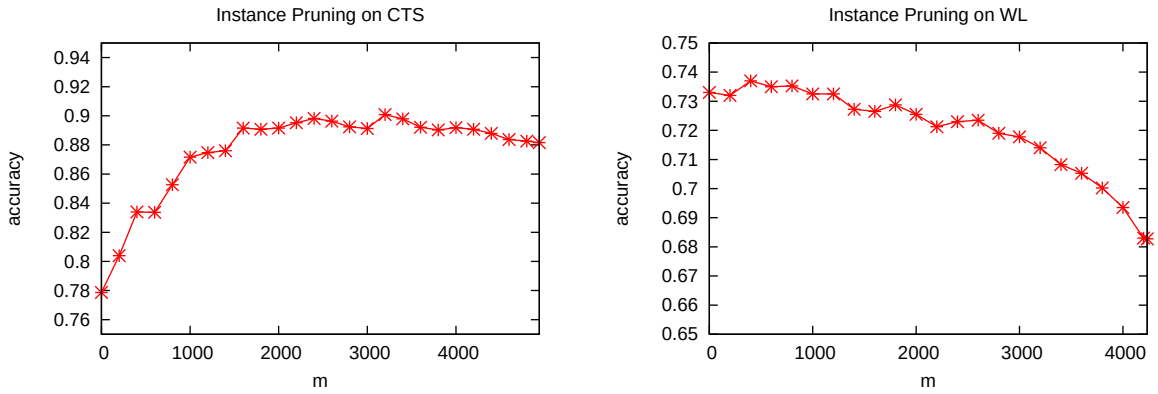

Figure 3.9: The effect of instance pruning for entity type classification.

ally remove the "misleading" source domain instances until all of them are removed. The x-axis shows the number of "misleading" instances removed. In this set of experiments, in order to later compare with direct estimation of β, we use the same set of $D_{t,l}$ for each task as in Table 3.2. As we can see, in all cases except for adaptation to Web logs, removing those "misleading" instances improves the performance from the baseline method where all labeled source domain instances are used. In some cases removing *all* "misleading" instances is not the optimal setting. For example, for adaptation from newswire to conversational telephone speech, removing around 3,000 "misleading" instances is better than removing all of them. Nevertheless, we can generally choose to remove *all* "misleading" instances and still obtain considerably better performance than the baseline without instance pruning.

For adaptation from newswire to Web logs, we see that instance pruning greatly hurts the

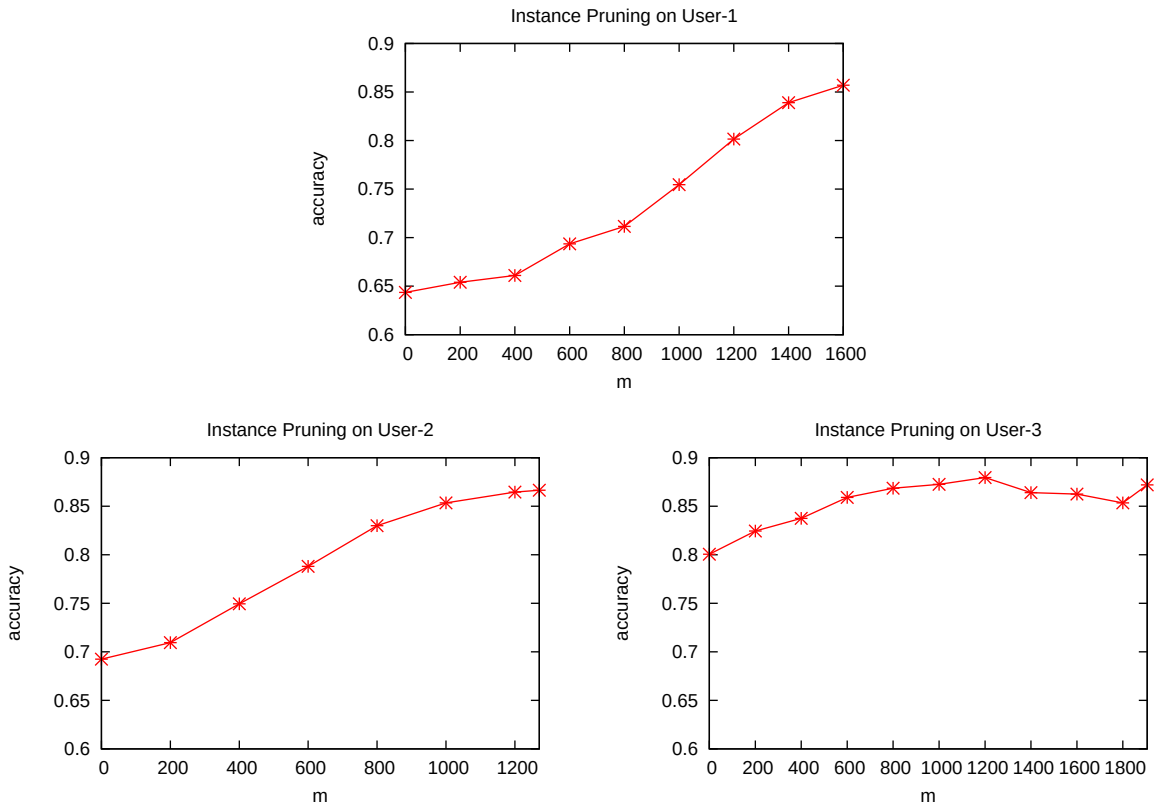

Figure 3.10: The effect of instance pruning for spam filtering.

performance. Actually previous experiment results also showed that instance weighting using α and using β did not improve the baseline performance for adaptation to Web logs from newswire. All these results suggest that the Web log domain is not very different from the newswire domain. With a small set of $D_{t,l}$ from the Web log domain, the "misleading" source domain instances we identify may not be correct, and therefore removing them may even hurt the performance.

To compare instance pruning with direct estimation of β, we list the performance on each task when *all* "misleading" source domain instances are removed, and compare the performance with the baseline performance as well as the performance obtained by weighting the source domain instances using directly estimated β. The performance is shown in Table 3.3. Note that for *IW-*β and *instance pruning*, the same set of $D_{t,l}$ is used for each source-target domain pair. We can see that for adaptation from newswire to conversational telephone speech and for spam filtering, instance pruning outperforms instance weighting using directly estimated β. However, for part-of-

Task	POS	NE Type		Spam		
Src Domain	WSJ	Newswire (10K)		Public Email Collection		
Trgt Domain	Oncology	CTS	WL	User-1	User-2	User-3
BL	0.8613	0.7787	0.7330	0.6435	0.6925	0.8005
IW-β	0.8740	0.8732	0.7330	0.7245	0.7808	0.8610
instance pruning	0.8697	0.8816	0.6828	0.8570	0.8665	0.8720

Table 3.3: Comparison between directly estimating β and instance pruning.

Task	POS	NE Type		Spam		
Src Domain	WSJ	Newswire (10K)		Public Email Collection		
Trgt Domain	Oncology	CTS	WL	User-1	User-2	User-3
BL	0.8613	0.7787	0.7330	0.6435	0.6925	0.8005
IW-α	0.8622	0.8403	0.7310	0.6845	0.7255	0.8410
IW-β	0.8740	0.8732	0.7330	0.7245	0.7808	0.8610
IW-$\alpha\beta$	0.8722	0.8911	0.7395	0.7265	0.7660	0.8480

Table 3.4: Comparison between instance weighting using α only, using β only, and using α and β together. In all experiments, $N_{t,l}$ is roughly $\frac{1}{20}$ of N_s.

speech tagging and for adaptation from newswire to Web logs, instance pruning is less effective or even harmful. In these two cases, instance weighting using β does not generate much improvement over the baseline in the first place, which suggests that the source domain does not contain much "noise" with respect to the target domain. In this case, instance pruning may cause useful instances in the source domain to be pruned.

Overall, we can conclude that instance pruning is a more aggressive way of removing "noise" from the source domain. If the source domain does contain much "noise," then instance pruning is more effective than directly estimating β; otherwise, directly estimating β is a more conservative and safer choice.

3.4.5 Combining α and β

In previous sections we saw that using α only and β only to weigh the source domain instances are both useful, with β being more effective. In this section, we use both α and β to weigh the source domain instances as shown in Equation (3.7).

Table 3.4 shows the comparison between the performance achieved by using α only, using β

Task	POS	NE Type				Spam		
Src Domain	WSJ	Newswire (10K)		Newswire (5K)		Public Email Collection		
Trgt Domain	Oncology	CTS	WL	CTS	WL	User-1	User-2	User-3
BL	0.8613	0.7819	0.7274	0.7463	0.6832	0.6480	0.6929	0.8028
BST	0.8731	0.8910	0.7542	0.8883	0.7116	0.8860	0.9268	0.9780
ENT	0.8735	0.8419	0.7386	0.8277	0.7004	0.6396	0.7640	0.9756

Table 3.5: The effect of using γ to include $D_{t,u}$ in unsupervised domain adaptation.

only and using both. The results show that using both α and β is not necessarily better than using only β; its performance is often between using α only and using β only. Therefore, in the case of supervised domain adaptation, we should just apply instance weighting using β.

3.4.6 The Effect of γ

In all the experiments above, we always set $\lambda_{t,u} = 0$, that is, we exclude the use of unlabeled target domain instances in $D_{t,u}$. In this set of experiments, we evaluate the effect of including these unlabeled target domain instances into the training set. We consider the setting of unsupervised domain adaptation, i.e. we do not use any labeled target domain instance. We use the two methods of setting γ as presented in Section 3.3.3. For bootstrapping, at each iteration, we add 10 unlabeled target domain instances that are predicted with the highest confidence into the training set. We set $\lambda_{t,u}$ to 1. We continue the iterations until all instances from $D_{t,u}$ are added to the training set. For entropy minimization, we start with $\lambda_{t,u} = 10^{-5}$, and gradually increase it to 1. We do not perform any instance weighting on D_s.

Table 3.5 shows the performance of bootstrapping (BST) and entropy minimization (ENT) on the three tasks. We also include the baseline (BL) performance where $D_{t,u}$ is not used. We can draw very clear conclusions from the results. First, both bootstrapping and entropy minimization improve the performance over the baseline method. It therefore shows that inclusion of target domain instances is useful even if pseudo labeled instances are used. Second, bootstrapping clearly outperforms entropy minimization for all tasks and for all source-target domain pairs. This comparison suggests that for many NLP problems, bootstrapping is a better choice than entropy

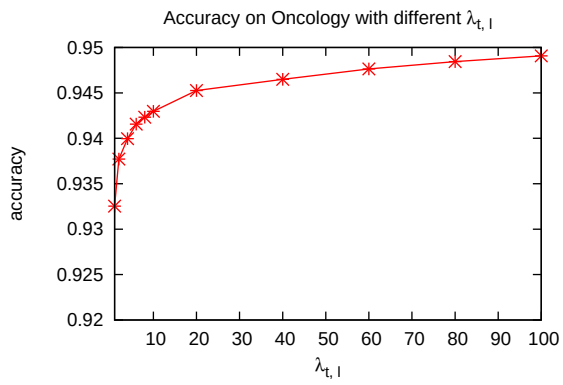

Figure 3.11: The effect of increasing $\lambda_{t,l}$ on part-of-speech tagging.

minimization if we want to apply semi-supervised learning.

3.4.7 The Effect of $\lambda_{t,l}$

As we mentioned earlier, in standard supervised learning, λ_s and $\lambda_{t,l}$ are both set to 1, that is, we ignore the difference between the source and the target domains. For domain adaptation, however, we expect the labeled target domain instances to be more useful than the labeled source domain instances. We therefore want to increase the value of $\lambda_{t,l}$. In this section, we test the effect of increasing $\lambda_{t,l}$ when $D_{t,l}$ is available. For part-of-speech tagging, we include all the 300 labeled target domain sentences, which contain around 8,000 tokens, into $D_{t,l}$. For entity type classification and spam filtering, we include only 50 labeled target domain instances in $D_{t,l}$. We do not make use of the unlabeled target domain instances so we set $\lambda_{t,u}$ to 0. We do not perform any instance weighting on D_s.

Figure 3.11, Figure 3.12 and Figure 3.13 show the performance on the three tasks as λ_s is set to 1 and $\lambda_{t,l}$ changes from 1 to 100 (for part-of-speech tagging), 200 (for entity type classification), or 80 (for spam filtering). The message from these figures is very clear. Increasing $\lambda_{t,l}$ to some value greater than 1 improves the performance on the target domain. In all cases except for WL, up to the point when $\lambda_{t,l} = \frac{N_s}{N_{t,l}}$, the performance always increases as $\lambda_{t,l}$ increases. For adaptation to WL, however, there is some fluctuation of the performance in the range of $\lambda_{t,l}$ between 1 and

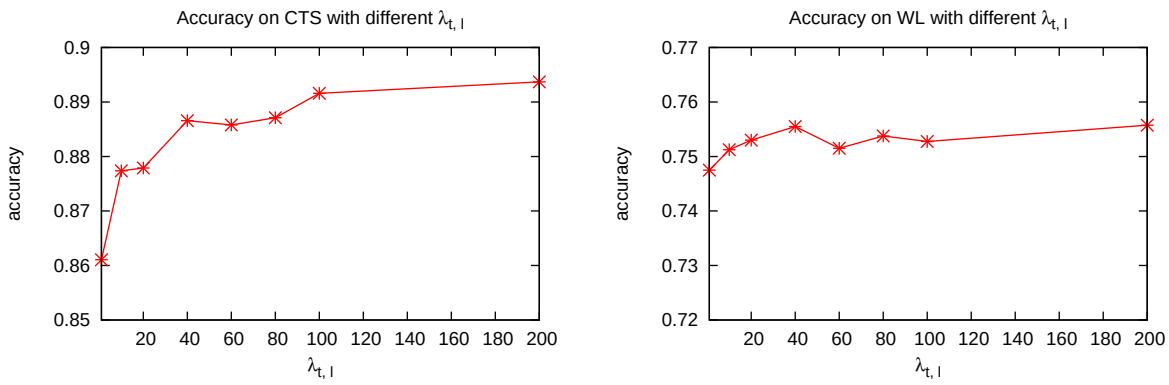

Figure 3.12: The effect of increasing $\lambda_{t,l}$ on entity type classification.

Task	POS	NE Type		Spam		
Src Domain	WSJ	Newswire (10K)		Public Email Collection		
Trgt Domain	Oncology	CTS	WL	User-1	User-2	User-3
$\lambda_{t,l} = 1$	0.9325	0.8611	0.7475	0.8550	0.8530	0.9070
$\lambda_{t,l} = \frac{N_s}{N_{t,l}}$	0.9453	0.8937	0.7558	0.8935	0.9190	0.9435

Table 3.6: The effect of setting $\lambda_{t,l}$ heuristically to $\frac{N_s}{N_{t,l}}$.

$\frac{N_s}{N_{t,l}}$ (i.e. 200). However, setting $\lambda_{t,l}$ to $\frac{N_s}{N_{t,l}}$ still outperforms setting it to 1. The fluctuation shows that the optimal value for $\lambda_{t,l}$ in this case (adaptation from newswire to Web logs) is smaller than those for the other cases (e.g. adaptation from newswire to conversational telephone speech). This suggests that the difference between the newswire domain and the Web log domain is relatively smaller than the difference in other domain pairs, and therefore we need less emphasis on the target domain instances here. This hypothesis can be confirmed by previous experiment results.

To select a good value for $\lambda_{t,l}$, we can heuristically set $\lambda_{t,l}$ to $\frac{N_s}{N_{t,l}}$ while setting λ_s to 1. Table 3.6 compares the performance when we use this heuristic setting of $\lambda_{t,l}$ and that of setting $\lambda_{t,l}$ to 1, which is the default setting in standard supervised learning. We can see that in all cases setting $\lambda_{t,l}$ to $\frac{N_s}{N_{t,l}}$ improves over the default setting. Therefore, we can recommend this heuristic as a default setting for $\lambda_{t,l}$.

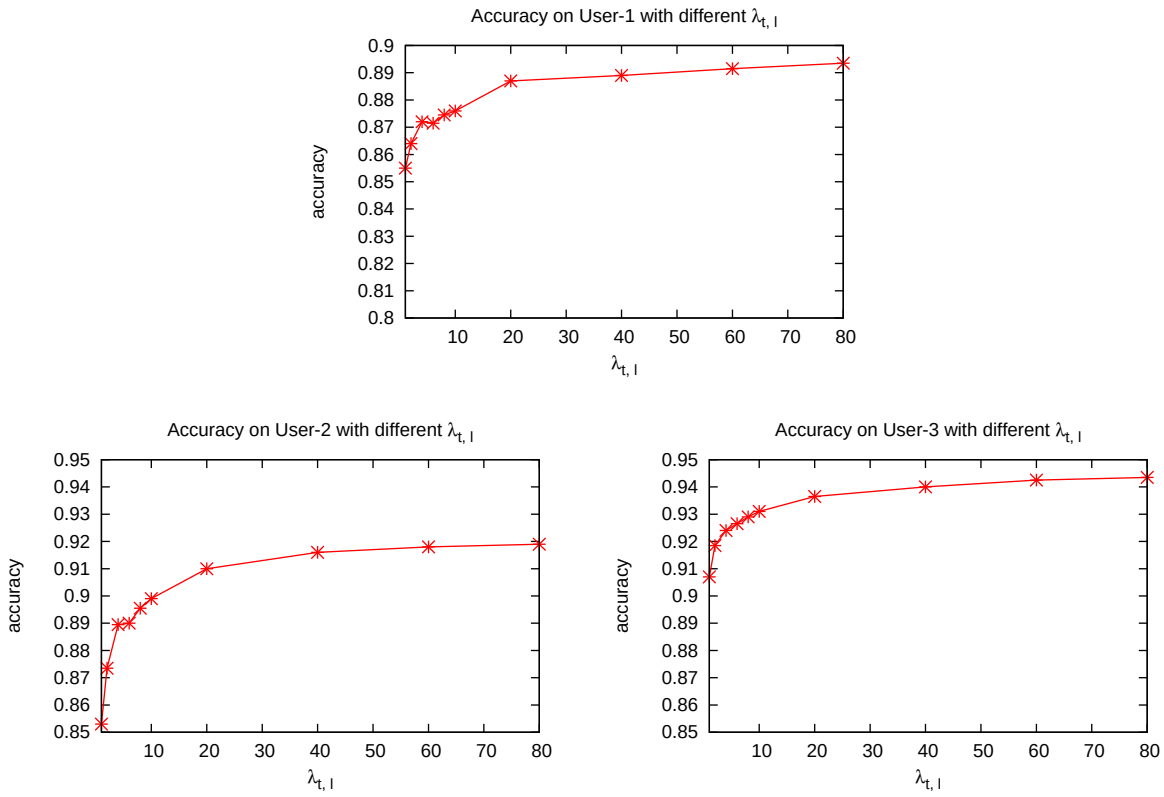

Figure 3.13: The effect of increasing $\lambda_{t,l}$ on spam filtering.

3.4.8 The Effect of $\lambda_{t,u}$

Similar to $\lambda_{t,l}$, $\lambda_{t,u}$ can also be set to a value greater than 1. In this set of experiments, we use bootstrapping as the semi-supervised learning method to set γ, and gradually increase the value of $\lambda_{t,u}$. In bootstrapping, we stop when all instances in $D_{t,u}$ are added to the training data set. The performance on the three tasks is shown in Figure 3.14, Figure 3.15 and Figure 3.16.

