A Two-Stage Approach to Domain Adaptation for Statistical Classifiers

Jing Jiang
University of Illinois at Urbana-Champaign
201 N. Goodwin Ave.
Urbana, IL 21801
jiang4@cs.uiuc.edu

ChengXiang Zhai
University of Illinois at Urbana-Champaign
201 N. Goodwin Ave.
Urbana, IL 61801
czhai@cs.uiuc.edu

ABSTRACT

In this paper, we consider the problem of adapting statistical classifiers trained from some source domains where labeled examples are available to a target domain where no labeled example is available. One characteristic of such a *domain adaptation* problem is that the examples in the source domains and the target domain are known to follow different distributions. Thus a regular classification method would tend to overfit the source domains. We present a two-stage approach to domain adaptation, where at the first *generalization* stage, we look for a set of features generalizable across domains, and at the second *adaptation* stage, we pick up useful features specific to the target domain. Observing that the exact objective function is hard to optimize, we then propose a number of heuristics to approximately achieve the goal of generalization and adaptation. Our experiments on gene name recognition using a real data set show the effectiveness of our general framework and the heuristics.

Categories and Subject Descriptors

I.2.7 [**Artificial Intelligence**]: Natural Language Processing—*text analysis*; I.5.1 [**Pattern Recognition**]: Models—*statistical*

General Terms

algorithms, experimentation

Keywords

classification, domain adaptation, feature selection, semi-supervised learning, logistic regression

1. INTRODUCTION

Classification is a commonly used technique for knowledge management, and in particular, for managing textual information. Besides text categorization, many other text analysis tasks also rely on accurate classification algorithms. For example, information extraction problems such as named entity recognition and relation extraction from textual data are often cast into classification problems. Spam filtering is usually carried out with classifiers trained from known spam emails. Sentiment analysis such as classifying subjective vs. objective or positive vs. negative statements is also a typical classification problem.

In standard classification, the labeled data on which we train our classifier and the data on which we want to make predictions come from the same domain, and thus share the same distribution. However, in reality, we often face situations where we want to make predictions on a *target* domain, but the labeled examples we have are all from some different *source* domains, and the data in the target and the source domains are known to follow different distributions. For example, when training spam filters, we only have example spam and ham emails from some public resources, but ideally we would like to customize the spam filter for a particular user, especially if we have some (unlabeled) emails from the user's mailbox. When classifying positive and negative product reviews, we may have labeled reviews of some products, but would like to classify unlabeled reviews of some other product. Another example is gene name recognition in biomedical text mining. We may want to recognize gene names in literature about a "new organism" (e.g., honey bee), but we only have labeled training data for some well-studied "old organisms" (e.g., fly and yeast). In these situations, a regular classifier trained from the labeled examples in the source domains may not perform well on the unlabeled examples in the target domain because it may overfit the source domains. We thus face a domain adaptation problem—we need to adapt classifiers trained from one or several source domains to a target domain that is different from but related to the source domains.

Note that although in standard classification we also face the problem of overfitting, the problem would be alleviated if we have a lot of training data. In domain adaptation, however, the problem of overfitting is *inherent* because our training data is always a biased sample w.r.t. the target domain; the risk of overfitting would not disappear no matter how much training data we have in the source domains. Thus the problem of domain adaptation is fundamentally different from standard classification.

Formally, we define the general domain adaptation problem in the following way. First of all, we have a classification task that involves an input variable $\mathbf{X}$, represented by a p-dimensional feature vector, and a discrete output variable Y. Let $\mathcal{X}$ and $\mathcal{Y}$ denote the set of values $\mathbf{X}$ and Y may

CIKM'07, November 6–8, 2007, Lisboa, Portugal

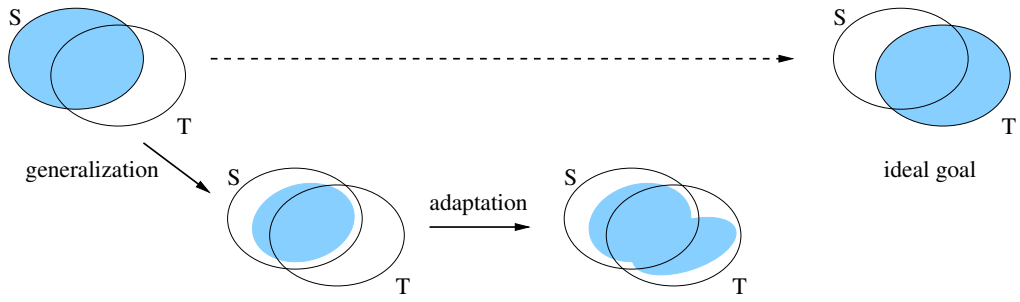

Figure 1: Two-Stage Domain Adaptation

take, respectively. The classification task is to predict the correct value of Y given any value of $\mathbf{X}$. Modeling the conditional probability distribution $p(y|\mathbf{X})$ appropriately is the key problem in order to achieve high classification accuracy; once $p(Y|\mathbf{X})$ is estimated, we can make predictions based on this distribution.

