2009 Ninth IEEE International Conference on Data Mining

Extending Semi-supervised Learning Methods for Inductive Transfer Learning

Yuan Shi
School of Software, Sun Yat-sen University
Guangzhou, China
sycaszsu@gmail.com

Zhenzhong Lan
School of Software, Sun Yat-sen University
Guangzhou, China
lzzsysu@gmail.com

Wei Liu
Department of Computer Science, Nanjing University
Nanjing, China
lwbiosoft@gmail.com

Wei Bi
Department of Computer Science, Sun Yat-sen University, Guangzhou, China
biwei@mail2.sysu.edu.cn

***Abstract*—Inductive transfer learning and semi-supervised learning are two different branches of machine learning. The former tries to reuse knowledge in labeled out-of-domain instances while the later attempts to exploit the usefulness of unlabeled in-domain instances. In this paper, we bridge the two branches by pointing out that many semi-supervised learning methods can be extended for inductive transfer learning, if the step of labeling an unlabeled instance is replaced by re-weighting a diff-distribution instance. Based on this recognition, we develop a new transfer learning method, namely COITL, by extending the co-training method in semi-supervised learning. Experimental results reveal that COITL can achieve significantly higher generalization and robustness, compared with two state-of-the-art methods in inductive transfer learning.**

Keywords-Inductive transfer learning; semi-supervised learning; co-training

I. INTRODUCTION

In traditional data mining and machine learning research, it is often assumed that there are sufficient training data. For example, in supervised classification task, the labeled training instances are often assumed to be sufficient to train a good learner that can correctly classify unseen instances. In many practical applications, however, labeled instances are insufficient, because labeling training instances is often expensive, difficult, or time-consuming. Meanwhile, unlabeled instances may be relatively easy to obtain, thus can be used to compensate the lack of labeled instances, which motivates a recent machine learning branch - semi-supervised learning [1]. Due to its importance in practice and theory, semi-supervised learning has received considerable attention. A number of learning methods, including self-training, co-training, graph-based methods, etc., have been proposed to effectively learn from both labeled and unlabeled data.

Many machine learning methods, no matter for traditional machine learning or semi-supervised learning problems, are under a common assumption: the training and test data are drawn from the same distribution and the same feature space. So, when the distribution changes, most learning methods fail to work well. Therefore, human effort is required to re-collect a large amount of labeled data to re-train a learner to fit the new distribution. Since the labeling process is often expensive, reducing human effort to label new data attracts increasing attention. This motivates another machine learning branch, namely transfer learning [2], which aims to transfer the knowledge in a source domain to solve the learning task in a related target domain, i.e. the domain we are interested in. The source and target domain have different data distributions or even different feature spaces. For example, in web page classification, we can use transfer learning to adapt a classifier for university pages to the classification task on Facebook.com [2]. This is because that although the data distribution between university web and social web pages are quite different, there still exists some common classification knowledge that can be reused to reduce the labeling effort. Even if we still want to classify the university web pages, transfer learning may also help, because the contents on web pages are changing over time, and the training data can be easily outdated (i.e. under a different distribution to the current data).

Transfer learning has been studied for more than one decade, and various approaches have been developed. Most of these approaches can be summarized into two categories, namely instance-transfer and feature-representation-transfer. In instance-transfer, training instances in the source domain are re-weighted according to their impact on the learning in the target domain [3]. While feature-representation-transfer tries to learn a common feature representation across domains to reduce the domain divergence as well as the training error, making knowledge in different domains easy to transfer [4].