Comparing the results shown in these figures with those for increasing $\lambda_{t,l}$, we can clearly see that increasing $\lambda_{t,u}$ is not as effective and can often be dangerous, such as the case for adaptation to Web logs in entity type classification and adaptation to User-2's email collection in spam filtering. However, if we think about where $\lambda_{t,u}$ is applied, the results we obtained are not surprising. $\lambda_{t,u}$ is the overall weighting factor for those instances from $D_{t,u}$. Since we stop bootstrapping when all instances from $D_{t,u}$ are added to the training set, the term $\lambda_{t,u} \sum_{i=1}^{N_{t,u}} \sum_{y \in \mathcal{Y}} \gamma_i(y) L(x_i^{t,u}, y, f)$ already has a lot of contribution to the overall objective function even when $\lambda_{t,u}$ is set to 1. The

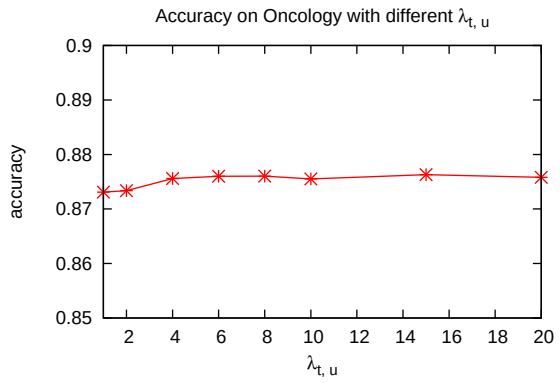

Figure 3.14: The effect of increasing $\lambda_{t,u}$ on part-of-speech tagging.

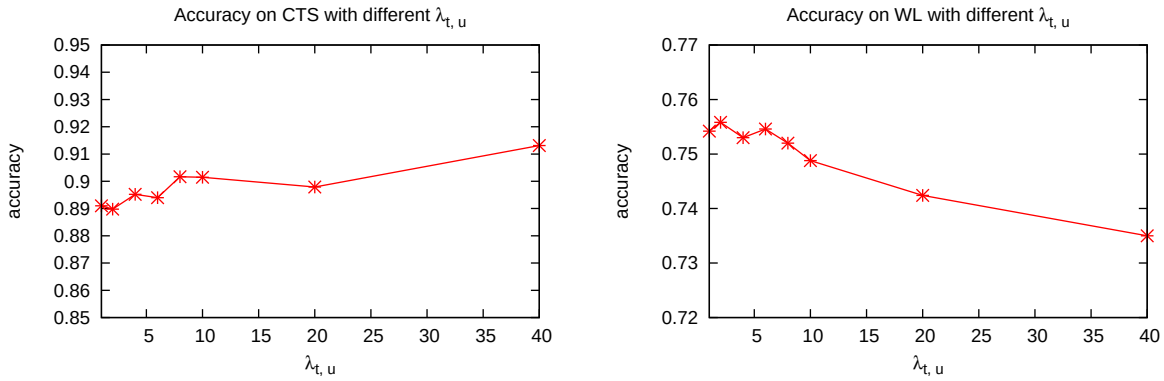

Figure 3.15: The effect of increasing $\lambda_{t,u}$ on entity type classification.

effect of increasing $\lambda_{t,u}$ depends on how noisy the pseudo-labeled $D_{t,u}$ are. Note that D_s is also noisy for the target domain because instances in D_s are drawn from the source domain. If the pseudo-labeled $D_{t,u}$ is indeed a better representation of the target domain than D_s, then increasing $\lambda_{t,u}$ may improve the performance of the classifier on the target domain; if, however, that the pseudo-labeled $D_{t,u}$ contains much noise, then it is better not to increase $\lambda_{t,u}$. We therefore recommend to keep $\lambda_{t,u}$ to be 1.

3.5 Summary and Discussions

In this chapter, we presented an instance weighting framework for domain adaptation. The instance weighting approach is naturally derived from the analysis of the distributional difference

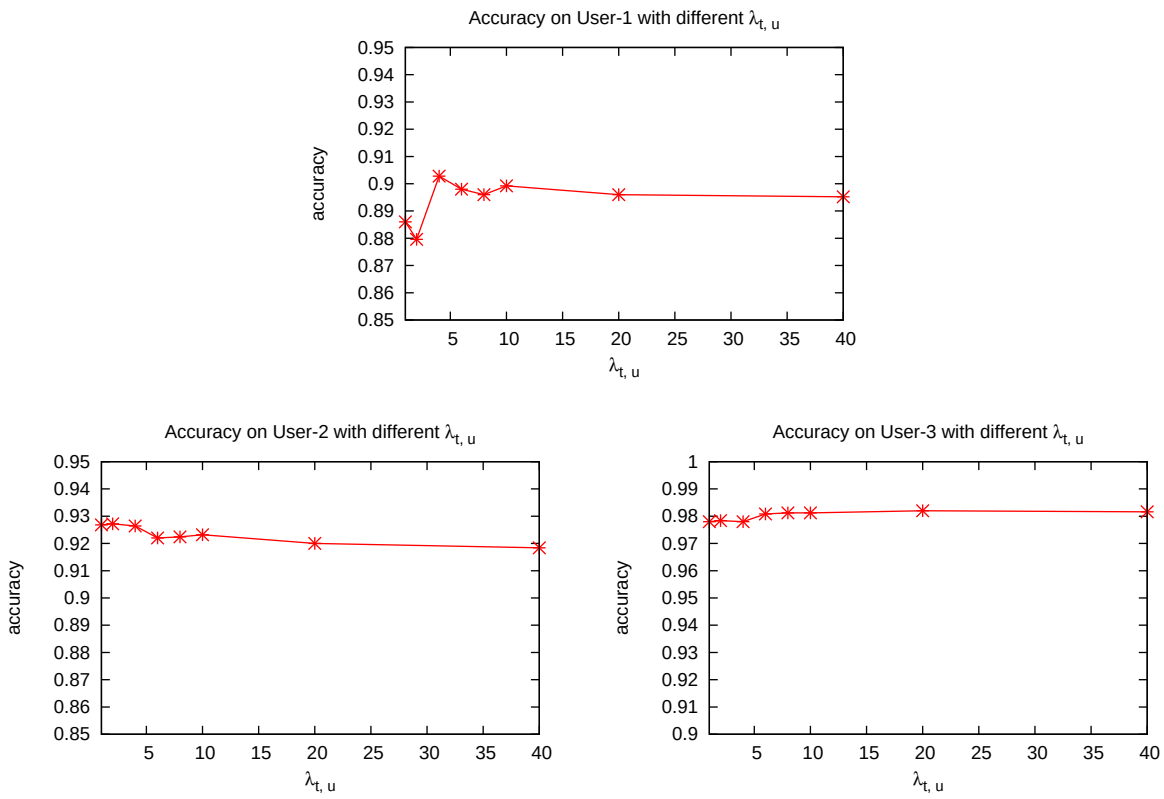

Figure 3.16: The effect of increasing $\lambda_{t,u}$ on spam filtering.

between the source and the target domains. While the original idea of instance weighting is well-justified and looks straightforward, to implement the idea, we have to introduce several weighting parameters in the overall framework, and estimation of these parameters is not trivial.

We presented alternative ways of estimating each of the weighting parameters we introduced into the framework, and evaluated the effect of these estimation methods on three natural language processing tasks with real data sets. The experiment results suggest that the following instance weighting strategies are effective for domain adaptation:

First, for unsupervised domain adaptation, we can apply two techniques: (1) Instance weighting using $\alpha \propto \frac{P_t(x)}{P_s(x)}$. To estimate α, when the number of source domain instances is relatively small, we can use either the logistic regression method or the mean matching method as shown in Section 3.3.1. When the number of source domain instances is relatively large, mean matching may be too expensive to compute, and we should apply the logistic regression method. (2)

Bootstrapping using the unlabeled target domain instances.

Second, for supervised domain adaptation, three techniques can be effective: (1) Instance weighting using $\beta \propto \frac{P_t(y|x)}{P_s(y|x)}$. $P_t(y|x)$ and $P_s(y|x)$ can be estimated by training two logistic regression classifiers on the target and the source domains, respectively, using the available labeled instances in each domain. (2) Balancing the contributions of the labeled target domain instances and the labeled source domain instances by setting $\lambda_{t,l}$ to $\frac{N_s}{N_{t,l}}$. (3) Bootstrapping using the unlabeled target domain instances.

Currently, the above strategies are tested separately and each is shown to be generally effective. We also conducted some experiments combining these strategies. Our results show that combining these techniques by using the default settings of the instance weighting parameters as suggested above does not always give better classification performance than using the individual techniques. An explanation is that the default parameter settings we found are based on experiments using individual techniques. An important future work is therefore to *globally* optimize the settings of the various instance weighting parameters in order to effectively combine the various instance weighting techniques.

Chapter 4

A Feature Selection Framework for Domain Adaptation

In Chapter 2, we briefly introduced the idea of feature selection for domain adaptation. In this chapter, we further explain the motivation for feature selection in detail, and formally present our feature selection framework for domain adaptation. The framework we present in this chapter works for a general setting where there are $K(K > 1)$ different source domains, each containing a set of labeled instances. There may or may not be some labeled instances from the target domain, but there are always a set of unlabeled instances from the target domain. Again, as in the instance weighting framework, the unlabeled target domain instances can be used for training the classifier through semi-supervised learning.

In this chapter, we assume that instances are represented by high-dimensional binary feature vectors, and we consider only linear classifiers to separate different classes. With this assumption, the absolute value of the weight of a feature in a learned classifier roughly represents the degree of correlation between the feature and class labels: a feature that is highly correlated with some class label will have a relatively large weight in the learned classifier. For many natural language processing problems such as part-of-speech tagging and named entity recognition, binary feature representations and linear classifiers are commonly used and have achieved the state-of-the-art performance. Therefore, our assumptions do not impose much restriction on the applicability of our framework to natural language processing problems.

Our key idea is to differentiate between features that are useful for all domains and features that are only useful for a specific domain. We hypothesize that one reason why the classifier trained on the source domains does not work well on the target domain is that the classifier puts too much weight on features that are only useful for the source domains but not useful for the target

domains, and hence causes the general features not to get enough weight. We therefore propose to first identify a set of features that are generalizable across the different source domains, and then put more emphasis on these generalizable features during training in order to learn a more general classifier that can be better applied to the target domain.

The target domain may also have its own domain-specific features, that is, features that are useful in the target domain but not useful in the source domains. We cannot expect to obtain large weights for these features by training only on the source domain instances because these features do not show high correlation with class labels in the source domains. We therefore propose to apply semi-supervised learning in order to obtain appropriate weights for these features.

Our general feature selection framework works when there are multiple source domains. In the case when there is only a single source domain, we require some labeled target domain instances in order to identify the generalizable features. These labeled target domain instances can also help us learn target-domain-specific features.

We evaluate the feature selection framework on the task of gene name recognition where different organisms are regarded as different domains. We also evaluate the framework with a single source domain on the same natural language processing tasks as in Chapter 3. The experiment results show that the framework is effective and outperforms standard supervised and semi-supervised learning methods.

The rest of this chapter is organized as follows. In Section 4.1, we introduce the notions of *generalizable* features and *domain-specific* features, and argue that identifying and emphasizing generalizable features is a solution to avoid *domain overfitting*. We then present a formal two-stage feature selection framework in Section 4.2. A key issue in this framework is how to identify the generalizable features. We present two ways to identify generalizable features in Section 4.3, and discuss some of the implementation issues in Section 4.4. In Section 4.5, we show how the framework should be specialized to the case when there is only a single source domain. We present our experiment results in Section 4.6 and Section 4.7, and summarize the results in Section 4.8.

4.1 Generalizable Features and Domain-Specific Features

Why would a classifier learned from a source domain overfit the source domain and not work well for the target domain? An important observation we make is that there may be some features that are very useful for the classification task in the source domain but are no longer useful in the target domain. Let us consider an example from the gene name recognition task. The task is essentially a named entity recognition problem, where the named entities are gene and protein names in biomedical text. Suppose our source domain contains biomedical articles about the organism *fly*, and the target domain contains biomedical articles about the organism *mouse*. For fly, there are a number of gene names that end with the suffix "-less", such as *eyeless*, *daughterless* and *wingless*, because fly biologists tend to name a gene by its phenotype. Therefore, the feature "-less" has a relatively large absolute weight value in the classifier learned from fly articles. However, for many other organisms including mouse, genes are not named after their phenotypes. Now the feature "-less" is no longer useful for these other organisms or domains. On the other hand, there are features that are useful across different domains. For example, the contextual feature "X is expressed" is generally useful for gene name recognition for different organisms because gene expression is commonly discussed in biomedical articles. We refer to these features that are useful across different domains as *generalizable* features and features that are only useful for a specific domain as *domain-specific* features.

If in the source domains, the generalizable features are weaker than the domain-specific features, then the generalizable features may not get large weights, because we usually control the complexity of classifiers during the training process by placing a regularization term $\lambda\|\mathbf{w}\|^2$ on the weight vector, which penalizes large weights. In another word, generalizable features have to compete with domain-specific features for the weight mass. Ideally, if we want to learn a model from the source domains that is also useful in the target domain, we want the weight mass to be assigned to those generalizable features rather than those domain-specific features. This kind of *skewed* regularization can be achieved by imposing a larger λ to the weights of those domain-

specific features. In Section 4.2.1, we will show how this skewed regularization is imposed in the objective function.

4.2 A Two-Stage Framework with Feature Selection for Domain Adaptation

Having discussed the distinction between generalizable features and domain-specific features, we now propose a two-stage approach to domain adaptation. First, we learn a general classifier from the source domains that emphasizes the generalizable features and therefore presumably works better on the target domain than a classifier learned without distinguishing generalizable features from domain-specific features. We call this step *generalization*. Second, after we have the general classifier, we make use of unlabeled target domain instances to pick up features that are specifically useful for the target domain. We call this step *adaptation*.

4.2.1 Generalization

The first stage of our two-stage approach to domain adaptation is a domain generalization stage. We have shown that in order to make the model learned from the source domains useful in the target domain, we need to regularize the learning process such that the weight mass is mostly kept on the generalizable features. There are two problems that need to be solved here: (1) To identify the generalizable features; (2) To learn appropriate weights for these generalizable features. We first show how the second problem can be solved given a fixed set of generalizable features. We postpone the solutions to the first problem until Section 4.3.

To easily represent the separation of the generalizable features from other features in our formal problem formulation, we first introduce a matrix A, which we refer to as the *feature selection matrix*. Formally, A is an $h \times p$ matrix ($h < p$) that transforms an instance $\mathbf{x}$ represented as a p-dimensional vector in the original feature space into an h-dimensional vector $\mathbf{z} = A\mathbf{x}$ in the

reduced feature space with only generalizable features. In another word, A is a matrix in which each entry is either 0 or 1, and $AA^T = I_{h\times h}$. For example, the following 2×6 matrix A chooses the second and the fifth rows in a 6-dimensional feature vector to form a new 2-dimensional feature vector.

$$A = \begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \end{pmatrix}.$$

In the rest of this chapter, the constraints that each entry of A is 0 or 1 and $AA^T = I_{h\times h}$ are implied whenever we refer to A. Such a matrix A essentially defines the selection of h features from the original feature set. Here we assume that the number of generalizable features h is fixed.

Suppose we already know the set of h generalizable features, that is, we have a fixed A. How do we learn the appropriate weights for these generalizable features from the labeled instances in the K source domains? In standard supervised learning where we do not consider the difference between the domains, we learn a single weight vector $\mathbf{w}$ for all domains[1]. When we have K different source domains for training, however, we cannot expect the optimal classifiers, i.e. the optimal weight vectors, to be the same for all domains. We thus introduce K different weight vectors, $\{\mathbf{w}^k\}_{k=1}^K$, for the K source domains. However, since the generalizable features behave similarly in different domains, we expect the weights for them to be similar across domains. To capture this, we decompose each $\mathbf{w}^k$ as follows:

$$\mathbf{w}^k = A^T\mathbf{v} + \mathbf{u}^k,$$

where $\mathbf{v}$ is an h-dimensional weight vector shared by all domains, and $\mathbf{u}^k$ is a domain-specific p-dimensional weight vector. We can think of $\mathbf{v}$ as essentially the weight vector for the generalizable features.

[1]To simplify the discussion, we consider binary classification problems here so we have a single weight vector to represent a classifier. For multiclass classification, usually a classifier has $|\mathcal{Y}|$ weight vectors, one for each class. Our discussion based on binary classification can be easily generalized to multiclass classification.

Given the labeled instances from the K source domains, and given a fixed A, we can learn weight vectors $\mathbf{v}$ and $\{\mathbf{u}^k\}_{k=1}^K$ by optimizing the following objective function:

$$\begin{aligned}\left(\hat{\mathbf{v}}(A), \{\hat{\mathbf{u}}^k(A)\}\right) &= \underset{\mathbf{v},\{\mathbf{u}^k\}}{\arg\min} \left[\frac{1}{K}\sum_{k=1}^{K}\frac{1}{N_k}\sum_{i=1}^{N_k} L(\mathbf{x}_i^k, y_i^k, \mathbf{w}^k)\right. \\ &\quad \left. +\lambda\left(\|\mathbf{v}\|^2 + \mu_s\sum_{k=1}^{K}\|\mathbf{u}^k\|^2\right)\right], \end{aligned} \tag{4.1}$$

where $\mathbf{w}^k = A^T\mathbf{v} + \mathbf{u}^k$, $\{(x_i^k, y_i^k)\}_{i=1}^{N_k}$ is a set of labeled instances from the k'th source domain, and $L(\mathbf{x}, y, \mathbf{w})$ is a loss function. The first term on the right-hand side is the empirical loss, and the second term is the regularization term, where different regularization parameters are put on $\mathbf{v}$ and $\{\mathbf{u}^k\}$. Note that the learned $\hat{\mathbf{v}}$ and $\{\hat{\mathbf{u}}^k\}$ are dependent on A, and therefore we represent them as functions of A.

Now recall that we want to put more weight mass on the generalizable features. To achieve this goal, we set μ_s to be much larger than 1. With $\mu_s \gg 1$, we penalize large values of $\{\mathbf{u}^k\}$ more than large values of $\mathbf{v}$, and therefore, we naturally give the most weight mass to $\mathbf{v}$ unless the training instances strongly favor some large values of $\{\mathbf{u}^k\}$.

Note that in Equation (4.1) we have made two modifications to the standard objective function in supervised learning: (a) We have separated a subset of generalizable features (defined by A) so that we now have two sets of weights (i.e. $\mathbf{v}$ and $\mathbf{u}^k$) for each domain. (b) We tie the weight vector $\mathbf{v}$ for the generalizable features across all the domains, while allowing some slight domain variation captured by $\{\mathbf{u}^k\}$.

A key problem we need to solve is how to find a good A in the first place, that is, how to find a good set of generalizable features. We postpone this discussion until Section 4.3.

4.2.2 Adaptation

After the first stage of domain generalization, we will obtain a matrix A, which represents the set of generalizable features learned from the source domains, as well as a weight vector for these

generalizable features. In the second stage of domain adaptation, our goal is to pick up those features that are specifically useful for the target domain, but cannot be learned from the source domains. A sensible way to achieve this goal is to include some labeled instances from the target domain in the learning process. In the case where we do not have any labeled instance in the target domain, we can adopt semi-supervised learning to obtain some target domain instances with predicted labels and use these *pseudo-labeled* target domain instances in the learning process. Here we adopt bootstrapping. More specifically, with the best generalizable model that we have learned from the source domains, we make predictions on the unlabeled instances in the target domain, choose the most confident m instances together with their pseudo labels, and include these labeled instances in our objective function to learn the weights.