We further assume that there are K source domains $\{\mathcal{D}_k^s\}_{k=1}^K$ and a target domain $\mathcal{D}^t$, each having its only joint distribution $p(\mathbf{x}, y)$. Although $p(\mathbf{X})$, the marginal probability distribution of $\mathbf{X}$, may be very different in these different domains, we in general assume that the conditional probability distribution $p(Y|\mathbf{X})$ is pretty stable across domains. In fact, without this basic assumption, it is hard to learn from the source domains and transfer the knowledge to the target domain. However, assuming that different domains share the same $p(Y|\mathbf{X})$ does not mean that a model learned in a source domain to approximate $p(Y|\mathbf{X})$ can still make good predictions on a target domain, because the model learned from the source domain only provides good approximations to $p(Y|\mathbf{X} = \mathbf{x})$ for an example $\mathbf{x}$ in the source domain training set or near some source domain example, but in general we can expect the target domain to contain examples that do not occur in the source domain or are not close to any source domain example.

We assume that we have a set of labeled examples for each source domain, and a set of unlabeled examples for the target domain. Let $\{(\mathbf{x}_i^k, y_i^k)\}_{i=1}^{N_k}$ denote the examples for $\mathcal{D}_k^s$, and $\{\mathbf{x}_i^t\}_{i=1}^{N_t}$ the examples for $\mathcal{D}^t$. The goal of domain adaptation is then to estimate a good model for $p(Y|\mathbf{X})$ from the labeled examples such that this estimated model can make good predictions on the unlabeled examples in the target domain.

To solve the domain adaptation problem, intuitively, we want to make the most use of the labeled examples without overfitting the source domains. We also want to exploit the unlabeled examples in the target domain to possibly pick up patterns that do not exist in the source domains. Following this intuition, we propose to address the domain adaptation problem in two steps: (1) We first find the common features that are important for both the source and the target domains. (2) We then identify useful features that are specific to the target domain but cannot be learned from the source domains, possibly through semi-supervised learning. We thus propose a two-stage approach to domain adaptation, where at the first stage we identify a set of generalizable features for the classification task and learn the appropriate weights for these generalizable features, and at the second stage, we pick up useful features specific for the target domain by using some examples from the target domain with pseudo labels. The two-stage idea can be illustrated in an abstract way by Figure 1. For clarity, we assume that there is only one source domain. Here, the two ovals represent the source and the target domains, respectively. The shaded area represents the instances well-explained by the classification model. At the first stage, we want to shift the model towards the common part shared by the source and the target domains, and at the second stage, we want to expand the model with characteristics specific to the target domain.

We show that the two-stage approach can be formally defined as an optimization problem. However, since it is difficult to optimize the objective function directly, we propose two heuristics to approximately solve the problem. Our experiments on gene name recognition using a real data set show that this two-stage approach coupled with the heuristics performs better than standard supervised learning and a standard semi-supervised learning method, which we regard as baselines.

The rest of the paper is organized as follows. In Section 2, we introduce the two-stage approach to domain adaptation, formulate the problem as an optimization problem, and show two heuristics to approximately optimize the objective function. We then explain some implementation details of the two-stage approach in Section 3. We show our experiment results in Section 4. Finally we discuss related work in Section 5 and conclude in Section 6.

2. A TWO-STAGE APPROACH TO DOMAIN ADAPTATION

We will use the logistic regression classifier as a basis to present the proposed two-stage approach to domain adaptation [10]. Our approach to domain adaptation, however, is general to all linear classifiers, and therefore should also be applicable to other classifiers such as support vector machines and perceptrons.

We first briefly review the logistic regression model and discuss why it would overfit the source domains if we do not regularize the model appropriately.

2.1 Logistic Regression Models and Domain Overfitting

In logistic regression models (a.k.a. maximum entropy models) as well as other linear classification models, we assume that the input variable $\mathbf{X}$ is represented by a p-dimensional feature vector, and the classifier is a weight vector (in the case of binary classification) or a set of weight vectors (in the case of multi-class classification) that represent separating hyperplanes in the p-dimensional vector

space. Without loss of generality, in the rest of this paper, we consider only multi-class classification problems.

For logistic regression models, we assume that the conditional probability of a class label y given an example $\mathbf{x}$ is given by

$$p\left(y|\mathbf{x};\mathbf{w}\right) = \frac{1}{Z(\mathbf{x},\mathbf{w})}\exp(\mathbf{w}_y^T\mathbf{x}), \quad (1)$$

where

$$Z(\mathbf{x},\mathbf{w}) = \sum_{y'\in\mathcal{Y}}\exp(\mathbf{w}_{y'}^T\mathbf{x}). \quad (2)$$

Here $\mathbf{w}$ is a $p\times|\mathcal{Y}|$ weight matrix. $\mathbf{w}_y$ denotes the column vector of $\mathbf{w}$ corresponding to the class label y, and $\mathbf{w}_y^T$ denote the transpose of $\mathbf{w}_y$. $\mathbf{w}_y^T\mathbf{x}$ is therefore the inner product of $\mathbf{w}_y$ and $\mathbf{x}$.

In the standard formulation of supervised learning, we have a set of training examples together with their labels, denoted as $\{(\mathbf{x}_i,y_i)\}_{i=1}^N$, and we learn a weight matrix $\hat{\mathbf{w}}$ using regularized empirical risk minimization [14]. Specifically, if we use log likelihood as our loss function, we optimize the following objective function:

$$\hat{\mathbf{w}} = \arg\min_{\mathbf{w}}\left[\lambda\|\mathbf{w}\|^2 - \frac{1}{N}\sum_{i=1}^N\log p\left(y_i|\mathbf{x}_i;\mathbf{w}\right)\right], \quad (3)$$

where $\|\mathbf{w}\|^2=\sum_{y\in\mathcal{Y}}\|\mathbf{w}_y\|^2$, and λ is a regularization parameter that is manually set based on prior knowledge or empirically set by cross validation. The regularization term $\lambda\|\mathbf{w}\|^2$ is used to avoid overfitting the training data.