Besides the learning strategies, different problem settings of transfer learning have also been studied. There are mainly three settings of transfer learning, namely inductive transfer learning, transductive transfer learning and unsupervised transfer learning [2]. In this paper, we focus on inductive

1550-4786/09 $26.00 © 2009 IEEE
DOI 10.1109/ICDM.2009.75

transfer learning, in which only a few labeled data in the target domain are available. Besides, we assume that the source-domain data are also labeled, just the same as several other works like [3], [5]. In a word, our problem setting assumes that a large amount of source-domain data (referred to as diff-distribution data in the following) and a small amount of target-domain data (referred to as same-distribution data) are available. Both the diff-distribution and same-distribution training data are labeled and under the same feature space. However, we should note that although a few same-distribution data are provided, they are still insufficient to train a good learner for the target domain. One intuition to solve the problem is to use the same-distribution data to find out some useful diff-distribution instances and reuse them to improve the learning in the target domain, which exactly follows the instance-transfer strategy. In fact, several instance-based approaches for inductive transfer learning have already been proposed, such as auxiliary-data-based method [5] and boosting method for transfer learning [3], but their generalization ability and robustness still need to be improved.

It is noteworthy that although inductive transfer learning and semi-supervised learning address different problems, they share similarities on their methodologies, thus many approaches for semi-supervised learning can be extended to inductive transfer learning setting. To develop better instance-based methods for inductive transfer learning, this paper attempts to extend methods in semi-supervised learning. The basic idea of the extension is to replace the step of labeling an unlabeled instance by re-weighting a diff-distribution instance. Specifically, this paper extends the co-training method. As an often-used semi-supervised learning method, co-training trains two learners separately on two different views, and uses the predictions of each learner on unlabeled instances to augment the training set of the other learner. Such training style has the advantages of high generalization ability and strong robustness, which has been reported by several research works (for example, refer to [6]). In our CO-training method for Inductive Transfer Learning (COITL), two base learners also learn in a collaborated way, however, the information that one learner gives to the other is the weight of a diff-distribution instance instead of the predicted label of an unlabeled instance. The co-training method uses the weighted k Nearest Neighbor (k NN) method [7] as the base learner, and implement two learners by setting different values for the parameter k. Moreover, each k NN learner weights a diff-distribution instance according to its influence on the training error. An efficient implementation of COITL is also presented. Experimenting on eight data sets, we show that the proposed method outperforms state-of-the-art approaches in inductive transfer learning, in terms of generalization error and robustness to noise.

To summarize, this paper has mainly two contributions. First, it bridges inductive transfer learning and semi-supervised learning, and shows that many semi-supervised learning methods can be extended for inductive transfer learning. Second, it develops a new transfer learning method by extending the co-training method in semi-supervised learning. The rest of this paper is organized as follows. Section II presents the related work. Section III gives the problem statement. Section IV discusses the relationship of the two learning branches. Section V gives the details of COITL. Section VI reports the experimental results. Finally, Section VII concludes.

II. RELATED WORK

A. Instance-transfer

Transferring knowledge from instances is intuitively appealing. Although directly reusing the diff-distribution data is unfeasible, there are certain parts of the data that can still be reused to benefit the learning in the target domain [2]. Here we briefly review a few research works with the similar problem setting as ours.

Wu and Dietterich [5] presented a way of integrating auxiliary training data into k NN as well as support vector machine methods. The methodology is to minimize a weighted sum of two loss functions, one for original training data (i.e. same-distribution data) and the other for auxiliary data (i.e. diff-distribution data). Their experiments demonstrated that using auxiliary data can improve the classification performance when the original training data are inadequate. Note that in their work, all diff-distribution training data are given the same weight, yet our method sets the weight for each diff-distribution instance separately.

Liao et al. [8] introduced an auxiliary variable for each diff-distribution instance to reflect the distribution mismatch. Those auxiliary variables are estimated as a byproduct, along with the classifier. They also incorporated a new active learning method to select unlabeled same-distribution instances to be labeled with the help of diff-distribution data. In our method, however, we perform transfer learning only on labeled data.

Dai et al. [3] proposed a boosting algorithm, namely TrAdaBoost, as an extension of the AdaBoost algorithm. The idea is to use a small number of same-distribution instances to find out useful diff-distribution instances by iteratively adjusting their weights. In each iteration, a base learner is trained on the weighted training data and used to predict the label of each training instance. Moreover, TrAdaBoost uses the same strategy as AdaBoost to update the weights of incorrectly classified same-distribution instances while also adopts a different strategy from AdaBoost to update the weights of misclassified diff-distribution instances. Experimental results verified the high generalization of TrAdaBoost.