Formally, let A be the feature selection matrix that we have obtained in the first domain generalization stage, and let $\{\mathbf{x}_i^t, \hat{y}_i^t\}_{i=1}^m$ be the set of instances in the target domain that have been predicted with the highest confidence values, where $\hat{y}_i^t$ is the predicted label of $\mathbf{x}_i^t$. We learn weight vectors $\hat{\mathbf{v}}$ and $\hat{\mathbf{u}}^t$ by optimizing the following objective function:

$$\begin{aligned}\left(\hat{\mathbf{v}}, \hat{\mathbf{u}}^t, \{\hat{\mathbf{u}}^k\}\right) &= \underset{\mathbf{v},\mathbf{u}^t,\{\mathbf{u}^k\}}{\arg\min} \left[\frac{1}{K+1} \left(\sum_{k=1}^{K} \frac{1}{N_k} \sum_{i=1}^{N_k} L(\mathbf{x}_i^k, y_i^k, \mathbf{w}^k) + \frac{1}{m} \sum_{i=1}^{m} L(\mathbf{x}_i^t, \hat{y}_i^t, \mathbf{w}^t) \right) \right. \\ &\quad \left. + \lambda \left(\|\mathbf{v}\|^2 + \mu_s \sum_{i=1}^{K} \|\mathbf{u}^k\|^2 + \mu_t \|\mathbf{u}^t\|^2 \right) \right]. \end{aligned} \tag{4.2}$$

Note that Equation (4.2) is very similar to Equation (4.1): the target domain is treated in the same way as all source domains in the objective function, except for the regularization parameter μ_t, that is, the domain-specific weight vector $\mathbf{u}^t$ for the target domain has a different regularization parameter than the domain-specific weight vectors $\mathbf{u}^k$ for the source domains. The intuition is that while we do not want to put much weight mass on the features specifically useful for the source domains, we do want to allocate some weight mass to the features specifically useful for the target domain. Therefore, we want to set $\mu_t \ll \mu_s$ to achieve this goal. After we have learned $\hat{\mathbf{v}}$ and $\hat{\mathbf{u}}^t$, we set $\hat{\mathbf{w}}^t = A^T\hat{\mathbf{v}} + \hat{\mathbf{u}}^t$, and use this weight vector as the final linear classifier to make predictions

on the target domain.

4.3 Finding Generalizable Features

A remaining issue from the discussion above is how to identify the best h generalizable features. In this section, we show how ideally generalizable features should be defined based on the expected performance on the target domain. For any A, let $\hat{\mathbf{v}}(A)$ be the weight vector learned from the training domains, as defined in Equation (4.1). Suppose we only use these generalizable features to make predictions on the target domain. Recall that our ultimate goal is to make good predictions on the target domain. One way to quantify this goal is to minimize the expected loss with respect to $P_t(\mathbf{X}, Y)$ in the target domain, which can be captured by the following objective function:

$$A^* = \arg\min_A \sum_{(\mathbf{x},y)\in\mathcal{X}\times\mathcal{Y}} P_t(\mathbf{x}, y) L(\mathbf{z}, y, \hat{\mathbf{v}}(A)), \tag{4.3}$$

where $\mathbf{z} = A\mathbf{x}$, and $L(\mathbf{z}, y, \mathbf{v})$ is a loss function.

What Equation (4.3) means is that we want to choose the optimal A^* among all choices of A's so that the classifier learned from the source domains with strong emphasis on the features selected by A^* gives the lowest expected loss in the target domain. While Equation (4.3) is the ideal criterion for choosing the optimal A^*, in practice, it is infeasible to achieve the goal because (1) we do not know $P_t(\mathbf{X}, Y)$, and (2) a brute force enumeration of all possible values of A is too expensive. To make the objective function feasible to compute, our general idea is to obtain an approximation of A^*. Here we discuss two strategies of approximating A^*.

Joint Optimization of A and $\mathbf{v}$

In Equation (4.3), instead of using $P_t(\mathbf{x}, y)$, which is unknown, we can use the empirical joint probability of $\mathbf{x}$ and y from the source domains to approximate $P_t(\mathbf{x}, y)$. If we make several

further approximations, we obtain the following objective function:

$$\left(\hat{A}, \hat{\mathbf{v}}, \{\hat{\mathbf{u}}^k\}\right) = \underset{A,\mathbf{v},\{\mathbf{u}^k\}}{\arg\min} \left[\frac{1}{K} \sum_{k=1}^{K} \frac{1}{N_k} \sum_{i=1}^{N_k} L(\mathbf{x}_i^k, y_i^k, \mathbf{w}^k) + \lambda \left(\|\mathbf{v}\|^2 + \mu_s \sum_{k=1}^{K} \|\mathbf{u}^k\|^2 \right) \right], \tag{4.4}$$

where $\mathbf{w}^k = A^T\mathbf{v} + \mathbf{u}^k$. The $\hat{A}$ and $\hat{\mathbf{v}}$ chosen in this way form the final generalizable classifier we use for predictions on the target domain. Note that Equation (4.4) is the same as Equation (4.1) except that A is now free to change inside the objective function. In Section 4.4.1, we will explain how we can solve this optimization problem efficiently without enumerating all the possible values of A.

Note that in this approximation, we use the empirical risk on the training data to assess the quality of A, that is, we train and validate on the same data. However, to avoid overfitting, in general we want to use different data for training and validation. In the next section, we show such a method that uses cross validation. The experiment results in Section 4.6 also show that the cross validation method is better than the joint optimization method.

Domain Cross Validation

A better way to approximate A^* is a *domain cross validation* method. Here we borrow the idea of leave-one-out cross validation from regular supervised learning. However, we treat each training domain as if it were a single training instance. Thus, given a fixed A, we first learn the weight matrix $\hat{\mathbf{v}}(A)$ using all but one source domains, and then test the performance on the held-out source domain. We repeat this procedure for each held-out source domain, and take the average performance as an indicator of how good A is.

Formally, we want to find $\hat{A}$ that optimizes the following objective function:

$$\hat{A} = \underset{A}{\arg\min} \frac{1}{K} \sum_{k=1}^{K} \left[\frac{1}{N_k} \sum_{i=1}^{N_k} L(\mathbf{z}_i^k, y_i^k, \hat{\mathbf{v}}(k, A)) \right], \tag{4.5}$$

where $\mathbf{z}_i^k = A\mathbf{x}_i^k$, and $\hat{\mathbf{v}}(k, A)$ is the optimal weight matrix for the features selected by A, learned from all source domains except the k'th source domain. In another word, $\hat{\mathbf{v}}(k, A)$ is obtained from

$$\left(\hat{\mathbf{v}}(k, A), \{\hat{\mathbf{u}}^{k'}(k, A)\}_{k' \neq k}\right) = \underset{\mathbf{v}, \{\mathbf{u}^{k'}\}_{k' \neq k}}{\arg\min} \left[\frac{1}{K-1} \sum_{k' \neq k} \frac{1}{N_{k'}} \sum_{i=1}^{N_{k'}} L(\mathbf{x}_i^{k'}, y_i^{k'}, \mathbf{w}^{k'}) + \lambda \left(\|\mathbf{v}\|^2 + \mu_s \sum_{k' \neq k} \|\mathbf{u}^{k'}\|^2 \right) \right], \tag{4.6}$$

where $\mathbf{w}^{k'} = A^T\mathbf{v} + \mathbf{u}^{k'}$.

However, even with this approximation, in practice it is still infeasible to enumerate all possible A when optimizing Equation (4.5). In Section 4.4.2, we will propose a heuristic way to approximate the optimization problem in Equation (4.5).

4.4 Implementation Details

In this section, we discuss some implementation details that have been left out in Section 4.3. Since now we touch on the actual implementation of the framework, we need to choose a specific classification algorithm, or in another word, a specific loss function. We choose to use logistic regression classifiers.

4.4.1 An Alternating Optimization Procedure for Joint Optimization of A and $\mathbf{v}$

We first discuss how to solve the optimization problem in Equation (4.4). The problem formulation is very similar to the optimization problem in (Ando and Zhang, 2005), which can be solved by an alternating optimization procedure (Bezdek and Hathaway, 2002). We thus also use alternating optimization to solve our problem. We give the outline of the procedure below.

Recall that $\mathbf{w}^k = A^T\mathbf{v} + \mathbf{u}^k$ for each k. Equation (4.4) can then be rewritten as follows:

$$\begin{aligned}\left(\hat{A}, \hat{\mathbf{v}}, \{\hat{\mathbf{w}}^k\}\right) &= \underset{A,\mathbf{v},\{\mathbf{w}^k\}}{\arg\min} \left[\frac{1}{K}\sum_{k=1}^{K}\frac{1}{N_k}\sum_{i=1}^{N_k} L(\mathbf{x}_i^k, y_i^k, \mathbf{w}^k)\right. \\ &\quad \left. +\lambda\left(\|\mathbf{v}\|^2 + \mu_s\sum_{k=1}^{K}\|\mathbf{w}^k - A^T\mathbf{v}\|^2\right)\right]. \end{aligned} \tag{4.7}$$

When $\{\mathbf{w}^k\}_{k=1}^K$ are fixed, the first term on the right-hand side in Equation (4.7) is also fixed, while we can vary A and $\mathbf{v}$ to minimize the second term. When A is fixed, we can vary $\mathbf{v}$ and $\{\mathbf{u}^k\}_{k=1}^K$ to minimize the objective function, which is equivalent to varying $\mathbf{v}$ and $\{\mathbf{w}^k\}_{k=1}^K$. Thus, we alternate between fixing $\{\mathbf{w}^k\}_{k=1}^K$ and fixing A to solve for $(\hat{A}, \hat{\mathbf{v}}, \{\hat{\mathbf{w}}^k\})$ that minimizes the objective function:

1. Initialize $\{\mathbf{w}^k\}_{k=1}^K$: for each k, set $\mathbf{w}^k$ to the weight vector trained from the labeled instances from the k'th domain.

2. Fix $\{\mathbf{w}^k\}$, solve for $\hat{A}$ and $\hat{\mathbf{v}}$:

$$\left(\hat{A}, \hat{\mathbf{v}}\right) = \underset{A,\mathbf{v}}{\arg\min}\left[\|\mathbf{v}\|^2 + \mu_s\sum_{k=1}^{K}\|\mathbf{w}^k - A^T\mathbf{v}\|^2\right].$$

3. Fix A, solve for $\hat{\mathbf{v}}$ and $\{\hat{\mathbf{u}}^k\}$:

$$\left(\hat{\mathbf{v}}, \{\hat{\mathbf{u}}^k\}\right) = \underset{\mathbf{v},\{\mathbf{u}^k\}}{\arg\min}\left[\frac{1}{K}\sum_{k=1}^{K}\frac{1}{N_k}\sum_{i=1}^{N_k} L(\mathbf{x}_i^k, y_i^k, A^T\mathbf{v} + \mathbf{u}^k) + \lambda\left(\|\mathbf{v}\|^2 + \mu_s\sum_{k=1}^{K}\|\mathbf{u}^k\|^2\right)\right].$$

4. For all k, set $\mathbf{w}^k = A^T\mathbf{v} + \mathbf{u}^k$.

5. Repeat 2, 3 and 4 until A does not change any more.

In the algorithm outlined above, if we choose the loss function for logistic regression, the optimization problem defined in step 3 is similar to the standard training of logistic regression

classifiers, which can be solved by standard convex optimization methods. The optimization problem defined in step 2 in general can be reduced to an SVD (singular value decomposition) problem if A is an arbitrary matrix satisfying $AA^T = I$. See (Ando and Zhang, 2005) for the derivation of the solution for a similar problem. In our case, since we restrict the entries of A to be either 1 or 0, the problem is further simplified, and the solution can be easily obtained by ranking the features with a scoring function and selecting the best h features. We leave out the technical details here.

Since in both step 2 and 3, the value of the objective function decreases, the alternating optimization procedure converges to a local minimum.

4.4.2 A Heuristic for Domain Cross Validation

We now discuss how to approximate the optimization problem defined in Equation (4.5). First, let us consider a single component of Equation (4.5):

$$\frac{1}{N_k}\sum_{i=1}^{N_k} L(\mathbf{z}_i^k, y_i^k, \hat{\mathbf{v}}(k, A)), \tag{4.8}$$

where $\mathbf{z}_i^k = A^T\mathbf{x}_i^k$. Recall that $\hat{\mathbf{v}}(k, A)$ is the weight vector learned from all training instances except those from the k'th domain, with a fixed A. Consider another weight vector $\hat{\beta}(k, A)$ defined as follows:

$$\left(\hat{\beta}(k, A), \hat{\mathbf{u}}\right) \;=\; \arg\min_{\beta,\mathbf{u}} \left[\frac{1}{N_k}\sum_{i=1}^{N_k} L(\mathbf{x}_i^k, y_i^k, A^T\beta + \mathbf{u}) + \lambda\left(\|\beta\|^2 + \mu_s\|\mathbf{u}\|^2\right)\right]. \tag{4.9}$$

In order to minimize Formula (4.8), we want $\hat{\mathbf{v}}(k, A)$ to be as close to $\hat{\beta}(k, A)$ as possible because $\hat{\beta}(k, A)$ almost directly minimizes Formula (4.8). To measure the similarity between $\hat{\beta}(k, A)$ and $\hat{\mathbf{v}}(k, A)$, we use the inner product of $\hat{\beta}(k, A)$ and $\hat{\mathbf{v}}(k, A)$. Formally, we define a scoring function $S(\beta, \mathbf{v})$ as follows:

$$S(\beta, \mathbf{v}) \;=\; \beta^T\mathbf{v}.$$

We can then use the following optimization problem to approximate Equation (4.5):

$$\hat{A} \;=\; \arg\max_{A} \sum_{k=1}^{K} S\Big(\hat{\beta}(k,A), \hat{\mathbf{v}}(k,A)\Big),$$

where $\hat{\beta}(k,A)$ and $\hat{\mathbf{v}}(k,A)$ are defined in Equation (4.9) and Equation (4.6), respectively.

We still have not addressed the problem that it is not feasible to enumerate all possible A's. We now make a final approximation, which allows us to incrementally select h features rather than enumerating all possible subsets of features of size h. First, let $\hat{\theta}^k$ be the optimal weight vector learned from the training instances in the k'th source domain *without* any feature selection. Formally,

$$\hat{\theta}^k \;=\; \arg\min_{\theta} \left[\frac{1}{N_k} \sum_{i=1}^{N_k} L(\mathbf{x}_i^k, y_i^k, \theta) + \lambda \|\theta\|^2 \right].$$

Let $\hat{\theta}^{\bar{k}}$ be the optimal weight vector learned from the training instances in all source domains except the k'th source domain, also *without* any feature selection:

$$\hat{\theta}^{\bar{k}} \;=\; \arg\min_{\theta} \left[\frac{1}{K-1} \sum_{k' \neq k} \frac{1}{N_{k'}} \sum_{i=1}^{N_{k'}} L(\mathbf{x}_i^{k'}, y_i^{k'}, \theta) + \lambda \|\theta\|^2 \right].$$

We now approximate $\hat{\beta}(k,A)$ and $\hat{\mathbf{v}}(k,A)$ as follows:

$$\hat{\beta}(k,A) \;\approx\; A\,\hat{\theta}^k, \tag{4.10}$$

$$\hat{\mathbf{v}}(k,A) \;\approx\; A\,\hat{\theta}^{\bar{k}}. \tag{4.11}$$

What Equation (4.10) and Equation (4.11) mean is that we can roughly use the weights learned from the training data *without* feature selection to approximate the weights learned *with* feature selection. Note that since we only use this approximation to select features rather than to obtain the weights for the final classifier, this inaccurate approximation should not be critical.

Let $f_1, \ldots, f_h$ be the set of features selected by A. Let θ_f denote the weight for feature f in the weight vector θ. We then have

$$\begin{aligned}\sum_{k=1}^{K} S\Big(\hat{\beta}(k,A), \hat{\mathbf{v}}(k,A)\Big) &\approx \sum_{k=1}^{K} S\Big(A\hat{\theta}^k, A\hat{\theta}^{\bar{k}}\Big) \\ &= \sum_{k=1}^{K} \big(A\hat{\theta}^k\big)^T A\hat{\theta}^{\bar{k}} \\ &= \sum_{k=1}^{K} \Bigg(\sum_{i=1}^{h} \hat{\theta}^k_{f_i} \cdot \hat{\theta}^{\bar{k}}_{f_i} \Bigg). \end{aligned} \tag{4.12}$$

Now it is clear that we can maximize Equation (4.12) by selecting h features that have the highest scores defined below:

$$\sum_{k=1}^{K} \hat{\theta}^k_f \cdot \hat{\theta}^{\bar{k}}_f. \tag{4.13}$$

4.4.3 Summary of the Framework

We summarize the framework and the implementation details in this section. First, in Section 4.2, we have shown that given a fixed set of generalizable features, represented by matrix A, we can first learn a generalizable classifier $\hat{\mathbf{v}}(A)$ using the K source domains by solving the optimization problem in Equation (4.1). With this generalizable classifier, we can select m unlabeled instances from the target domain together their pseudo labels, and include them in the objective function. We can then learn a target-domain-specific classifier by solving the optimization problem in Equation (4.2).

To find a set of h generalizable features, we have shown in Section 4.3 that we can either jointly find A and $\mathbf{v}$ to optimize the objective function, or perform domain cross valuation. We then showed in Section 4.4 the implementation details of the two methods, where for the second method, a heuristic method was proposed to select h generalizable features without enumerating all possible A's.

4.5 Special Case with A Single Source Domain

The framework presented above assumes multiple source domains. In most cases, we may only have a single source domain. Because identification of generalizable features requires labeled instances from more than one domains, in the case with a single source domain, we need some labeled target domain instances in order to identify generalizable features. In this section, we show how the general framework presented above should be specialized for this case. We assume that we have a set of labeled source domain instances D_s and a set of labeled target domain instances $D_{t,l}$.

Equation (4.1) can now be re-written as follows:

$$\begin{aligned}\Big(\hat{\mathbf{v}}(A), \hat{\mathbf{u}}^s(A), \hat{\mathbf{u}}^t(A)\Big) \;=\; & \underset{\mathbf{v},\mathbf{u}^s,\mathbf{u}^t}{\arg\min} \Bigg[\sum_{i=1}^{N_s} L(\mathbf{x}_i^s, y_i^s, A^T\mathbf{v} + \mathbf{u}^s) \\ & + \sum_{j=1}^{N_{t,l}} L(\mathbf{x}_j^{t,l}, y_j^{t,l}, A^T\mathbf{v} + \mathbf{u}^t) \\ & \lambda\Big(\|\mathbf{v}\|^2 + \mu_s\|\mathbf{u}^s\|^2 + \mu_t\|\mathbf{u}^t\|^2 \Big) \Bigg]. \end{aligned} \tag{4.14}$$

To identify the generalizable features, that is, to find an appropriate matrix A, we can use either joint optimization or domain cross validation as discussed in Section 4.3. For domain cross validation, we can treat D_s and $D_{t,l}$ as labeled instances from two source domains, and use Formula (4.13) to find the best h generalizable features.

Note that in Equation (4.14), $D_{t,l}$ is used not only for learning generalizable features but also for learning those target-domain-specific features, i.e. for finding $\hat{\mathbf{u}}^t(A)$. So the generalization stage and the adaptation stage now become a single step in this case. If we incorporate bootstrapping, then the pseudo-labeled target domain instances will be treated the same as those in $D_{t,l}$.