To see why we may overfit a domain with this kind of models, let us consider the simple case where all features are binary. Intuitively, the weight for a given feature f and a given class label y would be high if many examples in the training set with label y contain the feature f while many examples with labels different from y do not contain the feature f, i.e., feature f is highly correlated with y in the training domain. When we move to a target domain that is different from the training domain, if the weights learned from the training domain no longer give good predictions, a reasonable explanation is that some features that have high correlations with certain class labels in the training domain do not have as high correlations with the same class labels any more in the new domain, and vise versa. However, the training domain is still useful to us if there exist some features that have high correlations with the same class labels in both the training and the target domains. We call these features *generalizable* features. If in the training domain, these generalizable features are weaker than those useful domain-specific features, then the generalizable features may not get high weights because of the regularization term $\lambda\|\mathbf{w}\|^2$, which penalizes high weights. In another word, because of this regularization term, generalizable features are competing with domain-specific features for the weight mass. Ideally, if we want to learn a model from the training domain 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 2.2, we will show how this skewed regularization is imposed in the objective function.

2.2 Domain Generalization

The first stage of our two-stage approach to domain adaptation is therefore 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 then show how to identify the generalizable features.

We first explain some notation. To easily represent the separation of the generalizable features from other features in our mathematical formulation, we introduce a matrix A to represent the set of generalizable features. 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 generalizable feature space. 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 paper, 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 a set of h generalizable features. Here we assume that the number of generalizable features h is fixed.

We now show how to learn a set of appropriate weights for the generalizable features selected by A given a fixed A. In standard supervised learning, we learn a single weight matrix $\mathbf{w}$ for all features using all the training examples. When the training data falls into several domains, however, we cannot expect the optimal weights to be the same for all domains. We thus introduce K different weight matrices, $\{\mathbf{w}^k\}_{k=1}^K$, for the K training 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:

$$\mathbf{w}^k = A^T\mathbf{v}+\mathbf{u}^k, \quad (4)$$

where $\mathbf{v}$ is an $h\times|\mathcal{Y}|$ matrix shared by all domains, and $\mathbf{u}^k$ is domain-specific. We can think of $\mathbf{v}$ as essentially the weight matrix for the generalizable features.

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

$$\begin{aligned}&\left(\hat{\mathbf{v}}(A),\{\hat{\mathbf{u}}^k(A)\}\right)\\ =\ &\arg\min_{\mathbf{v},\{\mathbf{u}^k\}}\left[\lambda\left(\|\mathbf{v}\|^2+\lambda_s\sum_{k=1}^K\|\mathbf{u}^k\|^2\right)\right.\\ &\left.-\frac{1}{K}\sum_{k=1}^K\frac{1}{N_k}\sum_{i=1}^{N_k}\log p(y_i^k|\mathbf{x}_i^k;A^T\mathbf{v}+\mathbf{u}^k)\right]. \quad (5)\end{aligned}$$

The first term in the objective function is the regularization term, where different regularization parameters are put on

$\mathbf{v}$ and $\{\mathbf{u}^k\}$. The second term is the empirical risk (log likelihood of the training data). 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 simply set λ_s to be much larger than 1. With $\lambda_s \gg 1$, 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 examples strongly favor some large values of $\{\mathbf{u}^k\}$.

Note that in (5) we have made two modifications to the standard learning of logistic regression models: (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 matrix $\mathbf{v}$ for the generalizable features across all the domains, while allowing some slight domain variation captured by $\{\mathbf{u}^k\}$.

Next, 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 matrix learned from the training domains, as defined in (5). 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 over all possible examples in the target domain, which can be captured by the following objective function:

$$A^* = \arg\min_A \sum_{\mathbf{x}\in\mathcal{X}} \sum_{y\in\mathcal{Y}} -p_t(\mathbf{x}, y) \log p\left(y|\mathbf{z}; \hat{\mathbf{v}}(A)\right), \quad (6)$$

where $p_t(\mathbf{x}, y)$ denote the true joint probability of $\mathbf{x}$ and y in the target domain, and $\mathbf{z} = A\mathbf{x}$.

Equation (6) is the ideal criterion for choosing the optimal A. However, (6) is in practice infeasible to compute because (1) we do not know $p_t(\mathbf{x}, y)$, and (2) a brute force enumeration of 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^*.

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

In (6), 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:

$$\begin{aligned}&\left(\hat{A}, \hat{\mathbf{v}}, \{\hat{\mathbf{u}}^k\}\right) \\ =\ &\arg\min_{A,\mathbf{v},\{\mathbf{u}^k\}} \left[\lambda\left(\|\mathbf{v}\|^2 + \lambda_s \sum_{k=1}^{K} \|\mathbf{u}^k\|^2\right)\right. \\ &\left. -\frac{1}{K}\sum_{k=1}^{K}\frac{1}{N_k}\sum_{i=1}^{N_k} \log p\left(y_i^k|\mathbf{x}_i^k; A^T\mathbf{v} + \mathbf{u}^k\right)\right]. \quad (7)\end{aligned}$$

The $\hat{A}$ and $\hat{\mathbf{v}}$ chosen in this way form the final generalizable model we use for predictions on the target domain. Note that (7) is the same as (5) except that A is now free to change inside the objective function. In Section 3.1, 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 also show that the cross validation method is better than the joint optimization method.