B. Co-training

The co-training method, first proposed by Blum and Mitchell [9], presents a novel learning style: two learners are separately trained on two sufficient and redundant

views, and the predictions of each learner on unlabeled instances are used to augment the training set of the other one. Here, the two sufficient and redundant views are two attribute sets which satisfy the following two requirements: first, each attribute set is sufficient to train a good learner; second, given the class label, each attribute set is conditionally independent to the other. Later, Dasgupta et al. [10] theoretically showed that when there exist two sufficient and redundant views, the co-trained learners can reduce the generalization error by maximizing their agreement over the unlabeled instances. Although co-training has been successfully utilized in several domains, such as statistical parsing [11] and noun phrase identification [12], in most application scenarios, the requirement of sufficient and redundant views, could not be met. Therefore, some variants of co-training which relax the requirement of sufficient redundancy were proposed.

Goldman and Zhou [13] developed a relaxing version of the co-training method. Their method employs two different supervised learning algorithms to divide the instance space into a set of equivalence classes, and uses cross validation to help label the unlabeled instances and generate the final hypothesis. Their experimental results demonstrated that although there are no sufficient and redundant views, such co-training version can still achieve excellent performance. Recently, Wang and Zhou [14] theoretically proved that, co-training can be effective if the learners are diverse, which implies that the sufficient and redundant views are actually used to achieve the diversity of the learners, and they are not necessary if the diversity can be achieved from other ways. For example, in [7], the diversity is achieved by setting different parameters for the k NN method, so that even with the same view, i.e. the same attribute set, co-training still outperforms many other methods.

In this paper, we adopt the same co-training style in [7], that is, our method does not use two views, and achieves the diversity of learners by setting different values for the parameter k in k NN. Clearly, such implementation is more general than the classical version of co-training.

III. PROBLEM STATEMENT

In our problem setting, the data in the source domain and target domain have the same feature space X and label set Y ($Y = \{0,1\}$), but they are under different distributions. The training set T consists of two sets of instances: training data in the target domain (denoted by T_s) and training data in the source domain (denoted by T_d). All instances in T_s and T_d are labeled. As mentioned above, in this paper, we would refer T_s to the same-distribution data set and T_d to the diff-distribution data set. Moreover, we denote the test set as E , which contains a set of unlabeled instances.

Both the same-distribution data set T_s and the test set E are from the target domain, thus have the same distribution. While T_d and E are from different domains, thus their distributions are different. Note that the size of T_s is supposed to be relatively small, so that merely training on T_s is unable to produce a high-performance learner. However, the size of T_d is relatively large, since we assume that it is easy to obtain large amounts of source-domain data. An illustrative example of T_s and T_d is shown in Figure 1.

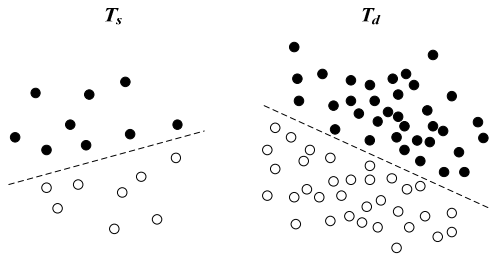


Figure 1. An example of the same-distribution data set and diff-distribution data set

More formally, we define T_s , T_d and E as follows.

The training set in the target domain:

$$T_s = \{(x_1^s, y_1^s), (x_2^s, y_2^s), ..., (x_n^s, y_n^s)\}$$

where $x_i^s \in X (i = 1, 2, ..., n)$ is the feature vector of the i th instance, $y_i^s \in Y$ is its corresponding label, and n is the size of T_s .