4.6 Experiments with Multiple Source Domains

In this section, we evaluate the two-stage feature selection framework for domain adaptation in the case where there are more than one source domains.

4.6.1 Data Set and Experiment Setup

We test our method on the problem of recognizing gene and protein names from biomedical literature. The data set we use is from the BioCreAtIvE I challenge, Task 1B[2]. The data set contains three subsets, corresponding to three organisms, fly, mouse, and yeast. Thus we can naturally treat each organism as an individual domain. Note that since the labels in this data set were automatically obtained using a dictionary-based named entity recognition method, the data set is noisy and the absolute performance on this data set is lower than the state-of-the-art for gene recognition. Nevertheless, it should not affect the evaluation of our framework compared with standard methods.

The task is cast into a classification problem to predict the boundaries of gene mentions, where each word in the text is classified as either part of a gene name or outside of any gene name. We follow a commonly used Markov model based sequential tagging method to recognition gene names in this way (Finkel et al., 2005). We use F1 as the primary performance measure, where F1 is defined as

$$\text{F1} = \frac{2 \times \text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}.$$

We ran three parallel sets of experiments. In each set of experiments, we use two organisms as the source domains, and the third organism as the target domain. We refer to the three sets of experiments as F+M⇒Y, M+Y⇒F and Y+F⇒M, where F, M and Y denote fly, mouse and yeast, respectively.

[2]http://biocreative.sourceforge.net/biocreative_1_task1b.html

Our feature selection framework consists of two stages. For each stage, we compare our method with a corresponding baseline method. For the generalization stage, we consider a baseline method which combines the two training domains without considering the domain difference. We also consider feature selection in the baseline method, that is, we first rank the features based on some commonly used feature selection criterion, and then select the top h features. In our experiments, we use the χ^2 statistic measure to rank the features. We call this baseline method *BL*. We implemented the first stage of our feature selection framework using both the joint optimization heuristic and the domain cross validation heuristic. We call the first *DA-1* and the second *DA-2*. When comparing DA-1 and DA-2 with BL, we vary the value of h for all three methods.

For the adaptation stage of our domain adaptation method, we consider a baseline method that uses regular bootstrapping. In particular, at the i'th round of bootstrapping, we use the current model to label the sentences in the target domain, and choose $m = 200 \times i$ sentences that are labeled with gene mentions and are predicted with the highest probabilities. We add these m sentences with the predicted labels to the training set, and retrain the model. We call this baseline semi-supervised learning method *BL-SSL*. In comparison, our domain adaptive method also chooses m sentences in each round of bootstrapping in the same way, but uses Equation (4.2) to learn a new model. Since our results in the first stage show that DA-2 is better than DA-1, in the adaptation stage, we only combine DA-2 with bootstrapping. We call this method *DA-2-SSL*.

In all experiments, we set λ to 10^{-6}. This value was chosen based on cross validation on the training data. In DA-1, DA-2 and DA-2-SSL, we set μ_s to 10^6. This value was arbitrarily chosen to be sufficiently large. In DA-2-SSL, we set μ_t to 1. This value was also arbitrarily chosen to be sufficiently smaller than μ_s.

4.6.2 Domain Generalization

In Table 4.1, we compare the performance of BL, DA-1 and DA-2 when h is set to the total number of features (designated as *Max* in the table) and the optimal value (designated as *Opt* in the table). First, we can see that when h is set to the total number of features, DA-1 and DA-2

Method	h	F+M⇒Y			M+Y⇒F			Y+F⇒M		
		Prec	Rec	F1	Prec	Rec	F1	Prec	Rec	F1
BL	Max	0.624	0.449	0.522	0.534	0.0641	0.114	0.586	0.297	0.394
	Opt	0.742	0.553	0.633	0.569	0.0727	0.129	0.607	0.316	0.416
DA-1	Max	0.657	0.523	0.583	0.535	0.0695	0.123	0.597	0.291	0.392
	Opt	0.638	0.616	0.627	0.498	0.0907	0.153	0.583	0.335	0.425
DA-2	Max	0.655	0.524	0.582	0.535	0.0695	0.123	0.598	0.291	0.391
	Opt	0.666	0.643	**0.654**	0.533	0.120	**0.195**	0.565	0.402	**0.470**

Table 4.1: Comparison between BL, DA-1 and DA-2.

either perform similarly to BL, or perform slightly better than BL. This comparison shows that even if we include all features in the generalizable set, considering the domain difference in the training data and optimizing Equation (4.1) to learn the common weights shared by all training domains is still better than ignoring the domain difference. Second, when h is set optimally, the performance achieved by DA-2 is better than that of DA-1 and of BL, and DA-1, in two out of the three cases, is better than BL. This comparison shows the advantage of the domain adaptive method if we can appropriately set h. It also shows that in general the domain cross validation heuristic is better than the joint optimization heuristic.

In Figure 4.1, we show the performance of the three methods when the value of h varies from 1 to the total number of features. As we can see, for F+M⇒Y and Y+F⇒M, BL achieves better performance when a small number of features (100 features) are used, but the performance drops when more features are included. For M+Y⇒F, BL achieves the best performance when 10000 features are used. In all three settings, however, the performance of BL at $h = 100000$ (roughly one tenth of the total number of features) is much lower than the performance when all features are used. This fluctuation of BL suggests that the χ^2 statistic measure computed from the labeled source domain instances is not reliable any more on the target domain. Since in practice, it is hard to predict the optimal value of h, DA-1 and DA-2 are more robust than BL for domain adaptation because their performance is more stable when h is relatively large.

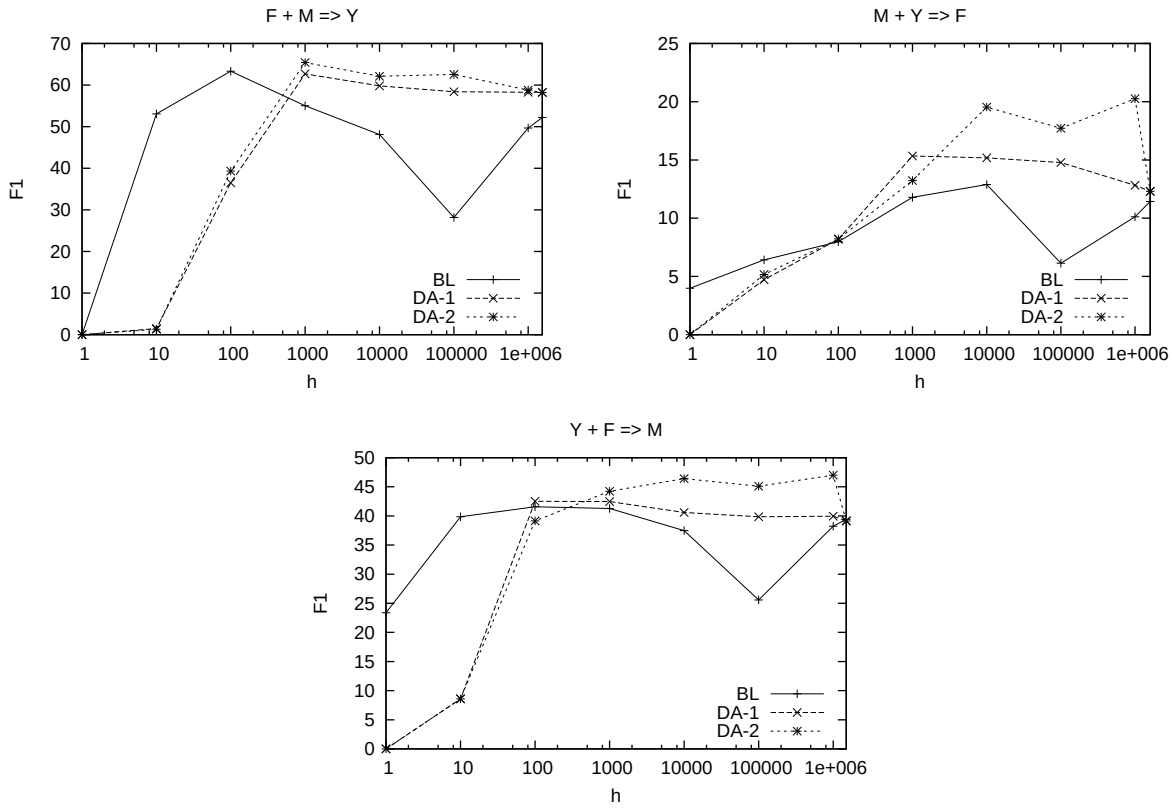

Figure 4.1: Comparison between BL, DA-1 and DA-2 as h Varies.

4.6.3 Domain Adaptation

For the methods BL-SSL and DA-2-SSL, we choose the best models learned by BL and DA-2 as the starting models, respectively. In another word, we assume that the optimal value of h is used. We also consider another baseline, *BL-SSL-2*, where we use *all* features instead of the top h features. The reason we include BL-SSL-2 is that we found that in the case with F+M⇒Y, BL-SSL-2 performed reasonably but BL-SSL performed poorly.

In Table 4.2, we show the performance of the three methods when $m = 1000$ and when m is set optimally (designated as *Opt* in the table), i.e. when we stop bootstrapping at the iteration right before the performance decreases. We can see that when $m = 1000$, DA-2-SSL always performs better than BL and BL-2, although in Y+F⇒M, the improvement is minor. When m is set optimally, DA-2-SSL performs significantly better than the two baseline methods.

In Figure 4.2, we show the comparison between BL-SSL, BL-SSL-2 and DA-2-SSL when

Method	m	F+M⇒Y			M+Y⇒F			Y+F⇒M		
		Prec	Rec	F1	Prec	Rec	F1	Prec	Rec	F1
BL-SSL	1000	0.588	0.414	0.486	0.632	0.149	0.241	0.616	0.365	0.458
	Opt	0.742	0.553	0.633	0.632	0.149	0.241	0.616	0.365	0.458
BL-SSL-2	1000	0.624	0.631	0.627	0.620	0.112	0.190	0.594	0.375	0.460
	Opt	0.624	0.631	0.627	0.620	0.112	0.190	0.594	0.375	0.460
DA-2-SSL	1000	0.727	0.741	0.734	0.377	0.248	0.300	0.395	0.579	0.470
	Opt	0.706	0.822	**0.759**	0.425	0.238	**0.305**	0.492	0.510	**0.501**

Table 4.2: Comparison between BL-SSL, BL-SSL-2, and DA-2-SSL.

m varies from 0 to 1000. We can make a number of observations from Figure 4.2. First, the performance of these semi-supervised learning methods does not always increase as m increases. Indeed, in semi-supervised learning, since the target instances added to the training set may not be labeled correctly, when more target instances are added, we may introduce noise and therefore decrease the performance. Second, except for BL-SSL with F+M⇒Y, the performance of the two baseline bootstrapping methods monotonically increases as m increases, but the performance of DA-2-SSL decreases as m increases when m is above a certain number. This comparison suggests that DA-2-SSL may be affected by the noise in the training data more than the baseline methods. Indeed, there is a tradeoff between adaptation and robustness. Because of the noise in the target instances with pseudo labels, when we adapt to the target domain using these target instances, we can pick up either correct or incorrect classification patterns. Since DA-2-SSL is a more aggressive adaptive method than BL-SSL and BL-SSL-2, it will both gain more from the correct information and suffer more from the noise contained in the target instances with pseudo labels. Nevertheless, DA-2-SSL outperforms both baselines in the whole spectrum of values of m. Also, in the cases of F+M⇒Y and M+Y⇒F, we can see that the performance of DA-2-SSL is still stable when m increases. In the case of Y+F⇒M, 600 is the threshold for m under which the performance is also stable. An important research question is how to automatically set m inside a safe range.

The difference between BL-SSL and DA-2-SSL is attributed to two factors: the difference between the pseudo labeled instances added to the training set, and the difference between the learning algorithms. In order to separate the two factors, we designed another diagnostic method,

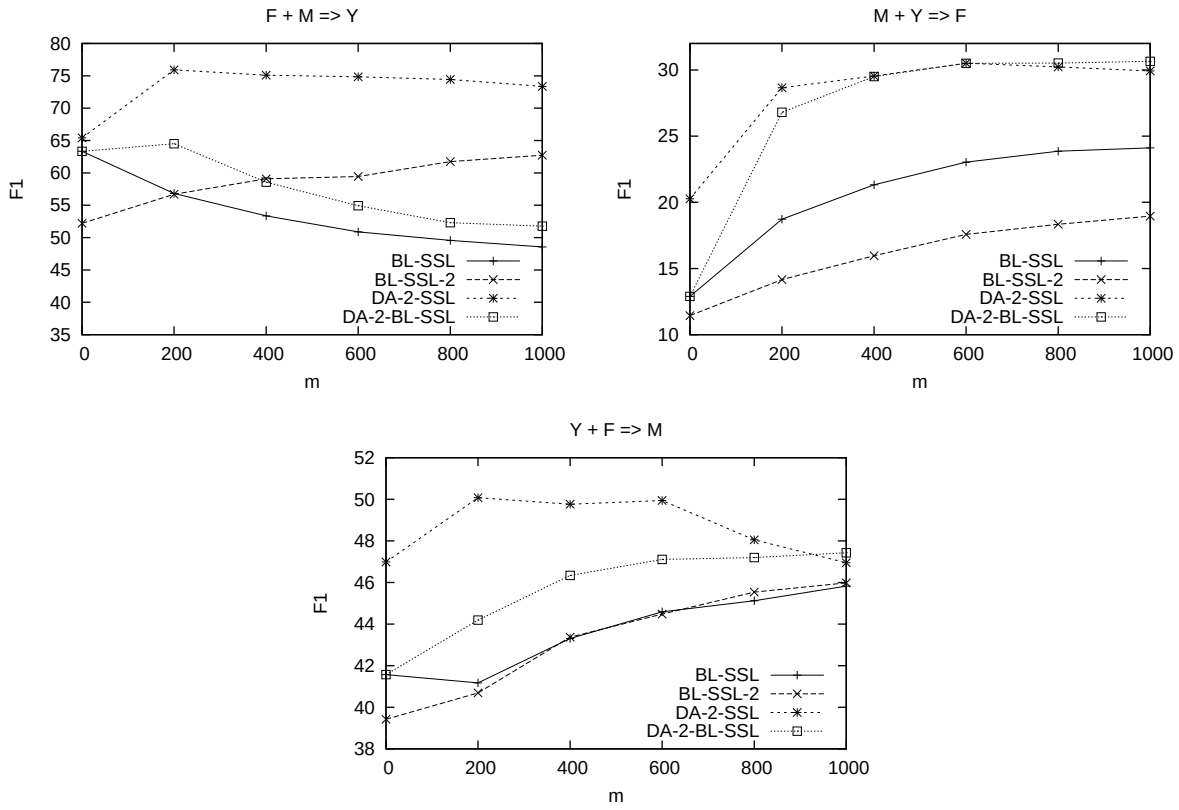

Figure 4.2: Comparison between BL-SSL, BL-SSL-2, DA-2-SSL and DA-2-BL-SSL as m Varies.

DA-2-BL-SSL, where we choose the best 200 sentences predicted by DA-2 to add to the training set at the first round of bootstrapping, but we use the regular bootstrapping method as in BL as the learning algorithm. In other words, DA-2-BL-SSL uses the same pseudo labeled sentences in the first round as DA-2-SSL, but uses the same learning algorithm as BL-SSL. We show the performance of DA-2-BL-SSL also in Figure 4.2. As we can see, compared with BL-SSL, DA-2-BL-SSL performed better, which suggests that the difference between BL-SSL and DA-2-SSL is indeed caused to some degree by the difference between the pseudo labeled sentences added to the training set. In other words, DA-2-SSL performed better than BL-SSL partly because DA-2 gave more accurate pseudo labels than BL to start with. Next, let us compare DA-2-BL-SSL with DA-2-SSL. For F+M⇒Y, DA-2-SSL performed consistently better than DA-2-BL-SSL for all values of m. For M+Y⇒F, when $m = 400, 600$, DA-2-BL-SSL performed similarly to DA-2-SSL, but when $m \geq 800$, DA-2-BL-SSL performed slightly better than DA-2-SSL. For Y+F⇒M,

DA-2-SSL again performed better than DA-2-BL-SSL except when $m = 1000$. This comparison between the two methods suggests that on the one hand, besides better pseudo labels, our domain adaptive learning algorithm in DA-2-SSL also contributed to the improvement over BL in many cases. On the other hand, when a relatively large number of pseudo labeled instances are used, the more aggressive method DA-2-SSL may introduce more noise, and hence not perform as well as the less aggressive, regular bootstrapping method. This again shows the importance of finding a good m in bootstrapping.

4.7 Experiments with A Single Source Domain

In this section, we show our experiment results with single source domains. We consider the tasks of part-of-speech tagging, entity type classification and spam filtering, and use the same data sets as we used in Chapter 3.

For each source-target domain pair, we change the size of $D_{t,l}$, and use Equation (4.14) to learn a classifier for the target domain. In all experiments, we set λ to 1, μ_s to 1000, and μ_t to 1. We use two strategies to choose h. First, we choose h by considering a range of values for h and selecting the optimal one. Second, we use the following heuristic way of choosing h: We first score features using Equation (4.13), and then select features whose scores are greater than or equal to 0. We call the first strategy *FS (Opt)* and the second strategy *FS (Def)*. We compare the feature selection method with a baseline method where $D_{t,l}$ is simply merged with D_s to train a classifier. We call this baseline method *BL*. The comparison between the three methods is shown in Figure 4.3, Figure 4.4 and Figure 4.5.

We can see from the figures that the two feature selection methods clearly outperform the baseline method. For entity type classification and spam filtering, we can also see that the advantage of feature selection over the baseline method is mostly obvious when $N_{t,l}$ is relatively small. We thus conclude that (1) feature selection is effective for domain adaptation when there is a single domain and there are some labeled target domain instances to help us identify generalizable features, and

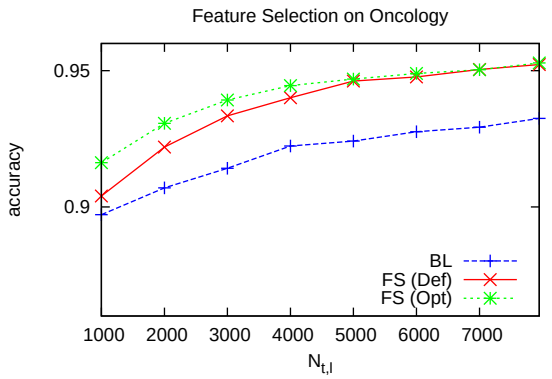

Figure 4.3: The effect of feature selection for part-of-speech tagging.

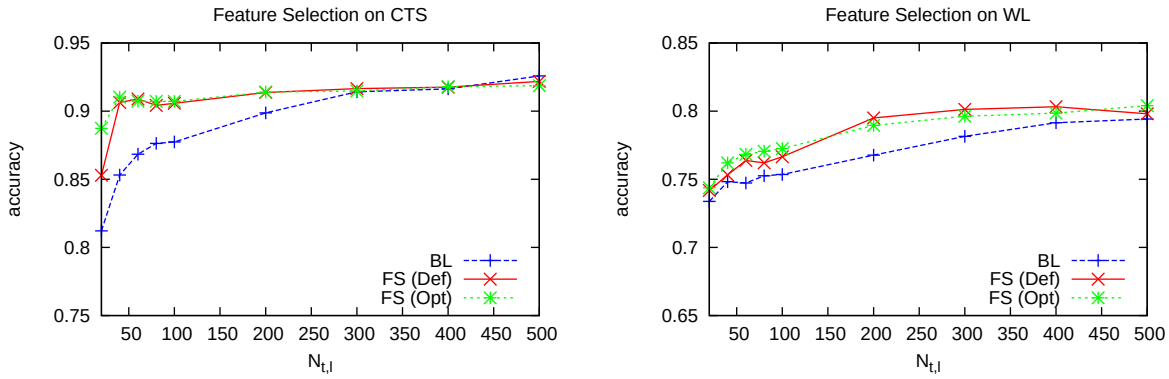

Figure 4.4: The effect of feature selection for entity type classification.