2.2.2 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 domain as if it were a single training example. Thus, given a fixed A, we first learn the weight matrix $\hat{\mathbf{v}}(A)$ using all but one training domains, and then test the performance on the held-out training domain. We repeat this procedure for each held-out training 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:

$$\hat{A} = \arg\min_A \frac{1}{K}\sum_{k=1}^{K}\left[-\frac{1}{N_k}\sum_{i=1}^{N_k}\log p\left(y_i^k|\mathbf{z}_i^k; \hat{\mathbf{v}}(k, A)\right)\right], \quad (8)$$

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 $\mathcal{D}_k^s$. In another word, $\hat{\mathbf{v}}(k, A)$ is obtained by

$$\begin{aligned}&\left(\hat{\mathbf{v}}(k, A), \{\hat{\mathbf{u}}^{k'}(k, A)\}_{k'\neq k}\right) \\ =\ &\arg\min_{\mathbf{v},\{\mathbf{u}^{k'}\}_{k'\neq k}} \left[\lambda\left(\|\mathbf{v}\|^2 + \lambda_s \sum_{k'\neq k}\|\mathbf{u}^{k'}\|^2\right)\right. \\ &\left. -\frac{1}{K-1}\sum_{k'\neq k}\frac{1}{N_{k'}}\sum_{i=1}^{N_{k'}}\log p(y_i^{k'}|\mathbf{x}_i^{k'}; A^T\mathbf{v} + \mathbf{u}^{k'})\right] \quad (9)\end{aligned}$$

However, even with this approximation, in practice it is still infeasible to enumerate all possible A when optimizing (8). In Sect. 3.2, we will propose a heuristic way to approximate the optimization problem in (8).

2.3 Domain Adaptation

After the first stage of domain generalization, we will obtain an A, which represents the set of generalizable features learned from the source domains, as well as a set of weights 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. Since we only consider the situation where we do not have any labeled examples in the target domain, here we adopt bootstrapping, a commonly used semi-supervised learning method, to make use of the target domain examples [15]. More specifically, with the best generalizable model that we have learned from the source domains, we make predictions on the unlabeled examples in the target domain, choose the most confident m examples together with the pseudo labels, and include these labeled examples in our objective function to learn the weights.

Formally, let $\hat{A}$ be the optimal 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 examples in the target domain that have been predicted with the highest probabilities $p\left(\hat{y}_i^t|\mathbf{x}_i^t\right)$, where $\hat{y}_i^t$ is the predicted label of $\mathbf{x}_i^t$. We

learn weight matrices $\hat{\mathbf{v}}$ and $\hat{\mathbf{u}}^t$ by optimizing the following objective function:

$$\begin{aligned}&\left(\hat{\mathbf{v}}, \hat{\mathbf{u}}^t, \{\hat{\mathbf{u}}^k\}\right)\\ =\ &\underset{\mathbf{v},\mathbf{u}^t,\{\mathbf{u}^k\}}{\arg\min}\left[\lambda\left(\|\mathbf{v}\|^2+\lambda_s\sum_{i=1}^{K}\|\mathbf{u}^k\|^2+\lambda_t\|\mathbf{u}^t\|^2\right)\right.\\ &-\frac{1}{K+1}\left(\sum_{k=1}^{K}\frac{1}{N_k}\sum_{i=1}^{N_k}\log p\left(y_i^k|\mathbf{x}_i^k; A^T\mathbf{v}+\mathbf{u}^k\right)\right.\\ &\left.\left.+\frac{1}{m}\sum_{i=1}^{m}\log p\left(\hat{y}_i^t|\mathbf{x}_i^t; A^T\mathbf{v}+\mathbf{u}^t\right)\right)\right]. \qquad (10)\end{aligned}$$

Equation (10) is very similar to (5). The target domain is treated in the same way as all source domains in the objective function, except for the regularization parameter λ_t. In general, we want to set $\lambda_t \ll \lambda_s$ because our goal is exactly to learn the weights for the domain-specific features in the target domain. 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 $p\left(y|\mathbf{x}; \hat{\mathbf{w}}^t\right)$ as our final model to make predictions on the target domain.

3. IMPLEMENTATION DETAILS

In this section, we discuss some implementation details that have been left out in Section 2.

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

We first discuss how to solve the optimization problem in (7). The problem formulation is very similar to the optimization problem in [1], which can be solved by an alternating optimization procedure [3]. 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 (7) can then be rewritten as follows:

$$\begin{aligned}\left(\hat{A}, \hat{\mathbf{v}}, \{\hat{\mathbf{w}}^k\}\right) \ =\ &\underset{A,\mathbf{v},\{\mathbf{w}^k\}}{\arg\min}\left[\lambda\left(\|\mathbf{v}\|^2+\lambda_s\sum_{k=1}^{K}\|\mathbf{w}^k - A^T\mathbf{v}\|^2\right)\right.\\ &\left.-\frac{1}{K}\sum_{k=1}^{K}\frac{1}{N_k}\sum_{i=1}^{N_k}\log p\left(y_i^k|\mathbf{x}_i^k; \mathbf{w}^k\right)\right]. \qquad (11)\end{aligned}$$

When $\{\mathbf{w}^k\}_{k=1}^K$ are fixed, the second term in (11) is also fixed, while we can vary A and $\mathbf{v}$ to minimize the first term. When A is fixed, we can vary $\mathbf{v}$ and $\{\mathbf{u}^k\}_{k=1}^K$ to minimize (7), which is equivalent to varying $\mathbf{v}$ and $\{\mathbf{w}^k\}_{k=1}^K$ to minimize (11). 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 matrix trained from the labeled examples from $\mathcal{D}_k^s$.