The training set in the source domain:

$$T_d = \{(x_1^d, y_1^d), (x_2^d, y_2^d), ..., (x_m^d, y_m^d)\}$$

where $x_i^d \in X (i = 1, 2, ..., m)$ is the feature vector of the i th instance, $y_i^d \in Y$ is its corresponding label, and m is the size of T_d .

The test set (in the target domain): $E = \{x_1^e, x_2^e, ..., x_r^e\}$, where $x_i^e \in X (i = 1, 2, ..., r)$ is the i th instance's feature vector, and r is the size of the test set.

The objective of inductive transfer learning is to use T_s and T_d to learn a function $f : X \to Y$, such that f can correctly predict the label of instances in the test set E .

IV. BRIDGING INDUCTIVE TRANSFER LEARNING AND SEMI-SUPERVISED LEARNING

We know that inductive transfer learning and semi-supervised learning are two different machine learning branches for solving different learning problems. They are motivated by the observation that in many application areas, labeling new data is expensive, as it requires the effort of experienced human annotators. Hence both branches attempt to explore a large number of auxiliary data to reduce the labeling effort. Their differences are:

a. Inductive transfer learning focuses on learning from auxiliary labeled data, while semi-supervised learning focuses on learning from auxiliary unlabeled data.
b. In inductive transfer learning, the auxiliary and original labeled training data are under different distributions or even different feature spaces, while in semi-supervised learning, the auxiliary unlabeled data and original labeled data are drawn from the same distribution.

Considering the above differences, we can see that, although in inductive transfer learning, labels of auxiliary data are available, they cannot be used directly due to the distribution divergence. Therefore, re-weighting training data is often adopted to reduce the distribution divergence. While in semi-supervised learning, although auxiliary data are unlabeled, their labels can be predicted as they are under the same distribution with the labeled data.

Roughly speaking, in both branches, the auxiliary data can be explored in two levels – instance-level and structure-level. In the instance-level, a learning method attempts to select some useful auxiliary instances and add them to the original training set. Typical methods are self-training [15] and co-training [9] [13] in semi-supervised learning, as well as TrAdaBoost [3] in transfer learning. While in the structure-level, a learning method uses the large collection of auxiliary data to help learn the intrinsic structure of the data. Typical methods are harmonic function [16] and manifold regularization [17] in semi-supervised learning, and Eigen-transfer [18] in transfer learning. In the following, we show the connection between the two branches from the instance-level.

It's clear that, in the instance-level, the key step in inductive transfer learning is to calculate the weight of a diff-distribution instance (we only consider instance-transfer here), while the key step in semi-supervised learning is to calculate the label of an unlabeled instance. The basic idea of extending semi-supervised methods for inductive transfer learning is to replace the step of calculating the label of an unlabeled instance with the step of calculating the weight of a diff-distribution instance. This idea is intuitive, since if the weights of diff-distribution instances are determined, the weighted diff-distribution data can be considered under a quite similar distribution to the same-distribution data (this is because, in instance-transfer, re-weighting diff-distribution data essentially reduces distribution divergence). Therefore, after re-weighting, a set of labeled instances under the same or at least very similar distribution to the original labeled data are produced, which is equivalent to the result in semi-supervised learning after the labels of unlabeled instanced being predicted. Thus, semi-supervised learning methods such as self-training and co-training can be easily extended for inductive transfer learning. For instance, co-training can be extended so that each learner teaches the other the weights of some diff-distribution instances. We will discuss the details of such method in next section.

The above thinking points out that, to make the extension feasible, a weighting strategy needs to be designed first. It is noteworthy that the weighting strategy can also be inspired by ideas in semi-supervised learning. For instance, in semi-supervised learning methods, the confidence of an unlabeled instance is often estimated, and the most confident instance is labeled with priority. Such labeling confidence is functionally similar to the weight in transfer learning, since it points out the usefulness of an auxiliary instance. In our co-training method, the weighting strategy can be seen as an extension of the confidence measurement strategy in [7].