(2) the advantage of the feature selection method is mostly shown when the number of labeled target domain instances is relatively small, which is usually the situation we face.

The figures also show that the heuristic way of choosing h gives near optimal performance. We therefore recommend this heuristic way of setting h as the default way to choose h.

4.8 Summary

In this chapter, we studied a feature selection framework for domain adaptation. The general framework assumes multiple source domains, and consists of two stages. In the first stage, features that are useful for the classification task across different domains are identified and emphasized during training, and features that are only specifically useful for the source domains are

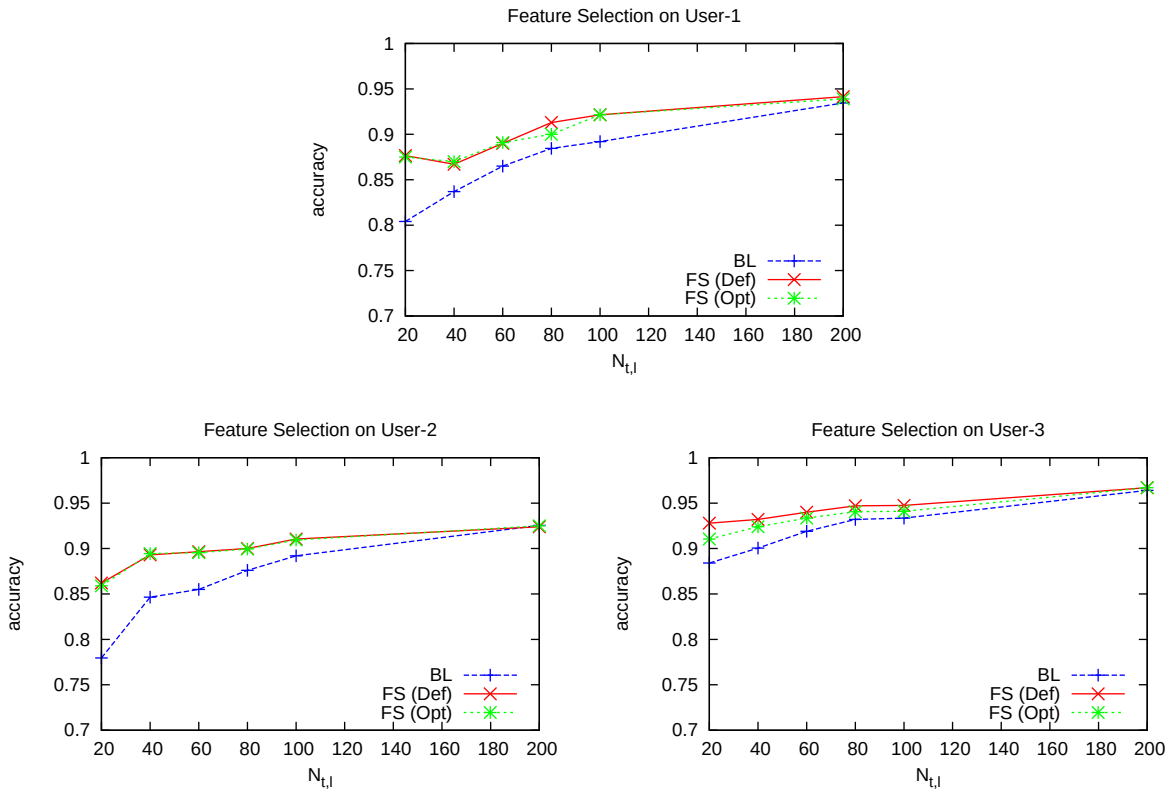

Figure 4.5: The effect of feature selection for spam filtering.

de-emphasized. In the second stage, bootstrapping is used to add instances from the target domain into the training data set. These pseudo labeled instances help learn features that are specifically useful for the target domain. Experiment results show that the two-stage feature selection framework is effective on the gene name recognition task. The framework can also be specialized for the case when there is a single source domain but there are some labeled instances from the target domain. In this case, the generalization stage and the adaptation stage become a single step. Experiments on part-of-speech tagging, entity type classification and spam filtering show that the feature selection framework is also effective for this case when there is only a single source domain.

Chapter 5

An Analysis of Domain Difference Types

In previous chapters, we focused on developing solutions to domain adaptation, and studied a number of techniques, including instance weighting, feature selection, and semi-supervised learning, which have all shown to be generally useful for domain adaptation. However, we also observed that the effectiveness of these techniques varies from data set to data set, and sometimes a technique may fail to work completely. A reasonable hypothesis to explain this phenomenon is that there exist different types of domain difference, and each domain adaptation technique is only suitable for certain types of domain difference. If applied inappropriately, a domain adaptation technique may even hurt the classification performance, especially when it is hard to optimally set certain learning parameters. Therefore, there is a need to study how to identify and characterize different types of domain difference and how to associate various domain adaptation techniques with these different types of domain difference. This research question has not been formally explored in previous chapters or any other existing study on domain adaptation. We study this research question in this chapter.

When we analyzed domain adaptation in Chapter 2, we already discussed some intuitions about what may cause two domains to differ. One perspective from which we can separate domain difference into different types is to decompose $P(X, Y)$ into $P(X)$ and $P(Y|X)$ as in Section 2.3.1, and to consider the difference between two domains in each of these two probability components. In our instance weighting framework, we also identified two weighting parameters, α and β, to address these two types of difference separately.

Another perspective to separate different types of domain difference is related to our discussion on two-stage domain adaptation in Section 2.3.3 of Chapter 2. From this perspective, we check

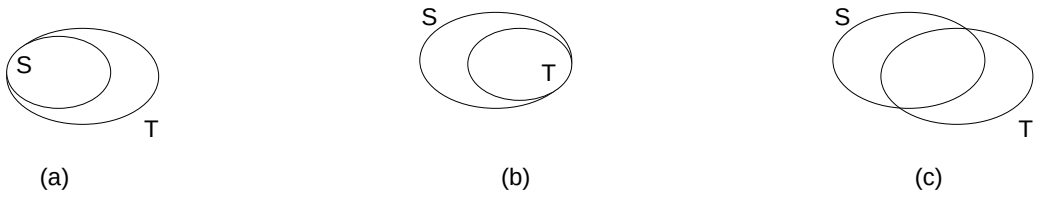

Figure 5.1: An abstract illustration of the different overlapping patterns between two domains.

Figure 5.2: Two types of domain difference based on where the domain-specific characteristics come from.

whether the domain difference comes from some special characteristics in the target domain or special characteristics in the source domain, or in both. By special characteristics, we vaguely mean anything about X and Y that may be important for the classification task, and we will materialize this abstract concept later. An abstract illustration of this second perspective is given in Figure 5.1 and explained by three different overlapping patterns between the source and the target domains. As we can see, difference between two domains may come from some special characteristics that are only observed in the target domain but not in the source domain, as diagram (a) in Figure 5.1 depicts, or from some special characteristics observed in the source domain only, as diagram (b) depicts, or both, as in diagram (c). Although we believe that in general two domains overlap with the pattern shown in diagram (c), it is still a good idea to separate the two types of difference as shown in Figure 5.2 and to study them individually.

There are several reasons why we want to devote another chapter to discussions of issues related to different types of domain difference. First of all, the two perspectives, namely, the perspective based on probability decomposition and the perspective based on domain-specific characteristics, are orthogonal, but they have not been considered together in previous chapters. For example, there may be completely new features such as new vocabulary words that are only ob-

served in the target domain but not present in the source domain, which can be considered to be diagram (a) in Figure 5.2 coupled with difference in $P(X)$. Or, there may be features that are useful for classification in the source domain but become not useful in the target domain, which can be considered to be diagram (b) in Figure 5.2 coupled with difference in $P(Y|X)$. Therefore, by combining these two perspectives in this chapter, we can get a better understanding of the different types of domain difference.

Second, we have introduced several domain adaptation techniques, motivated by different need in domain adaptation. For example, in Chapter 4, we argued that to learn target-domain-specific features, we should consider using semi-supervised learning. With a systematic analysis of domain difference types in this chapter, we can also associate different domain adaptation techniques with different types of domain difference in a systematic way. Presumably, different techniques address different types of domain difference, and therefore should be complementary to each other. But some technique may be useful for more than one types of domain difference, and some different techniques may have similar effects on the same type of domain difference. It is therefore natural to summarize all these associations in this chapter.

Third, after identifying and characterizing different types of domain difference, and associating them with different domain adaptation techniques, it is important to empirically verify the hypothesized associations. However, the real data sets used in previous chapters are usually mixtures of different domain difference types, preventing us from separate, controlled studies of each domain difference type. Indeed, although there are several real data sets that have been used for studying domain adaptation in a number of studies, we still lack standard benchmark data sets for evaluation of domain adaptation methods. In this chapter, we propose to perturb some real data sets and construct new data sets that each represent a single type of domain difference. These perturbed data sets allow us to conduct controlled studies of the properties of each domain difference type and its associations with different domain adaptation techniques. These data sets can also be used for comparing different domain adaptation techniques designed for the same type of domain difference.

Finally, all the analysis of domain difference types and their associations with domain adaptation techniques serves the purpose of obtaining a better understanding of domain adaptation and helping choose or design techniques to tailor to the need of specific domain adaptation problems.

The rest of this chapter is organized as follows. In Section 5.1, we characterize different types of domain difference based on the changes of feature properties across domains. Then in Section 5.2, we associate the different domain difference types with the domain adaptation techniques we have introduced in previous chapters. In Section 5.3, we use perturbed read data sets to show that different domain difference types are indeed associated with different domain adaptation techniques. Finally in Section 5.4 we make our conclusions based on the analysis and the empirical experiment results.

5.1 Characterizing Domain Difference Based on Changes of Feature Properties

As we briefly discussed earlier, we want to consider two perspectives from which we can separate domain difference into different types. First, we want to separate difference in $P(X)$ between two domains from difference in $P(Y|X)$. Second, we want to separate difference caused by "novel" characteristics in the target domain from difference caused by "noisy" characteristics in the source domain. Here, by "novel" characteristics, we mean characteristics specific to the target domain but not covered in the source domain, and by "noisy" characteristics, we mean characteristics specific to the source domain but not useful in the target domain. Since the two perspectives are orthogonal to each other, we need to consider target-specific characteristics and source-specific characteristics in both $P(X)$ and $P(Y|X)$.

However, although we have the intuitions about these two perspectives, it is still very vague as of what domain-specific characteristics in $P(X)$ and in $P(Y|X)$ mean. To make the concepts concrete, we propose to look at the properties of individual features in each domain and how these properties differ between domains. We then define domain difference types based on the changes

of the feature properties across domains. Note that this approach is a simplified view because we assume that features are independent so that we can study their properties independently. Although this assumption does not hold in reality, it is a reasonable starting point.

5.1.1 Feature Properties

We consider two feature properties, roughly corresponding to $P(X)$ and $P(Y|X)$, respectively. Because we are mostly interested in domain adaptation in natural language processing problems, here we assume that all features are binary.

- **Feature Frequency**

  For each feature f, and given a sample of instances $\{x_i\}_{i=1}^N$ (with or without labels), we compute the frequency of the feature as follows:

  $$\text{Freq}_f \quad = \quad \frac{\sum_{i=1}^N f(x_i)}{N},$$

  where N is the sample size, and $f(x_i)$ is 1 if feature f is present in x_i or 0 otherwise. The feature frequency is essentially the probability of observing the feature as estimated from the sample of instances.

  Usually, because features are dependent, $P(X)$ cannot be decomposed into a function of the probabilities of individual features. However, if we observe a change of a feature's frequency from the source domain to the target domain, this change is likely to indicate some change of $P(X)$ across the two domains. Therefore, we propose to use feature frequencies to characterize the difference between two domains in $P(X)$.

- **Information Gain**

  To describe difference in $P(Y|X)$, i.e. to consider the relation between X and Y, we want to use some correlation measure between X and Y. While there are many choices available, here we choose to use information gain because it is widely used and shown to be effective

for tasks such as feature selection. The information gain of a feature f is the mutual information between f and Y, or in another word, the reduction in the entropy of Y achieved by knowing the value of f.

For a feature f, and given a sample of *labeled* instances $\{(x_i, y_i)\}_{i=1}^N$, the information gain of f is computed as follows:

$$\mathrm{IG}_f = H(Y) - H(Y|f),$$

where

$$\begin{aligned} H(Y) &= -\sum_{y \in \mathcal{Y}} P(Y=y) \log P(Y=y), \\ P(Y=y) &= \frac{\sum_{i=1}^N \delta(y_i, y)}{N}, \end{aligned}$$

and

$$\begin{aligned} H(Y|f) &= P(f=1)H(Y|f=1) + P(f=0)H(Y|f=0) \\ &= -\Bigg(P(f=1) \sum_{y \in \mathcal{Y}} P(Y=y|f=1) \log P(Y=y|f=1) \\ &\quad + P(f=0) \sum_{y \in \mathcal{Y}} P(Y=y|f=0) \log P(Y=y|f=0) \Bigg), \\ P(f=1) &= \mathrm{Freq}_f, \\ P(f=0) &= 1 - P(f=1), \\ P(Y=y|f=1) &= \frac{\sum_{i=1}^N \delta(y_i, y) f(x_i)}{\sum_{i=1}^N f(x_i)}, \\ P(Y=y|f=0) &= \frac{\sum_{i=1}^N \delta(y_i, y)(1 - f(x_i))}{\sum_{i=1}^N (1 - f(x_i))}. \end{aligned}$$

Here $\delta(y_1, y_2)$ is the Kronecker delta.

As we can see, information gain is related to both Y and f, and f is part of X. If Y is

completely independent of f, $H(Y|f)$ is equal to $H(Y)$ and IG_f is 0; otherwise, IG_f is a positive number. IG_f is upper-bounded by $H(Y)$.

It is easy to see that IG_f has some direct relation to $P(Y|X)$. IG_f depends on $H(Y|f)$, which is computed from $P(Y|f)$, and $P(Y|f)$ is related to $P(Y|X)$ because f is part of X. Therefore, we propose to approximately characterize the difference in $P(Y|X)$ between two domains by looking at the difference in the information gains of features between the two domains.

5.1.2 Change of Feature Properties

The two feature properties we consider roughly correspond to the two types of domain difference based on the probability decomposition $P(X,Y) = P(X)P(Y|X)$. Now for each feature property, we consider its possible change from the source domain to the target domain, in order to capture the two types of domain difference based on where the domain-specific characteristics come from.

- **Feature Frequency**

  The frequency of a feature can change in two possible ways: changing from infrequent to frequent, and from frequent to infrequent. These two changes can be exactly regarded as the two cases depicted in Figure 5.2. When the target domain contains a considerable number of features that are frequent in the target domain but infrequent in the source domain, we can regard this case as in diagram (a) in Figure 5.2, where the target domain contains special characteristics about X. Similarly, when the source domain contains a considerable number of features that are frequent in the source domain but not in the target domain, it is the case as in diagram (b) in Figure 5.2, where the source domain contains special characteristics about X.

- **Information Gain**

Similar to feature frequency, we also consider two possible changes of the information gain of a feature from the source domain to the target domain: changing from non-discriminative to discriminative, and from discriminative to non-discriminative. These two cases again correspond to the two diagrams in Figure 5.2. When the target domain contains a considerable number of features that have high correlations with certain class labels in the target domain but are not discriminative in the source domain, we can regard this case as in diagram (a) in Figure 5.2, but this time the target domain contains special characteristics related to $P(Y|X)$ instead of $P(X)$. Similarly, when the target domain contains a considerable number of features that are discriminative in the source domain but become non-discriminative in the target domain, it is the case as in diagram (b) in Figure 5.2, with special characteristics related to $P(Y|X)$ in the source domain.

Note that the notions of frequent vs. infrequent features and of discriminative vs. non-discriminative features are subjective. Since feature frequency and feature discriminativeness as measured by information gain are continuous, there is no absolute definition for frequent features or discriminative features. In Section 5.3, when we construct perturbed data sets, we use some cut-off numbers to define frequent/infrequent features and discriminative/non-discriminative features.

5.1.3 Domain Difference Types

Based on our analysis above, we consider the following four types of domain difference. We use *F* to denote frequent features, *I* to denote infrequent features, *D* to denote discriminative features, or features with high information gains, and *N* to denote non-discriminative features, or features with low information gains.

- **[I→F]** This type of domain difference is characterized by a set of features that are infrequent in the source domain but frequent in the target domain.
- **[F→I]** This type of domain difference is characterized by a set of features that are frequent in the source domain but infrequent in the target domain.

- **[N→D]** This type of domain difference is characterized by a set of features that are non-discriminative in the source domain but discriminative in the target domain.
- **[D→N]** This type of domain difference is characterized by a set of features that are discriminative in the source domain but non-discriminative in the target domain.

5.2 Associating Domain Difference Types with Domain Adaptation Techniques

In this section, we analytically associate several domain adaptation techniques with the domain difference types we identified in the previous section.

5.2.1 Domain Adaptation Techniques

In previous chapters, we presented two general frameworks to address the domain adaptation problem, namely, an instance weighting framework and a feature selection framework. We also incorporated semi-supervised learning into both frameworks. Although instance weighting and feature selection are the basis of the two frameworks, several ideas are included in these two frameworks. We summarize them into the following five individual techniques. The first two techniques can be used in unsupervised domain adaptation, when no labeled target domain instance is available, while the last three techniques has to be used in supervised domain adaptation, when we have some labeled target domain instances.

- **[SSL]** Semi-supervised learning

  In Chapter 3 and Chapter 4, we showed that semi-supervised learning is very effective in improving the classification performance of the various NLP domain adaptation problems we considered. Between the two semi-supervised learning methods we evaluated, i.e. bootstrapping and entropy minimization, we found that bootstrapping is more effective. In our experiments in this chapter, we use bootstrapping as our semi-supervised learning methods.

- **[IW-α]** Instance weighting on source domain instances using α

  In Chapter 3, we showed that instance weighting on the source domain instances using $\alpha \propto \frac{P_t(x,y)}{P_s(x,y)}$ is useful for domain adaptation. In this chapter, we use the mean matching method to set the α's.

- **[BAL]** Balancing the labeled source domain instances and the labeled target domain instances

  In Chapter 4, we showed that setting the parameter $\lambda_{t,l}$ to $\frac{N_s}{N_{t,l}}$, namely, to make the total contribution of the labeled target domain instances to be equal to that of the labeled source domain instances in the objective function, can improve the performance over the standard method where target domain instances are each weighted the same as a single source domain instance. We refer to this technique as balancing the labeled instances from the two domains.

- **[IW-β]** Instance weighting on source domain instances using β

  In Chapter 3, it was shown that using $\beta \propto \frac{P_t(y|x)}{P_s(y|x)}$ to weigh the source domain instances is also effective. In this chapter, we use logistic regression models to estimate $P_t(y|x)$ and $P_s(y|x)$ to set the β's.