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

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

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

$$\begin{aligned}&\left(\hat{\mathbf{v}}, \{\hat{\mathbf{u}}^k\}\right)\\ =\ &\underset{\mathbf{v},\{\mathbf{u}^k\}}{\arg\min}\left[\lambda\left(\|\mathbf{v}\|^2+\lambda_s\sum_{k=1}^{K}\|\mathbf{u}^k\|^2\right)\right.\\ &\left.-\frac{1}{K}\sum_{k=1}^{K}\frac{1}{N_k}\sum_{i=1}^{N_k}\log p\left(y_i^k|\mathbf{x}_i^k; A^T\mathbf{v}+\mathbf{u}^k\right)\right].\end{aligned}$$

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, the optimization problem defined in step 3 is similar to the standard training of logistic regression models. 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 [1] 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 step 3, the value of the objective function decreases, the alternating optimization procedure converges to a local minimum.

3.2 A Heuristic for Domain Cross Validation

We now discuss how to approximate the optimization problem defined in (8). First, let us consider a single component of (8):

$$-\frac{1}{N_k}\sum_{i=1}^{N_k}\log p\left(y_i^k|\mathbf{x}_i^k; \hat{\mathbf{v}}(k, A)\right). \qquad (12)$$

Recall that $\hat{\mathbf{v}}(k, A)$ is the weight matrix learned from all training examples except those from $\mathcal{D}_k^s$, with a fixed A. Consider another weight matrix $\hat{\beta}(k, A)$ defined as follows:

$$\begin{aligned}&\left(\hat{\beta}(k, A), \hat{\mathbf{u}}\right)\\ =\ &\underset{\beta,\mathbf{u}}{\arg\min}\left[\lambda\left(\|\beta\|^2+\lambda_s\|\mathbf{u}\|^2\right)\right.\\ &\left.-\frac{1}{N_k}\sum_{i=1}^{N_k}\log p(y_i^k|\mathbf{x}_i^k; A^T\beta+\mathbf{u})\right]. \qquad (13)\end{aligned}$$

In order to minimize (12), 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 (12). To measure the similarity between $\hat{\beta}(k, A)$ and $\hat{\mathbf{v}}(k, A)$, we can use the sum of the inner products between each pair of the corresponding column vectors of $\hat{\beta}(k, A)$ and $\hat{\mathbf{v}}(k, A)$. Formally, we introduce a scoring function $S(\beta, \mathbf{v})$ as follows:

$$S(\beta, \mathbf{v}) \ =\ \sum_{y\in\mathcal{Y}}\beta_y^T\mathbf{v}_y. \qquad (14)$$

The choice of this similarity measure can be justified as follows. The weight matrices β and $\mathbf{v}$ each represent a linear classifier in an h-dimensional feature space. We can roughly think of the column vector β_y (or $\mathbf{v}_y$) as what classifier β (or $\mathbf{v}$) regards as a canonical example that belongs to class

y. To measure how close these two classifiers are to each other, we can measure the pairwise similarity between the canonical examples of the two classifiers, where similarity is defined as the inner product between two vectors. It is important to note that we want to use inner product instead of cosine similarity because the larger the norm of a canonical example is, the more confident the classifier is about that canonical example. In another word, the length of the vector β_y roughly corresponds to our confidence that examples belonging to class y are along the direction of this vector.

Having understood the meaning of $S(\beta, \mathbf{v})$, we can now use the following optimization problem to approximate (8):

$$\hat{A} = \arg\max_{A} \sum_{k=1}^{K} S\Big(\hat{\beta}(k, A), \hat{\mathbf{v}}(k, A)\Big), \tag{15}$$

where $\hat{\beta}(k, A)$ and $\hat{\mathbf{v}}(k, A)$ are defined in (13) and (9), respectively.

We still have not addressed the problem that it is not feasible to enumerate all possible A. 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 matrix learned from the training examples in $\mathcal{D}_k^s$ without any feature selection. Formally,

$$\hat{\theta}^k = \arg\min_{\theta} \left[\lambda \|\theta\|^2 - \frac{1}{N_k} \sum_{i=1}^{N_k} \log p\left(y_i^k | \mathbf{x}_i^k; \theta\right) \right]. \tag{16}$$

Similarly, define $\hat{\theta}^{\bar{k}}$ as

$$\begin{aligned} \hat{\theta}^{\bar{k}} &= \arg\min_{\theta} \Bigg[\lambda \|\theta\|^2 \\ &\quad - \frac{1}{K-1} \sum_{k' \neq k} \frac{1}{N_{k'}} \sum_{i=1}^{N_{k'}} \log p\left(y_i^{k'} | \mathbf{x}_i^{k'}; \theta\right) \Bigg]. \end{aligned} \tag{17}$$

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

$$\hat{\beta}(k, A) \approx A\,\hat{\theta}^k, \tag{18}$$

$$\hat{\mathbf{v}}(k, A) \approx A\,\hat{\theta}^{\bar{k}}. \tag{19}$$

What (18) and (19) 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.