V. CO-TRAINING FOR INDUCTIVE TRANSFER LEARNING

A. Weighting strategy

In our co-training method, k NN is chosen to be the base learner, with two considerations. First, due to the iterative property of co-training, a base learner needs to be re-trained many times. Since k NN is a lazy learner, the training process is just updating its training set, which is very efficient and easy to implement. Second, due to the local property of k NN (i.e. the label of an instance is predicted by local neighbors), calculating the weight of an instance can be efficient. Therefore, the weighting strategy discussed below is particularly suitable for k NN, yet its underlying idea can be generalized to other learners.

To facilitate our discussion, we first define the *neighbor error*.

Definition. The *neighbor error* of an instance is the error between its true label and the predicted label by its neighboring instances.

Let (x_i, y_i) denote an instance whose weight is w_i. Its k nearest neighbors are denoted by $x_{i_1}, x_{i_2}, ..., x_{i_k}$, with the weight $w_{i_1}, w_{i_2}, ..., w_{i_k}$, respectively. Therefore, the neighbor error of the instance (x_i, y_i) is calculated according to the following equation:

$$e_{x_i} = | y_i - \frac{\sum_{j=1}^{k} y_{i_j} w_{i_j}}{\sum_{j=1}^{k} w_{i_j}} | \quad (1)$$

Note in the above equation, $\sum_{j=1}^{k} y_{i_j} w_{i_j} / \sum_{j=1}^{k} w_{i_j}$ gives the predicted label of (x_i, y_i) by its k neighboring instances.

The underlying idea of the weighting strategy is to re-weight an instance according to its influence on the training error. If the influence is relatively positive, the instance will have a relatively high weight. Let T' denote the current training set. Initially, T' is set to be the same-distribution data set T_s, and the weights of all instances in T_s are set to be 1 (we will explain how T' being updated in next subsection). For each diff-distribution instance (x_i^d, y_i^d), consider its influence on T'. Assume that due to the addition of (x_i^d, y_i^d), a number of n_{good} instances' neighbor

error is reduced while a number of n_{bad} instances' neighbor error is increased. If $n_{good} \geq n_{bad}$ and $n_{good} > 0$, then the weight of (x_i^d, y_i^d), denoted by w_i^d, is set as follows:

$$w_i^d = \frac{n_{good}}{n_{good} + n_{bad}} \quad (2)$$

Otherwise, the instance is simply dropped. We can see that in equation (2), if n_{good} equals to n_{bad}, the weight has the smallest value 0.5; if $n_{good} > 0$ and n_{bad} equals to 0, the weight has the largest value 1. The intuition is that if a diff-distribution instance brings more positive influence and less negative influence, its weight will be higher. What should be pointed out is that the neighbor error of an instance is not exactly the same as the training error of an instance, since the neighbor error is a real value while the training error is usually a binary value (i.e. correct or not). However, using the neighbor error can help evaluate an instance's influence more accurately due to its real-value property.

B. COITL

In this section, the proposed co-training method (COITL) is discussed in detail. As mentioned in Section II, to make co-training effective, two base learners should be diverse. In our implementation of co-training, two learners both are k NN, thus the diversity is achieved by setting different values for the parameter k. Since the weighting strategy is based on the neighboring structure, different k values will result in differences on instances' weights given by two learners. Such setting also brings another profit, as it is usually difficult to decide which k value is better for the learning problem at hand, two learners with different k values might have complementary effects.

Let L_1, L_2 denote the two learners, and T_1, T_2 be their corresponding training set, respectively. Note, in co-training, each learner is trained on its own training set, and its training set can be updated by the other learner. Initially, T_1 and T_2 are both set to be the same-distribution data set T_s, in which the weights of all instances are set to be 1. In every iterative step, each learner randomly selects an unprocessed diff-distribution instance and judges whether the instance be re-weighted or just dropped. If the diff-distribution instance needs to be re-weighted, then the learner sets the weight for the instance using the weighting strategy discussed in the above subsection, and adds the weighted instance to the training set of the other learner. This training step repeats until all diff-distribution instances are processed. Note that in our method, a learner just selects a random diff-distribution instance and gives it to the other if it can be re-weighted. This implementation differs from the one in semi-supervised learning, where usually the most confident instance is selected. We use the random selection just because it is more efficient than selecting the instance with the maximum weight (analogous to the labeling confidence in semi-supervised learning).