- **[FS]** Feature selection on source domain instances

  In Chapter 4, we showed that if we separate the weight vector for the generalizable features from the weight vector for the domain-specific features, we can obtain better performance. In this chapter, we use the domain cross validation heuristic to rank and select features. To decide the number of features to choose, we rank the features using the scoring function as in Equation (4.13), and choose features whose scores are greater than or equal to zero.

We use the abbreviations as shown above in square brackets to denote the different techniques. As we can see, IW-α, IW-β and BAL come from the instance weighting framework, FS comes from the feature selection framework, and SSL is used in both frameworks. SSL and IW-α can

be applied to unsupervised domain adaptation while IW-β, BAL and FS can only be applied to supervised domain adaptation.

5.2.2 Associations

How are the domain adaptation techniques identified above related to the different types of domain difference identified in Section 5.1? Let us go through the techniques above.

Both SSL and BAL add target domain instances into the training data. Therefore, we expect them to be useful for the case when the target domain has special characteristics that are not covered in the source domain, that is, they are expected to work for I→F and N→D. The difference between SSL and BAL is that in SSL, the labels of the target domain instances are predicted and therefore not guaranteed to be correct. Therefore, SSL may or may not improve the performance, depending on how much noise there is in the predicted labels. On the other hand, SSL can potentially make use of all the unlabeled target domain instances in $D_{t,u}$, while BAL only uses the small number of instances in $D_{t,l}$. So it is possible for SSL to bring more performance improvement than BAL when the size of $D_{t,l}$ is small.

While the major contribution of SSL and BAL is likely to be bringing new characteristics in the target domain, inclusion of target domain instances may also have the side effect of reducing the "noise" in the source domain. For example, suppose we have some source-domain-specific features that are discriminative in the source domain but non-discriminative in the target domain. When target domain instances are added to the training set through SSL or BAL, the discriminativeness of these features will decrease, making these features less useful. Therefore, we also expect SSL and BAL to work to some degree for F→I and D→N.

IW-α tries to weigh the source domain instances to make the transformed distribution of X to be similar to that in the target domain. It therefore should remove noise caused by source-domain-specific characteristics in $P(X)$. Since we use change of feature frequencies to describe change of $P(X)$ when we define domain difference types, we expect IW-α to work for F→I.

IW-β and FS both transform the source domain instances by considering the difference between

Domain Difference Type	Domain Adaptation Techniques
I→F	BAL, SSL
F→I	IW-α, BAL, SSL
N→D	BAL, SSL
D→N	IW-β, FS, BAL, SSL

Table 5.1: Our expectation of the associations between different types of domain difference and different domain adaptation techniques.

$P_t(Y|X)$ and $P_s(Y|X)$. Therefore, they are expected to work well for the situation when there are "noisy" characteristics in the source domain about $P(Y|X)$, namely, the domain difference type D→N.

The analysis above is summarized in Table 5.1. As we can see, some domain adaptation techniques are expected to be useful for more than one domain difference types, and a domain difference type can have more than one solutions.

5.3 Experiments with Perturbed Real Data Sets

In this section, we use perturbed real data sets to verify the associations between domain difference types and domain adaptation techniques. We first describe how we construct the perturbed data sets in Section 5.3.1. We then show the performance of the different techniques on the perturbed data sets in Section 5.3.2.

5.3.1 Data Construction

In this subsection, we describe how we perturb a real data set to generate several versions of the original data set, each representing a single type of domain difference. Because we are constructing perturbed data sets for the purpose of evaluation, we assume that we have complete knowledge about the true labels of the target domain instances we have.

To begin with, suppose we are given a real data set that contains a large amount of labeled instances in both a source domain and a target domain. For each domain difference type, our idea

is to select a subset of features such that when the instances are represented with this feature subset, the two domains mainly contain the specified domain difference type.

Since in all types of domain difference, the source and the target domains still share a common part, our first step is to select a *core* set of features whose properties do not change much across the two domains. In order to do this, we first compute the feature frequencies and information gains in both domains, and choose features that are considered to be frequent and discriminative in both domains. We do this by choosing features whose frequencies and information gains are above certain thresholds. In our experiments, the frequency threshold for a domain is chosen such that 10% of the features in the domain are considered to be frequent. The information gain threshold is chosen similarly. We then rank the selected features by the summation of each feature's normalized frequency and information gain, and select the top 100 features as the *core* feature set.

For each domain difference type, we then choose an additional set of 100 features that have the corresponding feature property change as described in the definition of the domain difference type in Section 5.1.3. For example, for the domain difference type I→F, we choose 100 features that are considered to be infrequent in the source domain but frequent in the target domain, and combine these 100 features with the core feature set to form the final feature set for I→F.

The real data set we use as a starting point for perturbation is from Task A of the ECML 2006 Discovery Challenge. This is the same data set for spam filtering as we used in Chapter 3 and Chapter 4. We always use the public email collection as the source domain, and use each of the three personal email collections as the target domain. We thus get three groups of perturbed data sets, each group containing four perturbed data sets, corresponding to I→F, F→I, N→D, and D→N. We use 4,000 emails in the public collection as source domain training data, and 2,000 emails from each personal collection as target domain test data. An additional 200 emails from each personal collection are set aside and used to draw labeled target domain instances.

Domain Difference Type	BL	Domain Adaptation				
		SSL	IW-α	BAL	IW-β	FS
I→F	0.826	**0.839**	0.826	**0.856**	**0.829**	**0.846**
N→D	0.824	**0.843**	0.812	**0.861**	**0.849**	**0.841**
F→I	0.752	0.751	**0.754**	**0.786**	**0.784**	**0.793**
D→N	0.762	**0.769**	**0.784**	**0.790**	**0.783**	**0.791**

Table 5.2: Comparison between domain difference types and domain adaptation techniques on User-1's email collection.

Domain Difference Type	BL	Domain Adaptation				
		SSL	IW-α	BAL	IW-β	FS
I→F	0.877	**0.889**	0.859	**0.885**	0.862	0.872
N→D	0.880	**0.887**	**0.883**	**0.883**	0.864	0.870
F→I	0.772	0.772	0.764	**0.811**	**0.803**	**0.817**
D→N	0.776	**0.780**	0.766	**0.809**	**0.798**	**0.810**

Table 5.3: Comparison between domain difference types and domain adaptation techniques on User-2's email collection.

5.3.2 Experiments

We first compare the performance of the five different domain adaptation techniques on the four types of domain difference for each personal email collection. We set $N_{t,l}$ to 200 and $N_{t,u}$ to 2,000. The performance is shown in Table 5.2, Table 5.3 and Table 5.4. The column "BL" shows the performance when only labeled instances from the source domain are used for training without instance weighting or feature selection. For SSL, we use the bootstrapping method, where we continue the iterations until all unlabeled target domain instances are added to the training set. For IW-α, we use the mean matching method as described in Chapter 3. In IW-β and FS, these labeled target domain instances are not only used to find instance weights or generalizable features but also included in the final training set. We use classification accuracy as the performance measure. For the accuracies obtained by the five domain adaptation techniques, we compare them with the corresponding baseline accuracy, and highlight those numbers that improve over the baseline.

As we can see from the tables, the experiment results generally meet our expectations of the effectiveness of the different domain adaptation techniques on the domain difference types, although we also observe something that we did not expect. First, for I→F and N→D, we expected BAL to

Domain Difference Type	BL	Domain Adaptation				
		SSL	IW-α	BAL	IW-β	FS
I→F	0.811	**0.853**	0.794	**0.850**	0.811	**0.853**
N→D	0.816	**0.859**	0.801	**0.855**	0.801	0.811
F→I	0.676	**0.680**	0.656	**0.730**	**0.712**	**0.755**
D→N	0.695	0.687	0.694	**0.744**	**0.729**	**0.756**

Table 5.4: Comparison between domain difference types and domain adaptation techniques on User-3's email collection.

help, and indeed for all three users, BAL improved the baseline. For these two domain difference types, we also expected SSL to help, although SSL is not guaranteed to work because of the noise pseudo labels may contain. It is shown that SSL improved over baseline as well, although for User-1, the improvement was not substantial as BAL. We did not expect the other domain adaptation techniques to help for these two domain adaptation types, and as shown in the tables, the other domain adaptation techniques sometimes helped, but not consistently. Furthermore, these other techniques may even hurt the performance sometimes.

Second, for F→I, we expected IW-α and BAL to help. However, we instead found that BAL, IW-β and FS consistently helped but IW-α only helped for User-1. The explanation may be that in our perturbation of the data, when we selected features whose frequencies change across the domains, we did not force their discriminativeness to remain the same. In another word, we did not separate the change of $P(X)$ completely from the change of $P(Y|X)$. As a result, F→I also contains change of $P(Y|X)$. We therefore observe IW-β and FS to help because they are designed to help for change in $P(Y|X)$. On the other hand, the observation that IW-α did not consistently help suggests that IW-α is not very robust.

To further verify the associations between domain difference types and domain adaptation techniques, we run another set of experiments. We try different sizes of $D_{t,l}$, ranging from 20 to 200 with a step size of 20. For each size, we apply the five domain adaptation techniques. The performance curves as $N_{t,l}$ changes are shown in Figure 5.3, Figure 5.4 and Figure 5.5. We also plot the curves for the baseline method, where standard supervised learning is performed with D_s and $D_{t,l}$ simply combined.

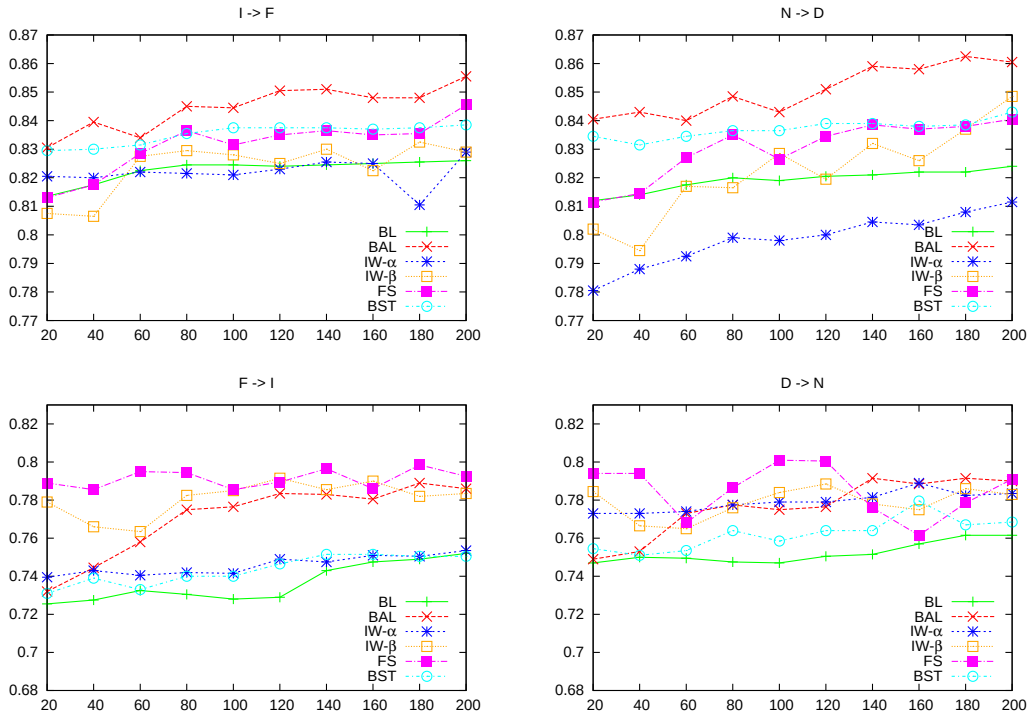

Figure 5.3: Comparison between the domain adaptation techniques as $N_{t,l}$ changes on User-1's email collection.

The three figures show stronger patterns of the associations between the domain difference types and the domain adaptation techniques. First of all, we can generally conclude that I→F and N→D should be combined into a single type while F→I and D→N should be combined into another single type. In another word, we only need to differentiate between domain difference caused by "novel" target domain characteristics and that caused by "noisy" source domain characteristics. Second, for I→F and N→D, BAL and SSL help improve the baseline performance the most. Third, for F→I and D→N, FS, BAL and IW-β generally all help, although FS is the most robust, followed by BAL, and IW-β may not help sometimes. Third, for F→I and D→N, IW-α sometimes helps, such as for User-1, but in general does not change the baseline performance much. For I→F and N→D, however, IW-α may hurt the performance.

The findings above show that if we can correctly identify the domain adaptation type, then we can select the most effective domain adaptation techniques accordingly. Otherwise, we should only apply those robust techniques which do not hurt the performance much when applied to the

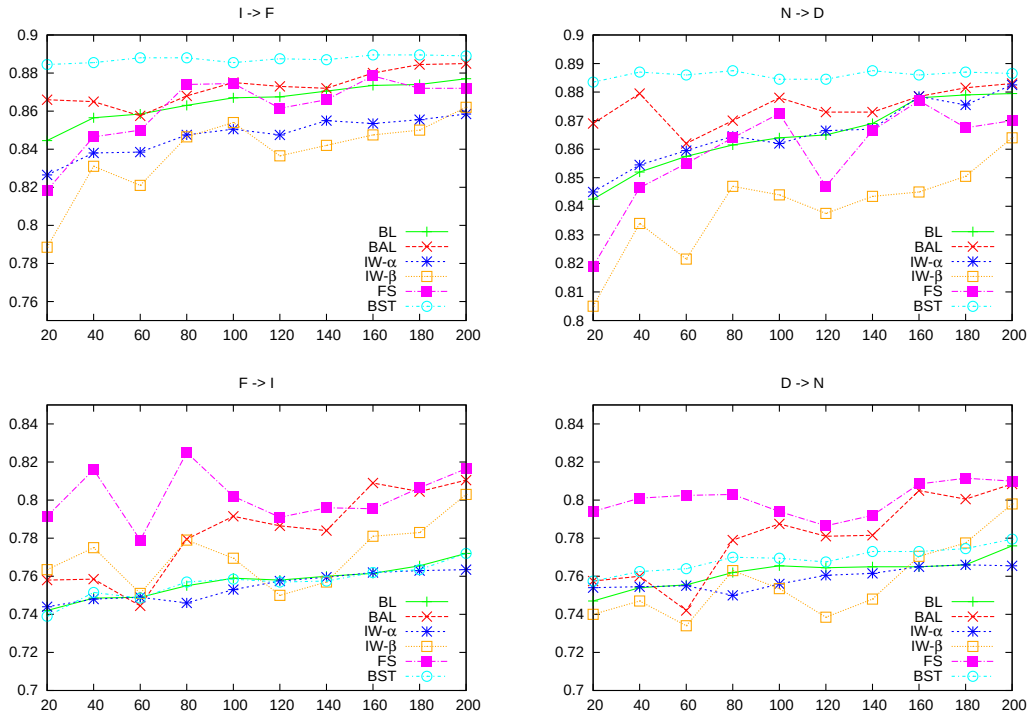

Figure 5.4: Comparison between the domain adaptation techniques as $N_{t,l}$ changes on User-2's email collection.

wrong domain difference type. The figures suggest that BAL, SSL and FS are relatively more robust than the other techniques.

5.4 Conclusions and Discussions

In this chapter, we studied the problem of how to identify and characterize different types of domain difference, and how these domain difference types are associated with different domain adaptation techniques. We looked at domain difference from two perspectives, one based on probability decomposition and the other based on domain-specific characteristics. Combining the two perspectives, we obtained four different types of domain difference. We analytically associated these domain different types with five typical domain adaptation techniques that we used in previous chapters.

To verify the hypothesized associations we established, we constructed several data sets by

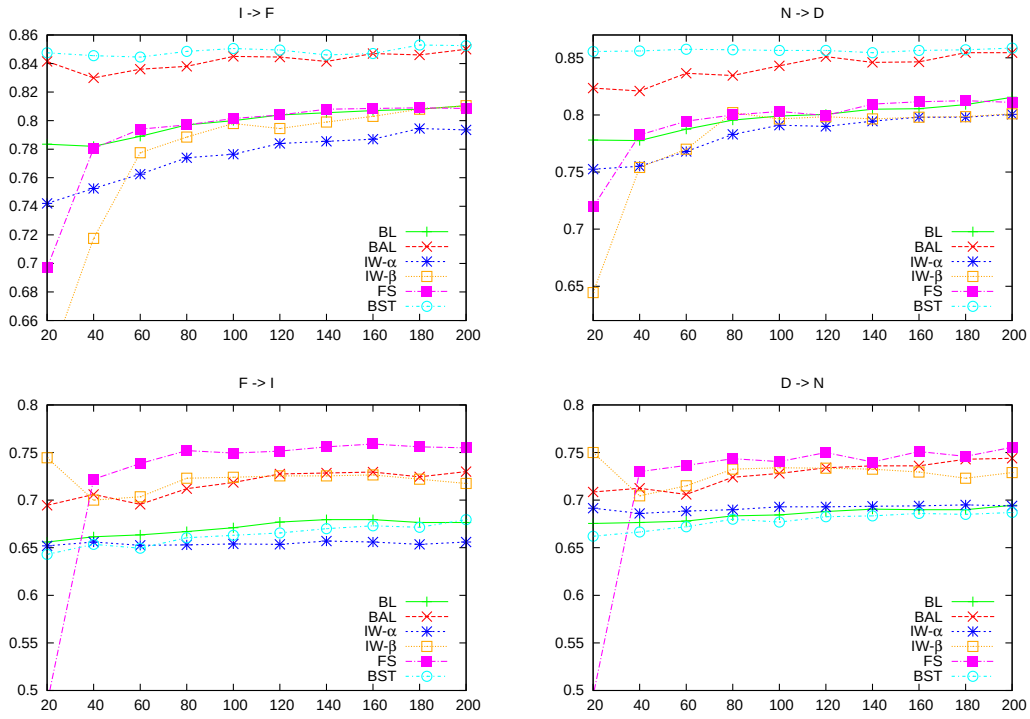

Figure 5.5: Comparison between the domain adaptation techniques as $N_{t,l}$ changes on User-3's email collection.

perturbing some real data sets. The constructed data sets each contain a single type of domain difference. Experiment results on these constructed data sets show that indeed different domain difference types are associated with different domain adaptation techniques.

Our general conclusions are as follows. For domain difference that is caused by "novel" characteristics in the target domain, feature selection and instance weighting on the source domain are not useful because these techniques aims at removing "noise" from the source domain. Instead, increasing the contribution of the labeled target domain instances or performing semi-supervised learning is the most effective. When the domain difference is caused by "noisy" characteristics from the source domain, and when we have a small amount of labeled target domain instances, we should try to use the small amount of target domain instances to help identify the noise in the training domain and remove the noise through feature selection, instance weighting, or increasing the contribution of the labeled target domain instances. Feature selection is generally the most effective choice among the three.

The analysis in this chapter shows that for different domain adaptation problems, different domain adaptation techniques should be applied. How to choose the appropriate techniques depends on properties of the domain difference in a particular domain adaptation problem. Our analysis provides some insight into the association of domain adaptation techniques with domain difference types, and deepens our understanding of domain adaptation. To apply the analysis to solving real problems, however, we need to design measures of domain difference types that do not rely on the availability of labels in the target domain. What we found in this chapter suggests that changes of feature frequencies are often associated with changes of feature discriminativeness. Since feature frequencies can be measured without knowing any class labels, in the future, we plan to study whether we can quantify the change of feature frequencies from a source domain to a target domain and whether we can give good recommendations of domain adaptation techniques based on this quantity.