Let $f_1, \ldots, f_h$ be the set of features selected by A. Let $\theta_{y,f}$ denote the weight for class y and feature f in weight matrix θ. We then have

$$\begin{aligned} & \sum_{k=1}^{K} S\Big(\hat{\beta}(k, A), \hat{\mathbf{v}}(k, A)\Big) \\ \approx & \sum_{k=1}^{K} S\Big(A\hat{\theta}^k, A\hat{\theta}^{\bar{k}}\Big) \\ = & \sum_{k=1}^{K} \sum_{y \in \mathcal{Y}} \left(A\hat{\theta}_y^k\right)^T A\hat{\theta}_y^{\bar{k}} \\ = & \sum_{k=1}^{K} \sum_{y \in \mathcal{Y}} \left(\sum_{i=1}^{h} \hat{\theta}_{y,f_i}^k \hat{\theta}_{y,f_i}^{\bar{k}} \right) \\ = & \sum_{i=1}^{h} \left(\sum_{k=1}^{K} \sum_{y \in \mathcal{Y}} \hat{\theta}_{y,f_i}^k \hat{\theta}_{y,f_i}^{\bar{k}} \right). \end{aligned} \tag{20}$$

Now it is clear that we can maximize (20) by selecting h features that have the highest scores

$$\sum_{k=1}^{K} \sum_{y \in \mathcal{Y}} \hat{\theta}_{y,f}^k \hat{\theta}_{y,f}^{\bar{k}}. \tag{21}$$

3.3 Summary

To summarize the techniques we have discussed, we have shown that at the domain generalization stage, we have two heuristics for finding a good set of generalizable features, represented by the matrix A, together with a weight matrix $\mathbf{v}$. The first heuristic is a joint optimization method, which can be carried out using an alternating optimization procedure. The second heuristic is a domain cross validation method, which can be approximated by ranking the features based on (21) and selecting the top h features. Once the top h features are selected, we still optimize (5) to find the optimal $\hat{\mathbf{v}}$.

It is worth pointing out that although the first heuristic method may largely increase the computational complexity because of the number of iterations, the second heuristic method is not very expensive. For K source domains, the second heuristic method requires training $2K$ classifiers at the domain cross validation step and training a final classifier after the optimal A is found. Furthermore, although our domain adaptation method increases the computational complexity at the *training* stage, the computational complexity at the *testing* stage is not affected because the output of our classification method is simply a weight matrix, which is the same as in regular classification method.

4. EXPERIMENTS

In this section, we show our empirical evaluation of the two-stage domain adaptation approach with the proposed heuristics.

4.1 Data Set and Experiment Setup

We tested our method on the problem of recognizing gene and protein names from biomedical literature. The data set we used is from the BioCreAtIvE I challenge, Task 1B, and was processed as described in [11].[1] In particular, the data set contains three subsets, corresponding to three organisms, fly, mouse, and yeast. Thus we can naturally treat each

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

organism as an individual domain. Note that since the labels in this data set were not manually assigned, the data set is noisy and the absolute performance on this data set is lower than the state-of-the-art for gene recognition. 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 [8]. We use F1 as the primary performance measure, where F1 is defined as

$$\text{F1} = \frac{2 \times \text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}. \quad (22)$$

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.

Our domain adaptation method 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 used the χ^2 statistic measure to rank the features. We call this baseline method *BL*. We implemented the first stage of our method 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 (10) to learn a new model. Since our results showed that DA-2 is better than DA-1, in the adaptation stage, we only combined 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 λ_s to 10^6. This value was arbitrarily chosen to be sufficiently large. In DA-2-SSL, we set λ_t to 1. This value was also arbitrarily chosen to be sufficiently smaller than λ_s.

4.2 Domain Generalization

In Table 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 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 (5) to learn the common weights shared by all training domains is still better than ignoring the domain difference. Second, the optimal performance achieved by DA-2 is better than that of DA-1, which, in two out of the three cases, is better than that of 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 2, 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 source domain examples 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.

4.3 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 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 3, we show the comparison between BL-SSL, BL-SSL-2 and DA-2-SSL when m varies from 0 to 1000. We can make a number of observations from Figure 3. First, the performance of these semi-supervised learning methods does not always increase as m increases. Indeed, in semi-supervised learning, since the target examples added to the training set may not be labeled correctly, when more target examples 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 examples with pseudo labels, when we adapt to the target domain using these target examples, we can pick

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

Method	h	F+M⇒Y Precision	F+M⇒Y Recall	F+M⇒Y F1	M+Y⇒F Precision	M+Y⇒F Recall	M+Y⇒F F1	Y+F⇒M Precision	Y+F⇒M Recall	Y+F⇒M 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 2: Comparison between BL-SSL, BL-SSL-2, and DA-2-SSL

Method	m	F+M⇒Y Precision	F+M⇒Y Recall	F+M⇒Y F1	M+Y⇒F Precision	M+Y⇒F Recall	M+Y⇒F F1	Y+F⇒M Precision	Y+F⇒M Recall	Y+F⇒M 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**

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 examples 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 future research question is 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 sentences added to the training set, and the difference between the learning algorithms. In order to separate the two factors, we designed another diagnostic method, *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 3. 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 examples 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 suggests that it is important to find a good value of m.