Chapter 6

Related Work

Although the domain adaptation problem is a fundamental problem in machine learning, it only started gaining much attention very recently (Chelba and Acero, 2004; Daumé III and Marcu, 2006; Blitzer et al., 2006; Ben-David et al., 2007; Daumé III, 2007; Satpal and Sarawagi, 2007; Blitzer et al., 2008). However, some special kinds of domain adaptation problems have been studied before under different names including class imbalance (Japkowicz and Stephen, 2002), covariate shift (Shimodaira, 2000), and sample selection bias (Zadrozny, 2004; Heckman, 1979). There are also some closely-related but not equivalent machine learning problems that have been studied extensively, including multi-task learning (Caruana, 1997) and semi-supervised learning (Chapelle et al., 2006; Zhu, 2005).

In this chapter, we review some existing work in both the machine learning and the natural language processing communities related to domain adaptation. The goal of this literature review is twofold. First, there have been a number of methods proposed to address domain adaptation, but it is not clear how these methods are related to each other. This review thus tries to organize the existing work and lay out an overall picture of the domain adaptation problem with its possible solutions. Second, a systematic literature review naturally reveals the limitations of current work and points out promising directions that should be explored in the future.

6.1 Overview

Recently, there have been a number of studies related to domain adaptation. However, the motivating ideas behind these methods are different. To connect the existing work and hence to better

understand the problem, in the following sections, we organize the existing work into several categories from our own viewpoint. First, in Section 6.2, we consider a line of work that is based on instance weighting. In Section 6.3, we look at some work that bears strong resemblance to semi-supervised learning. In Section 6.4, we review another line of work that is based on changing the representation of X. Section 6.5 reviews work using Bayesian priors, and Section 6.6 reviews work related to multi-task learning. In Section 6.7, ensemble methods for domain adaptation are considered.

The categories are ordered in this way so that methods in Section 6.2, Section 6.3 and Section 6.4 are generally applicable to unsupervised domain adaptation problems, while methods in Section 6.5 and Section 6.6 can only handle supervised domain adaptation problems.

6.2 Instance Weighting

One general approach to addressing the domain adaptation problem is to assign instance-dependent weights to the loss function when minimizing the expected loss over the distribution of data. To see why instance weighting may help, let us first briefly review the empirical risk minimization framework for standard supervised learning (Vapnik, 1999), and then informally derive an instance weighting solution to domain adaptation. Let Θ be a model family from which we want to select an optimal model θ^* for our classification task. Let $l(x, y, \theta)$ be a loss function. Strictly speaking, we want to minimize the following objective function in order to obtain the optimal model θ^* for the distribution $P(X, Y)$:

$$\theta^* = \arg\min_{\theta \in \Theta} \sum_{(x,y) \in \mathcal{X} \times \mathcal{Y}} P(x, y) l(x, y, \theta).$$

Because $P(X, Y)$ is unknown, we can use the empirical distribution $\tilde{P}(X, Y)$ to approximate $P(X, Y)$. Let $\{(x_i, y_i)\}_{i=1}^N$ be a set of training instances randomly sampled from $P(X, Y)$. We

then minimize the following empirical risk in order to find a good model $\hat{\theta}$:

$$\begin{aligned}\hat{\theta} &= \arg\min_{\theta\in\Theta} \sum_{(x,y)\in\mathcal{X}\times\mathcal{Y}} \tilde{P}(x,y)l(x,y,\theta) \\ &= \arg\min_{\theta\in\Theta} \sum_{i=1}^{N} l(x_i,y_i,\theta).\end{aligned}$$

Now consider the setting of domain adaptation. Ideally, we want to find an optimal model for the target domain that minimizes the expected loss over the target distribution:

$$\theta_t^* = \arg\min_{\theta\in\Theta} \sum_{(x,y)\in\mathcal{X}\times\mathcal{Y}} P_t(x,y)l(x,y,\theta).$$

However, our training instances, $D_s = \{(x_i^s, y_i^s)\}_{i=1}^{N_s}$, are randomly sampled from the source distribution $P_s(X,Y)$. We can rewrite the equation above as follows:

$$\begin{aligned}\theta_t^* &= \arg\min_{\theta\in\Theta} \sum_{(x,y)\in\mathcal{X}\times\mathcal{Y}} \frac{P_t(x,y)}{P_s(x,y)} P_s(x,y)l(x,y,\theta) \\ &\approx \arg\min_{\theta\in\Theta} \sum_{(x,y)\in\mathcal{X}\times\mathcal{Y}} \frac{P_t(x,y)}{P_s(x,y)} \tilde{P}_s(x,y)l(x,y,\theta) \\ &= \arg\min_{\theta\in\Theta} \sum_{i=1}^{N_s} \frac{P_t(x_i^s,y_i^s)}{P_s(x_i^s,y_i^s)} l(x_i^s,y_i^s,\theta). \qquad (6.1)\end{aligned}$$

As we can see, weighting the loss for the instance (x_i^s, y_i^s) with $\frac{P_t(x_i^s,y_i^s)}{P_s(x_i^s,y_i^s)}$ provides a well-justified solution to the domain adaptation problem.

It is not possible to compute the exact value of $\frac{P_t(x,y)}{P_s(x,y)}$ for a pair (x,y), especially because we do not have enough labeled instances in the target domain. Section 6.2.1 reviews one line of work in which $P_t(X|Y) = P_s(X|Y)$ is assumed, while Section 6.2.2 reviews another line of work in which $P_t(Y|X) = P_s(Y|X)$ is assumed.

6.2.1 Class Imbalance

One simple assumption we can make about the connection between the distributions of the source and the target domains is that given the same class label, the conditional distributions of X are the same in the two domains. However, the class distributions may be different in the source and the target domains. Formally, we assume that $P_s(X|Y = y) = P_t(X|Y = y)$ for all $y \in \mathcal{Y}$, but $P_s(Y) \neq P_t(Y)$. This difference is referred to as the *class imbalance* problem in some work (Japkowicz and Stephen, 2002).

When this class imbalance assumption is made, the ratio $\frac{P_t(x,y)}{P_s(x,y)}$ that we derived in Equation (6.1) can be rewritten as follows:

$$\begin{aligned} \frac{P_t(x,y)}{P_s(x,y)} &= \frac{P_t(y)}{P_s(y)} \frac{P_t(x|y)}{P_s(x|y)} \\ &= \frac{P_t(y)}{P_s(y)}. \end{aligned}$$

Therefore, we only need to use $\frac{P_t(y)}{P_s(y)}$ to weight the instances. This approach has been explored in (Lin et al., 2002). Alternatively, we can re-sample the training instances from the source domain so that the re-sampled data roughly has the same class distribution as the target domain. In re-sampling methods, under-represented classes are over-sampled, and over-represented classes are under-sampled (Kubat and Matwin, 1997; Chawla et al., 2002; Zhu and Hovy, 2007).

For classification algorithms that directly model the probability distribution $P(Y|X)$ such as logistic regression classifiers, it can be shown theoretically that the estimated probability $P_s(y|x)$ can be transformed into $P_t(y|x)$ in the following way (Lin et al., 2002; Chan and Ng, 2005):

$$P_t(y|x) = \frac{r(y)P_s(y|x)}{\sum_{y' \in \mathcal{Y}} r(y')P_s(y'|x)},$$

where $r(y)$ is defined as

$$r(y) = \frac{P_t(y)}{P_s(y)}.$$

Now we can first estimate $P_s(y|x)$ from the source domain, and then derive $P_t(y|x)$ using $P_s(Y)$ and $P_t(Y)$.

For other classification algorithms that do not directly model $P(Y|X)$, such as naive Bayes classifiers and support vector machines, if $P(Y|X)$ can be obtained through careful calibration, the same trick can be applied. Chan and Ng (2006) applied this method to the domain adaptation problem in word sense disambiguation (WSD) using naive Bayes classifiers.

In practice, one needs to know the class distribution in the target domain in order to apply the methods described above. In some studies, it is assumed that this distribution is known a priori (Lin et al., 2002). However, in reality, we may not have this information. Chan and Ng (2005) proposed to use the EM algorithm to estimate the class distribution in the target domain.

6.2.2 Covariate Shift

Another assumption one can make about the connection between the source and the target domains is that given the same observation $X = x$, the conditional distributions of Y are the same in the two domains. However, the marginal distributions of X may be different in the source and the target domains. Formally, we assume that $P_s(Y|X = x) = P_t(Y|X = x)$ for all $x \in \mathcal{X}$, but $P_s(X) \neq P_t(X)$. This difference between the two domains is called *covariate shift* (Shimodaira, 2000).

At first glance, it may appear that covariate shift is not a problem. For classification, we are only interested in $P(Y|X)$. If $P_s(Y|X) = P_t(Y|X)$, why would the classifier learned from the source domain not perform well on the target domain even if $P_s(X) \neq P_t(X)$? Shimodaira (2000) showed that this covariate shift becomes a problem when *misspecified* models are used. Suppose we consider a parametric model family $\{P(Y|X, \theta)\}_{\theta \in \Theta}$ from which a model $P(Y|X, \theta^*)$ is selected to minimize the expected classification error. If none of the models in the model family can exactly match the true relation between X and Y, that is, there does not exist any $\theta \in \Theta$ such that $P(Y|X = x, \theta) = P(Y|X = x)$ for all $x \in \mathcal{X}$, then we say that we have a misspecified model family. The intuition of why covariate shift under model misspecification becomes a problem is

as follows. With a misspecified model family, the optimal model we select depends on $P(X)$, and if $P_t(X) \neq P_s(X)$, then the optimal model for the target domain will differ from that for the source domain. The intuitive is that the optimal model performs better in dense regions of X than in sparse regions of X, because the dense regions dominate the average classification error, which is what we want to minimize. If the dense regions of X are different in the source and the target domains, the optimal model for the source domain will no longer be optimal for the target domain.

Under covariate shift, the ratio $\frac{P_t(x,y)}{P_s(x,y)}$ that we derived in Equation (6.1) can be rewritten as follows:

$$\begin{aligned} \frac{P_t(x,y)}{P_s(x,y)} &= \frac{P_t(x)}{P_s(x)} \frac{P_t(y|x)}{P_s(y|x)} \\ &= \frac{P_t(x)}{P_s(x)}. \end{aligned}$$

We therefore want to weight each training instance with $\frac{P_t(x)}{P_s(x)}$.

Shimodaira (2000) first proposed to re-weight the log likelihood of each training instance (x, y) using $\frac{P_t(x)}{P_s(x)}$ in maximum likelihood estimation for covariate shift. It can be shown theoretically that if the support of $P_t(X)$ (the set of x's for which $P_t(X = x) > 0$) is contained in the support of $P_s(X)$, then the optimal model that maximizes this re-weighted log likelihood function asymptotically converges to the optimal model for the target domain.

A major challenge is how to estimate the ratio $\frac{P_t(x)}{P_s(x)}$ for each x in the training set. In some work, a principled method of using non-parametric kernel density estimation is explored (Shimodaira, 2000; Sugiyama and Müller, 2005). In some other work, it is proposed to transform this density ratio estimation into a problem of predicting whether an instance is from the source domain or from the target domain (Zadrozny, 2004; Bickel and Scheffer, 2007). Huang et al. (2007) transformed the problem into a kernel mean matching problem in a reproducing kernel Hilbert space. Bickel et al. (2007) proposed to learn this ratio together with the classification model parameters.

6.2.3 Change of Functional Relations

Both class imbalance and covariate shift simplify the difference between $P_s(X, Y)$ and $P_t(X, Y)$. It is still possible that $P_t(X|Y)$ differs from $P_s(X|Y)$ or $P_t(Y|X)$ differs from $P_s(Y|X)$.

We considered the case when $P_t(Y|X)$ differs from $P_s(Y|X)$, and proposed a heuristic method to remove "misleading" training instances from the source domain, where $P_s(y|x)$ is very different from $P_t(y|x)$ (Jiang and Zhai, 2007a). To discover these "misleading" training instances, some labeled data from the target domain is needed. This method therefore is only suitable for *supervised* domain adaptation.

6.3 Semi-Supervised Learning

If we ignore the domain difference, and treat the labeled source domain instances as labeled data and the unlabeled target domain instances as unlabeled data, then we are facing a semi-supervised learning (SSL) problem. We can then apply any SSL algorithms (Zhu, 2005; Chapelle et al., 2006) to the domain adaptation problem. The subtle difference between SSL and domain adaptation is that (1) the amount of labeled data in SSL is small but large in domain adaptation, and (2) the labeled data may be noisy in domain adaptation if we do not assume $P_s(Y|X = x) = P_t(Y|X = x)$ for all x, whereas in SSL the labeled data is all reliable.

There has been some work extending semi-supervised learning methods for domain adaptation. Dai et al. (2007a) proposed an EM-based algorithm for domain adaptation, which can be shown to be equivalent to a semi-supervised EM algorithm (Nigam et al., 2000) except that Dai et al. proposed to estimate the trade-off parameter between the labeled and the unlabeled data using the KL-divergence between the two domains. Jiang and Zhai (2007a) proposed to not only include weighted source domain instances but also weighted unlabeled target domain instances in training, which essentially combines instance weighting with bootstrapping. Xing et al. (2007) proposed a bridged refinement method for domain adaptation using label propagation on a nearest neighbor graph, which has resemblance to graph-based semi-supervised learning algorithms (Zhu, 2005;

Chapelle et al., 2006).

6.4 Change of Representation

As has been pointed out, the cause of the domain adaptation problem is the difference between $P_t(X, Y)$ and $P_s(X, Y)$. Note that while the representation of Y is fixed, the representation of X can change if we use different features. Such a change of representation of X can affect both the marginal distribution $P(X)$ and the conditional distribution $P(Y|X)$. One can assume that under some change of representation of X, $P_t(X, Y)$ and $P_s(X, Y)$ will become the same.

Formally, let $g : \mathcal{X} \rightarrow \mathcal{Z}$ denote a transformation function that transforms an observation x represented in the original form into another form $z = g(x) \in \mathcal{Z}$. Define variable Z and an induced distribution of Z that satisfies $P(z) = \sum_{x \in \mathcal{X}, g(x)=z} P(x)$. The joint distribution of Z and Y is then

$$P(z, y) \quad = \quad \sum_{x \in \mathcal{X}, g(x)=z} P(x, y).$$

If we can find a transformation function g so that under this transformation, we have $P_t(Z, Y) = P_s(Z, Y)$, then we no longer have the domain adaptation problem because the two domains have the same joint distribution of the observation and the class label. The optimal model $P(Y|Z, \theta^*)$ we learn to approximate $P_s(Y|Z)$ is still optimal for $P_t(Y|Z)$.

Note that with a change of representation, the entropy of Y conditional on Z is likely to increase from the entropy of Y conditional on X, because Z is usually a simpler representation of the observation than X, and thus encodes less information. In another word, the Bayes error rate usually increases under a change of representation. Therefore, the criteria for good transformation functions include not only the distance between the induced distributions $P_t(Z, Y)$ and $P_s(Z, Y)$ but also the amount of increment of the Bayes error rate.

Ben-David et al. (2007) first formally analyzed the effect of representation change for domain

adaptation. They proved a generalization bound for domain adaptation that is dependent on the distance between the induced $P_s(Z, Y)$ and $P_t(Z, Y)$.

A special and simple kind of transformation is feature subset selection. Satpal and Sarawagi (2007) proposed a feature subset selection method for domain adaptation, where the criterion for selecting features is to minimize an approximated distance function between the distributions in the two domains. Note that to measure the distance between $P_s(Z, Y)$ and $P_t(Z, Y)$, we still need class labels in the target domain. To solve this problem, in (Satpal and Sarawagi, 2007), predicted labels for the target domain instances are used.

Blitzer et al. (2006) proposed a structural correspondence learning (SCL) algorithm that makes use of the unlabeled data from the target domain to find a low-rank representation that is suitable for domain adaptation. It is empirically shown in (Ben-David et al., 2007) that the low-rank representation found by SCL indeed decreases the distance between the distributions in the two domains. However, SCL does not directly try to find a representation Z that minimizes the distance between $P_s(Z, Y)$ and $P_t(Z, Y)$. Instead, SCL tries to find a representation that works well for many related classification tasks for which labels are available in both the source and the target domains. The assumption is that if a representation Z gives good performance for the many related classification tasks in both domains, then Z is also a good representation for the main classification task we are interested in in both domains. The core algorithm in SCL is from (Ando and Zhang, 2005).

6.5 Bayesian Priors

Most of the work reviewed in the previous sections does not require labeled data from the target domain. In this section and the next section, we review two kinds of methods that work for supervised domain adaptation, i.e. when a small amount of labeled data from the target domain is available.

When we use the maximum a posterior (MAP) estimation approach for supervised learning, we

can encode some prior knowledge about the classification model into a Bayesian prior distribution $P(\theta)$, where θ is the model parameter. More specifically, instead of maximizing

$$\prod_{i=1}^{N} P(y_i|x_i;\theta),$$

we maximize

$$P(\theta)\prod_{i=1}^{N} P(y_i|x_i;\theta).$$

In domain adaptation, the prior knowledge can be drawn from the source domain. More specifically, we first construct a Bayesian prior $P(\theta|D_s)$, which is dependent on the labeled instances from the source domain. We then maximize the following objective function:

$$P(\theta|D_s)P(D_{t,l}|\theta) \;\; = \;\; P(\theta|D_s)\prod_{i=1}^{N_{t,l}} P(y_i^t|x_i^t;\theta).$$

Li and Bilmes (2007) proposed a general Bayesian divergence prior framework for domain adaptation. They then showed how this general prior can be instantiated for generative classifiers and discriminative classifiers. Chelba and Acero (2004) applied this kind of a Bayesian prior for the task of adapting a maximum entropy capitalizer across domains.

6.6 Multi-Task Learning

Multi-task learning, sometimes known as transfer learning, is a machine learning topic highly related to domain adaptation. The original definition of multi-task learning considers a different setting than domain adaptation. In multi-task learning, there is a single distribution of the observation, i.e. a single $P(X)$. There are, however, a number of different output variables $Y_1, Y_2, \ldots, Y_M$, corresponding to M different tasks. Therefore, there are M different joint distributions $\{P(X, Y_k)\}_{k=1}^{M}$. Note that the class label sets are different for these M different tasks. We as-

sume that these different tasks are related. When learning M conditional models $\{P(Y_k|X, \theta_k)\}_{k=1}^{M}$ for the M tasks, we impose a common component shared by $\{\theta_k\}_{k=1}^{M}$. There have been a number of studies on multi-task learning (Caruana, 1997; Ben-David and Schuller, 2003; Micchelli and Pontil, 2005; Xue et al., 2007).

Strictly speaking, domain adaptation is a different problem than multi-task learning because we have only a single task but different domains. However, domain adaptation can be treated as a special case of multi-task learning, where we have two tasks, one on the source domain and the other on the target domain, and the class label sets of these two tasks are the same. If we have some labeled data from the target domain, we can then directly apply some existing multi-task learning algorithm.

Indeed, some domain adaptation methods proposed recently are essentially multi-task learning algorithms. Daumé III (2007) proposed a simple method for domain adaptation based on feature duplications. The idea is to make a domain-specific copy of the original features for each domain. An instance from domain k is then represented by both the original features and the features specific to domain k. It can be shown that when linear classification algorithms are used, this feature duplication based method is equivalent to decomposing the model parameter θ_k for domain k into $\theta_c + \theta'_k$, where θ_c is shared by all domains. This formulation then is very similar to the regularized multi-task learning method proposed by Evgeniou and Pontil (2004). Similarly, we proposed a two-stage domain adaptation method, where in the first generalization stage, labeled instances from K different source training domains are used together to train K different models, but these models share a common component, and this common model component only applies to a subset of features that are considered generalizable across domains (Jiang and Zhai, 2007b) .

6.7 Ensemble Methods

In previous sections, only learning algorithms that return single classification models are considered. Ensemble methods are a type of learning algorithms that combine a set of models to construct

a complex classifier for a classification problem. Ensemble methods include bagging, boosting, mixture of experts, etc. There has been some work using ensemble methods for domain adaptation.

One line of work uses mixture models. It can be assumed that there are a number of different component distributions $\{P^{(k)}(X,Y)\}_{k=1}^{K}$, each of which modeled by a simple model. The distribution of X and Y in either the source domain or the target domain is then a mixture of these component distributions. The source and the target domains are related because they share some of these component distributions. However, the mixture coefficients are different in the two domains, making the overall distributions different.

Daumé III and Marcu (2006) proposed a mixture model for domain adaptation, in which three mixture components are assumed, one shared by both the source and the target domains, one specific to the source domain, and one specific to the target domain. Labeled data from both the source and the target domains is needed to learn this three-component mixture model using the conditional expectation maximization (CEM) algorithm. Storkey and Sugiyama (2007) considered a more general mixture model in which the source and the target domains share more than one mixture components. However, they did not assume any target domain specific component, and as a result, no labeled data from the target domain is needed. The mixture model is learned using the expectation maximization (EM) algorithm.

Boosting is a general ensemble method that combines multiple weak learners to form a complex and effective classifier. Dai et al. (2007b) proposed to modify the widely-used *AdaBoost* algorithm to address the domain adaptation problem. With some labeled data from the target domain, the idea here is to put more weight on mistakenly classified target domain instances but less weight on mistakenly classified source domain instances in each iteration, because the goal is to improve the performance of the final classifier on the target domain only.

Chapter 7

Conclusions

In this chapter, we summarize the findings of this thesis and point out some future research directions.

7.1 Summary

Supervised statistical machine learning is now an essential tool for solving many natural language processing problems. Standard supervised learning requires a large amount of correctly labeled training data, which is harder and harder to obtain nowadays given the various domains in which text analysis is needed. For example, we have texts in personal emails, Web logs, scientific literature, legal documents, etc. to analyze, and these different domains certainly contain different text properties. Therefore, there is a great need in natural language processing to exploit existing annotated corpora from limited domains, usually the news domain, and adapt classifiers trained on these domains to new domains.

In this thesis, we conducted an in-depth study of the domain adaptation problem in natural language processing. We first analyzed the problem from different angles. Observing that the essential cause for domain difference is the difference in the joint distribution of the observed instance and the class label, we directly looked at how standard empirical risk minimization framework should to be modified to account for this distributional domain difference. The analysis naturally leads to an instance weighting approach to domain adaptation. However, setting the instance weights is a challenging problem. We therefore proposed several heuristic ways to estimate the instance weights. Evaluation of the instance weighting framework on several real data sets shows that

instance weighting is effective in reducing the noise in the source domain caused by irrelevant instances.

We also analyzed the domain adaptation problem from the perspective of the difference in the classification functions for the two domains. This analysis leads to a feature selection framework for domain adaptation. In this framework, we consider a generalization stage where we identify features generalizable to both domains and an adaptation stage where we learn new characteristics of the target domain. The feature selection framework can also be generalized to the situation when there are multiple source domains from which we can adapt a classifier. Our experiment results show that the feature selection framework improves over standard supervised learning where domain difference is not considered.

We also studied different types of domain difference and their associations with different domain adaptation techniques. We constructed perturbed real data sets in order to evaluate the associations. The experiment results confirmed our hypothesis that different domain adaptation types require different techniques.

This thesis makes the following contributions to the understanding of the domain adaptation problem:

- **A formal analysis of domain adaptation**

  We provided a formal analysis of the domain adaptation problem, and the analysis is from a number of different angles. Such an analysis helps us better understand domain adaptation, finding the essential causes of domain difference in order to design solutions accordingly, and identifying different types of domain difference.

- **Two general solutions to domain adaptation**

  We proposed two general frameworks to address domain adaptation, both shown to be effective. While instance weighting for a similar "sample selection bias" problem has also been explored by other people, to the best of our knowledge, we first proposed to weigh instances not only based on the difference in $P(X)$ but also based on the difference in $P(Y|X)$. Our

experiment results show that for many natural language processing problems, the weighting factor based on $P(Y|X)$ is much more useful than the factor based on $P(X)$. Another major difference between our work and other existing work on instance weighting and feature selection is that we propose to combine instance weighting or feature selection with semi-supervised learning. Our experiment results show that including semi-supervised learning into the general frameworks can indeed further improve the performance.

- **Domain difference types**

  Existing work on domain adaptation treats all domain difference the same. We observe that there are different types of domain difference, which need different domain adaptation techniques. We constructed perturbed data sets to allow controlled study of individual domain difference types. These constructed data sets are also better benchmark data sets for comparison of different domain adaptation algorithms because they each contain a single type of domain difference.

7.2 Future Directions

In the future, we plan to further optimize the current solutions to domain adaptation as well as to study a few new directions related to domain adaptation.

- **Global optimization of a unified framework**

  Currently, we have not found a good solution to globally optimize the instance weighting framework in order to combine all the useful instance weighting techniques and generate the best performance for a given problem. Preliminary experiment results showed that combining the techniques in a straightforward manner is not the best, suggesting that further study is needed. Furthermore, the instance weighting framework and the feature selection framework can also be unified into a single framework, but we will again encounter the challenge

of setting the various parameters to be globally optimal. We therefore plan to extend our current work along this direction.

- **Predict domain difference types**

  Although in Chapter 5 we found that different types of domain difference require different domain adaptation techniques, how to predict domain difference types without a large amount of labels in the target domain still remains a challenging problem. Our experiments suggest that it may be possible to choose appropriate domain adaptation techniques based only on changes of feature frequencies. In the future, we plan to further study prediction of domain difference types along this direction. Such predictions should also be combined with global optimization of the unified framework as discussed above, because the various instance weighting and feature selection parameters essentially determine the domain adaptation techniques employed for a particular problem.

- **Exploiting existing knowledge bases**

  Our findings show that it is still important to have some labeled instances from the target domain. These instances can help us identify the difference between the source and the target domains. However, obtaining labels from humans is expensive. For some natural language processing problems in special domains, knowledge about the new domain may already exist in some knowledge bases and in formats different from labeled data. For example, for entity and relation extraction on biomedical literature, there exist many databases with information such as gene names and protein-protein interactions stored. Such information has been extracted by humans from biomedical literature, but what we can get from the databases is not labeled text but rather individual entities or relations taken out of context. However, such information is still very useful for obtaining knowledge about the new domains. For example, a gene list for a new organism we want to study may help us filter out morphological and orthographic features that are only useful to our training organisms. How to exploit such information in knowledge bases is an interesting and challenging research question.

- **Human interaction**

  As we discussed above, obtaining labeled instances from humans in order to obtain domain knowledge can be expensive. It is also a waste of human efforts as humans can provide more useful information than merely labels. Another interesting direction to explore is how to obtain domain knowledge through human interactions. For example, is it possible to obtain domain-specific features directly from expert users?

References

Rie Ando and Tong Zhang. A framework for learning predictive structure from multiple tasks and unlabeled data. *Journal of Machine Learning Research*, 6:1817–1853, November 2005.

Shai Ben-David and Reba Schuller. Exploiting task relatedness for multiple task learning. In *Proceedings of the 16th Annual Conference on Learning Theory*, Washington D.C., USA, August 2003.

Shai Ben-David, John Blitzer, Koby Crammer, and Fernando Pereira. Analysis of representations for domain adaptation. In B. Schölkopf, J. Platt, and T. Hoffman, editors, *Advances in Neural Information Processing Systems 19*, pages 137–144. MIT Press, Cambridge, Massachusetts, USA, 2007.

James C. Bezdek and Richard J. Hathaway. Some notes on alternating optimization. In *Proceedings of the 2002 AFSS International Conference on Fuzzy Systems*, pages 288–300, 2002.

Steffen Bickel and Tobias Scheffer. Dirichlet-enhanced spam filtering based on biased samples. In B. Schölkopf, J. Platt, and T. Hoffman, editors, *Advances in Neural Information Processing Systems 19*, pages 161–168. MIT Press, Cambridge, Massachusetts, USA, 2007.

Steffen Bickel, Michael Brückner, and Tobias Scheffer. Discriminative learning for differing training and test distributions. In *Proceedings of the 24th Annual International Conference on Machine Learning*, pages 81–88, Corvallis, Oregon, USA, June 2007.

John Blitzer, Ryan McDonald, and Fernando Pereira. Domain adaptation with structural correspondence learning. In *Proceedings of the 2006 Conference on Empirical Methods in Natural Language Processing*, pages 120–128, Sydney, Australia, July 2006.

John Blitzer, Mark Dredze, and Fernando Pereira. Biographies, bollywood, boom-boxes and blenders: Domain adaptation for sentiment classification. In *Proceedings of the 45th Annual Meeting of the Association for Computational Linguistics*, pages 440–447, Prague, Czech Republic, June 2007.

John Blitzer, Koby Crammer, Alex Kulesza, Fernando Pereira, and Jennifer Wortman. Learning bounds for domain adaptation. In J.C. Platt, D. Koller, Y. Singer, and S. Roweis, editors, *Advances in Neural Information Processing Systems 20*, pages 129–136. MIT Press, Cambridge, Massachusetts, USA, 2008.

Rich Caruana. Multitask learning. *Machine Learning*, 28(1):41–75, July 1997.

Yee Seng Chan and Hwee Tou Ng. Estimating class priors in domain adaptation for word sense disambiguation. In *Proceedings of the 21st International Conference on Computational Linguistics and 44th Annual Meeting of the Association for Computational Linguistics*, pages 89–96, Sydney, Australia, July 2006.

Yee Seng Chan and Hwee Tou Ng. Word sense disambiguation with distribution estimation. In *Proceedings of the 19th International Joint Conference on Artificial Intelligence*, pages 1010–1015, Edingurgh, Scotland, July 2005.

Olivier Chapelle, Bernhard Schölkopf, and Alexander Zien, editors. *Semi-Supervised Learning*. MIT Press, 2006.

Nitesh V. Chawla, Kevin W. Bowyer, Lawrence O. Hall, and W. Philip Kegelmeyer. SMOTE: Synthetic minority over-sampling technique. *Journal of Artificial Intelligence Research*, 16: 321–357, June 2002.

Ciprian Chelba and Alex Acero. Adaptation of maximum entropy capitalizer: Little data can help a lot. In *Proceedings of the 2004 Conference on Empirical Methods in Natural Language Processing*, pages 285–292, Barcelona, Spain, July 2004.

Michael Collins and Yoram Singer. Unsupervised models for named entity classification. In *Proceedings of the Joint SIGDAT Conference on Empirical Methods in Natural Language Processing and Very Large Corpora*, pages 100–110, College Park, Maryland, USA, 1999.

Wenyuan Dai, Gui-Rong Xue, Qiang Yang, and Yong Yu. Transferring naive bayes classifiers for text classification. In *Proceedings of the 22nd AAAI Conference on Artificial Intelligence*, pages 540–545, Vancouver, British Columbia, Canada, July 2007a.

Wenyuan Dai, Qiang Yang, Gui-Rong Xue, and Yong Yu. Boosting for transfer learning. In *Proceedings of the 24th Annual International Conference on Machine Learning*, pages 193–200, Corvallis, Oregon, USA, June 2007b.

Hal Daumé III. Frustratingly easy domain adaptation. In *Proceedings of the 45th Annual Meeting of the Association for Computational Linguistics*, pages 256–263, Prague, Czech Republic, June 2007.

Hal Daumé III and Daniel Marcu. Domain adaptation for statistical classifiers. *Journal of Artificial Intelligence Research*, 26:101–126, May 2006.

Theodoros Evgeniou and Massimiliano Pontil. Regularized multi-task learning. In *Proceedings of the 10th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 109–117, Seattle, Washington, USA, August 2004.

Jenny Finkel, Shipra Dingare, Chirstopher D. Manning, Malvina Nissim, Beatrice Alex, and Claire Grover. Exploring the boundaries: Gene and protein identification in biomedical text. *BMC Bioinformatics*, 6(Suppl 1):S5, May 2005.

Yves Grandvalet and Joshua Bengio. Semi-supervised learning by entropy minimization. In Lawrence K. Saul, Yair Weiss, and Léon Bottou, editors, *Advances in Neural Information Processing Systems 17*, pages 529–536. MIT Press, Cambridge, MA, 2005.

Sariel Har-Peled, Dan Roth, and Dav Zimak. Constraint classification for multiclass classification and ranking. In *Advances in Neural Information Processing Systems 15*, pages 785–792. MIT Press, Cambridge, MA, 2003.

James J. Heckman. Sample selection bias as a specification error. *Econometrica*, 47(1):153–161, January 1979.

Chih-Wei Hsu and Chih-Jen Lin. A comparison of methods for multiclass support vector machines. *IEEE Transactions on Neural Networks*, 13(2):415–425, March 2002.

Jiayuan Huang, Alexander J. Smola, Arthur Gretton, Karsten M. Borgwardt, and Bernhard Schölkopf. Correcting sample selection bias by unlabeled data. In B. Schölkopf, J. Platt, and T. Hoffman, editors, *Advances in Neural Information Processing Systems 19*, pages 601–608. MIT Press, Cambridge, MA, 2007.

Nathalie Japkowicz and Shaju Stephen. The class imbalance problem: A systematic study. *Intelligent Data Analysis*, 6(5):429–450, November 2002.

Jing Jiang and ChengXiang Zhai. Instance weighting for domain adaptation in NLP. In *Proceedings of the 45th Annual Meeting of the Association for Computational Linguistics*, pages 254–271, Prague, Czech Republic, June 2007a.

Jing Jiang and ChengXiang Zhai. A two-stage approach to domain adaptation for statistical classifiers. In *Proceedings of the ACM 16th Conference on Information and Knowledge Management*, 2007b. to appear.

Feng Jiao, Shaojun Wang, Chi-Hoon Lee, Russell Greiner, and Dale Schuurmans. Semi-supervised conditional random fields for improved sequence segmentation and labeling. In *Proceedings of the 21st International Conference on Computational Linguistics and 44th Annual Meeting of the Association for Computational Linguistics*, pages 209–216, Sydney, Australia, July 2006.

Miroslav Kubat and Stan Matwin. Addressing the curse of imbalanced training sets: One-sided selection. In *Proceedings of the 14th Annual International Conference on Machine Learning*, pages 179–186, Nashville, Tennessee, USA, July 1997.

Seth Kulick, Ann Bies, Mark Liberman, Mark Mandel, Ryan McDonald, Martha Palmer, Andrew Schein, and Lyle Ungar. Integrated annotation for biomedical information extraction. In *Proceedings of the HLT-NAACL Workshop on Linking Biological Literature, Ontologies and Databases*, pages 61–68, Boston, Massachusetts, USA, May 2004.

Xiao Li and Jeff Bilmes. A Bayesian divergence prior for classifier adaptation. In *Proceedings of the 11th International Conference on Artificial Intelligence and Statistics*, San Juan, Puerto Rico, March 2007.

Yi Lin, Yoonkyung Lee, and Grace Wahba. Support vector machines for classification in nonstandard situations. *Machine Learning*, 46(1–3):191–202, January 2002.

Chris Manning and Hinrich Schütze. *Foundations of Statistical Natural Language Processing*. MIT Press, Cambridge, Massachusetts, USA, 1999.

Mitchell P. Marcus, Beatrice Santorini, and Mary Ann Marcinkiewicz. Building a large annotated corpus of english: The Penn Treebank. *Computational Linguistics*, 19(2):313–330, June 1993.

David McClosky, Eugene Charniak, and Mark Johnson. Reranking and self-training for parser adaptation. In *Proceedings of the 21st International Conference on Computational Linguistics and 44th Annual Meeting of the Association for Computational Linguistics*, pages 337–344, Sidney, Australia, July 2006.

Charles A. Micchelli and Massimiliano Pontil. Kernels for multi-task learning. In Lawrence K. Saul, Yair Weiss, and Léon Bottou, editors, *Advances in Neural Information Processing Systems 17*, pages 921–928. MIT Press, Cambridge, Massachusetts, USA, 2005.

Kamal Nigam, Andrew K. McCallum, Sebastian Thrun, and Tom Mitchell. Text classification from labeled and unlabeled documents using EM. *Machine Learning*, 39(2–3):103–134, May 2000.

Martha Palmer, Dan Gildea, and Paul Kingsbury. The Proposition Bank: A corpus annotated with semantic roles. *Computational Linguistics*, 31(1):71–105, March 2005.

Ellen Riloff and Rosie Jones. Learning dictionaries for information extraction by multi-level bootstrapping. In *Proceedings of 16th National Conference on Artificial Intelligence*, pages 474–479, Orlando, Florida, USA, July 1999.

Sandeepkumar Satpal and Sunita Sarawagi. Domain adaptation of conditional probability models via feature subsetting. In *Proceedings of the 11th European Conference on Principles and Practice of Knowledge Discovery in Databases*, pages 224–235, Warsaw, Poland, September 2007.

Hidetoshi Shimodaira. Improving predictive inference under covariate shift by weighting the log-likelihood function. *Journal of Statistical Planning and Inference*, 90(2):227–244, October 2000.

Amos J. Storkey and Masashi Sugiyama. Mixture regression for covariate shift. In B. Schölkopf, J. Platt, and T. Hoffman, editors, *Advances in Neural Information Processing Systems 19*, pages 1337–1344. MIT Press, Cambridge, Massachusetts, USA, 2007.

Masashi Sugiyama and Klaus-Robert Müller. Input-dependent estimation of generalization error under covariate shift. *Statistics & Decisions*, 23(4):249–279, 2005.

Vladimir N. Vapnik. *The Nature of Statistical Learning Theory*. Springer, second edition, 1999.

Vldimir N. Vapnik. *Statistical Learning Theory*. Wiley, 1998.

Marc Vilain, Jennifer Su, and Suzi Lubar. Entity extraction is a boring solved problem—or is it? In *Proceedings of the Human Language Technology Conference of the North American Chapter of the Association for Computational Linguistics*, pages 181–184, Rochester, New York, USA, April 2007.

Dikan Xing, Wenyuan Dai, Gui-Rong Xue, and Yong Yu. Bridged refinement for transfer learning. In *Proceedings of the 11th European Conference on Principles and Practice of Knowledge Discovery in Databases*, pages 324–335, Warsaw, Poland, September 2007.

Ya Xue, Xuejun Liao, Lawrence Carin, and Balaji Krishnapuram. Multi-task learning for classification with Dirichlet process priors. *Journal of Machine Learning Research*, 8:35–63, May 2007.

Bianca Zadrozny. Learning and evaluating classifiers under sample selection bias. In *Proceedings of the 21th Annual International Conference on Machine Learning*, pages 114–121, Banff, Canada, July 2004.

Jingbo Zhu and Eduard Hovy. Active learning for word sense disambiguation with methods for addressing the class imbalance problem. In *Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, pages 783–790, Prague, Czech Republic, June 2007.

Xiaojin Zhu. Semi-supervised learning literature survey. Technical report, Carnegie Mellon University, 2005.

Author's Biography

Jing Jiang was born in Zhenjiang, China in 1979. She received her Bachelors and Masters degrees in Computer Science from Stanford University in 2002 and 2003, respectively. Following the completion of her PhD, she will join the Singapore Management University as an Assistant Professor of Information Systems in July 2008.