5. RELATED WORK

The problem of domain adaptation has drawn a lot of attention recently. For text analysis and textual information management, people have proposed different methods to address the domain adaptation problem in the specific learning tasks they study [13, 6, 9, 5].

Recently there have also been a few studies that proposed principled methods to address the general domain adaptation problem. In most studies, it is assumed that there is a *single* source domain and a target domain. Daumé and Marcu proposed a model in which they assume that the data in each domain is generated from a mixture of a general distribution and a domain-specific distribution, and these distributions share a common prior [7]. With labeled examples from both the source domain and the target domain, they can jointly learn the three distributions as well as the classification models for each distribution. Li and Bilmes proposed a method for domain adaptation by imposing a Bayesian divergence prior learned from the source domain when training on the target domain [12].

Ando and Zhang [1] proposed a multi-task learning method for semi-supervised learning. Although their method was not designed for domain adaptation, later in [4, 2], the authors borrowed Ando and Zhang's method, and studied the domain adaptation problem from a feature representation perspective. They identify a generalizable feature representation across domains by creating many auxiliary problems and performing multi-task learning on these problems. Their assumption is that a good common representation for these auxiliary problems is also a good representation for the learning problem of interest. Our model looks very sim-

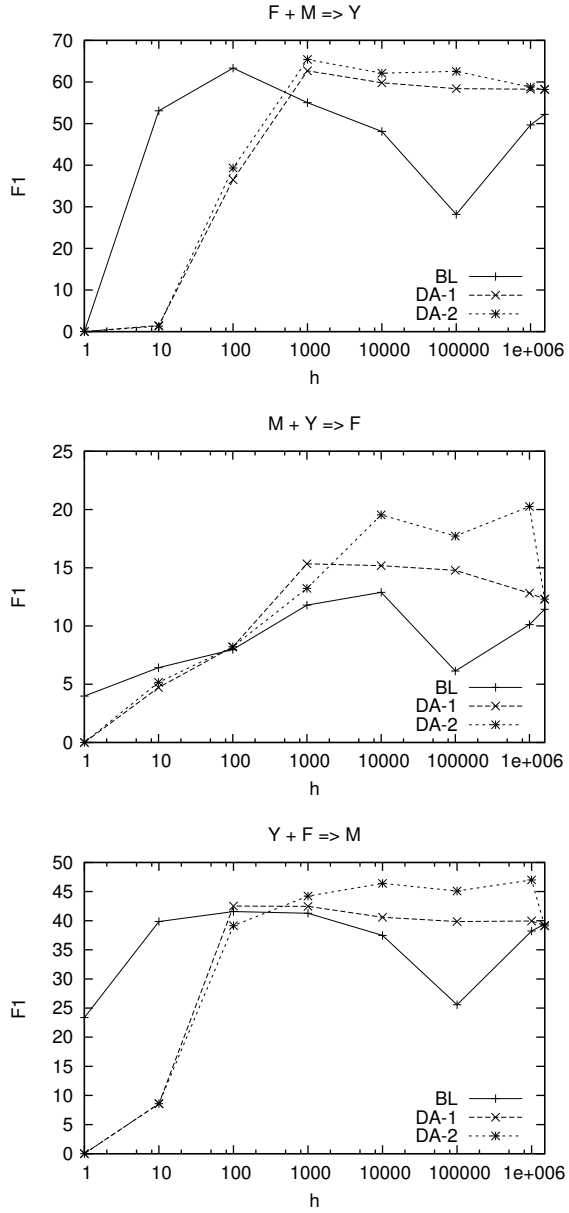

Figure 2: Comparison between BL, DA-1 and DA-2 as h Varies

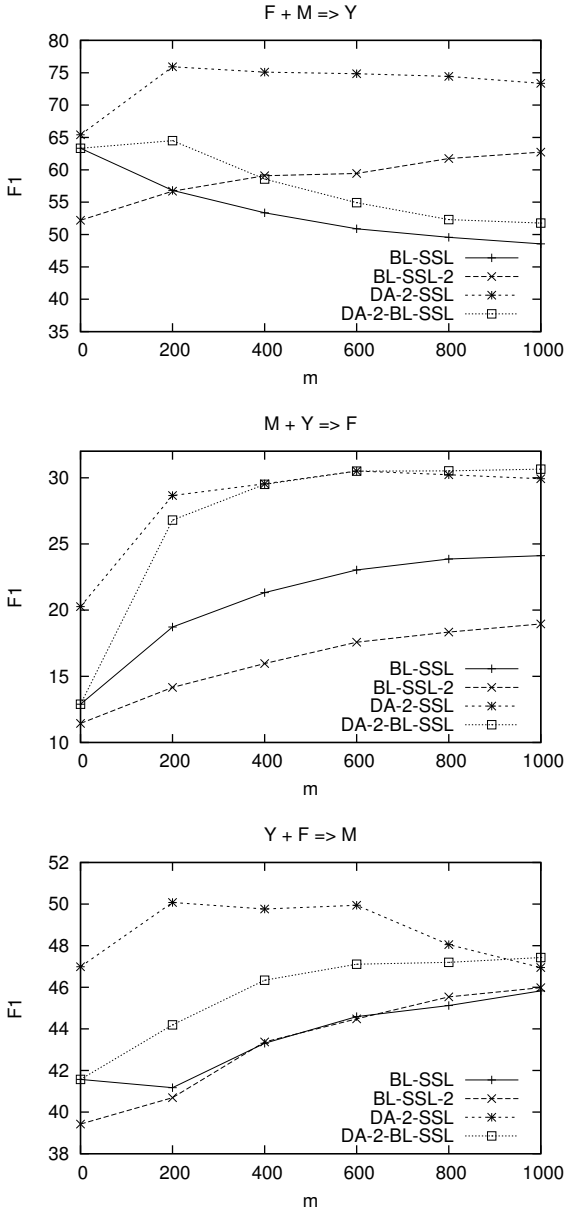

Figure 3: Comparison between BL-SSL, BL-SSL-2, DA-2-SSL and DA-2-BL-SSL as m Varies

ilar to the model in the aforementioned work. However, in contrast, our model is a special case of multi-task learning where both the lower-dimensional feature representation and the weights for these features are shared by the different tasks, because we are essentially learning the same task.

6. CONCLUSIONS AND FUTURE WORK

Classification is an important technique for many knowledge management applications, but we often lack training data in the domain in which we want to make predictions. If we have training data from some related domains, we can adapt the classifier learned from these related source domains to our target domain. In this work, we proposed a two-stage approach to domain adaptation for statistical classifiers where at the first stage we try to find a generalizable feature representation across different domains as well as appropriate weights for these features, and at the second stage we try to pick up features specifically useful for the target domain by employing semi-supervised learning. Our experiment results from the gene name recognition task using a real data set show that our proposed two-stage approach coupled with the heuristics we have proposed is effective for domain adaptation. Our two-stage approach is a general method so it should be applicable to other tasks that involve domain adaptation for classifiers.

In our method, there are several parameters that need to be set. The regularization parameter λ is similar to what is used in regular supervised learning, and therefore can be tuned by regular cross validation. For λ_s, we set it to 10^6, but we have found it not very sensitive as long as $\lambda_s \gg 1$. For λ_t, currently we set it to 1. Since this parameter controls how much we want to rely on the pseudo-labeled target domain examples, in our future work, we will vary the value and study its impact to the performance. The parameter h defines the number of generalizable features to use. In our experiments, setting h to a relatively large value is safe. However, we need experiments on more data sets in order to draw any conclusion. On the other hand, presumably we can use domain cross validation to select a good h. We also leave this study to our future work. The parameter m defines the number of pseudo labeled target examples to use. As in regular bootstrapping method, setting this number can be tricky. In the future, we will also study automatic ways of choosing a safe range for m.

Our current framework also has the limitation that in order to identify the generalizable features, we need to have at least two domains in the training data. In the future, we plan to remove this constraint, and study whether target examples with pseudo labels can also help us identify the generalizable features.

7. ACKNOWLEDGMENTS

This research is supported in part by MIAS, a DHS Institute of Discrete Science Center for Multimodal Information Access and Synthesis, and by National Science Foundation under award number 0425852. We thank the anonymous reviewers for their invaluable comments.

8. REFERENCES

[1] R. K. Ando and T. Zhang. A framework for learning predictive structures from multiple tasks and unlabeled data. *Journal of Machine Learning Research*, 6:1817–1853, 2005.

[2] S. Ben-David, J. Blitzer, K. Crammer, and F. Pereira. Analysis of representations for domain adaptation. In *Advances in Neural Information Processing Systems 19*, 2007.

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

[4] J. Blitzer, R. McDonald, and F. Pereira. Domain adaptation with structural correspondence learning. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pages 120–128, 2006.

[5] Y. S. Chan and H. T. 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, 2006.

[6] C. Chelba and A. Acero. Adaptation of maximum entropy capitalizer: Little data can help a lot. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pages 285–292, 2004.

[7] H. Daumé III and D. Marcu. Domain adaptation for statistical classifiers. *Journal of Artificial Intelligence Research*, 26:101–126, 2006.

[8] J. Finkel, S. Dingare, C. D. Manning, M. Nissim, B. Alex, and C. Grover. Exploring the boundaries: gene and protein identification in biomedical text. *BMC Bioinformatics*, 6(Suppl 1):S5, 2005.

[9] R. Florian, H. Hassan, A. Ittycheriah, H. Jing, N. Kambhatla, X. Luo, N. Nicolov, and S. Roukos. A statistical model for multilingual entity detection and tracking. In *Proceedings of the Human Language Technology Conference of the North American Chapter of the Association for Computational Linguistics*, pages 1–8, 2004.

[10] D. W. Hosmer and S. Lemeshow. *Applied Logistic Regression*. Wiley Series in Probability and Statistics. John Wiley & Sons, Inc., 2000.

[11] J. Jiang and C. Zhai. Exploiting domain structure for named entity recognition. In *Proceedings of The Human Language Technology Conference of the North American Chapter of the Association for Computational Linguistics*, pages 74–81, 2006.

[12] X. Li and J. Bilmes. A Bayesian divergence prior for classifier adaptation. In *Proceedings of the 11th International Conference on Artificial Intelligence and Statistics*, 2007.

[13] B. Roark and M. Bacchiani. Supervised and unsupervised PCFG adaptatin to novel domains. In *Proceedings of the Human Language Technology Conference of the North American Chapter of the Association for Computational Linguistics*, pages 126–133, 2003.

[14] V. N. Vapnik. *The Nature of Statistical Learning Theory*. Springer-Verlag New York, Inc., 1995.

[15] X. Zhu. Semi-supervised learning literature survey. Technical Report 1530, University of Wisconsin, 2005.