After co-training, the hypotheses of L_1 and L_2 should be combined to give the final hypothesis. A simple method is to use the linear combination, that is, constructing a combined hypothesis h^* by: $h^* = \alpha_1 h_1 + \alpha_2 h_2$, where α_1, $\alpha_2 \in [0,1]$ are two weighing factors, h_1 and h_2 are two hypotheses given by L_1 and L_2, respectively. We can calculate α_1 and α_2 as follows. Let e_1 and e_2 be the training error of L_1 and L_2 on the same-distribution data set T_s. Then $\alpha_1 = \frac{e_2}{e_1 + e_2}$ and $\alpha_2 = \frac{e_1}{e_1 + e_2}$, which implies that the weighting factor of a learner is larger if it produces less training error on T_s. The procedures of COITL are shown in Table I.

TABLE I. THE PROCEDURES OF COITL

Algorithm: COITL

Input: the same-distribution data set T_s (size = n), the diff-distribution data set T_d (size = m), the neighboring parameter k_1, k_2

Process:

for i =1 to n

$w_i^s \leftarrow 1$

end for

$T_1 \leftarrow T_s$, $T_2 \leftarrow T_s$

Set T_1 and T_2 to be the training sets for two learners L_1 and L_2, respectively. The neighboring parameter for L_1 and L_2 are k_1 and k_2, respectively.

for t = 1 to $m/2$

for $j = \{1,2\}$ do

Randomly select an unprocessed diff-distribution instance (x_i^d, y_i^d) from T_d

Use T_j to calculate n_{good} and n_{bad} of (x_i^d, y_i^d)

if $n_{good} \geq n_{bad}$ and $n_{good} > 0$ **then**

$w_i^d = \frac{n_{good}}{n_{good} + n_{bad}}$

if j equals to 1 **then**

Add (x_i^d, y_i^d) with w_i^d to T_{j+1}

else

Add (x_i^d, y_i^d) with w_i^d to T_{j-1}

end if

end if

end for

end for

$h_1 \leftarrow L_1$, $h_2 \leftarrow L_2$

Set α_1 and α_2 as the weights of L_1 and L_2, according to their training error on T_s.

Output: $h^* = \alpha_1 h_1 + \alpha_2 h_2$

C. Efficient implementation

It is noteworthy that the method in [7] also needs to find a set of instances being influenced by an added instance, i.e. having the added instance as a neighboring instance. They stated that finding all influenced instances in the whole training set is time-consuming and proposed an approximation strategy. However, in this subsection, we give an efficient implementation of our method which can accurately find all influenced instances with the same time and space complexity as the approximation strategy.

Note that in the weighting strategy, to calculate n_{good} and n_{bad} of an instance (x_i^d, y_i^d), we need to find out a set of instances influenced by (x_i^d, y_i^d). Here we call the set containing a few instances influenced by (x_i^d, y_i^d) as the influenced set (denoted by Λ_i). In order to efficiently process the influenced set, our method needs to store necessary information. For simplicity, let us first consider the case when there is only one training set, denoted by T'. The size of T' is $O(n+m)$. For each instance (x_i, y_i) in T', we use an array A_i to store the indexes of its k neighbors $(i_1, i_2, ..., i_k)$. The array is sorted in ascending order according to the distance between (x_i, y_i) to each neighbor, i.e., $D(x_i - x_{i_1}) \le D(x_i - x_{i_2}) \le ... \le D(x_i - x_{i_k})$. Here $D()$ returns the distance between two instances. The largest neighbor distance, i.e., $D(x_i - x_{i_k})$, is named the radius of the neighboring structure (denoted by r_i).

It's easy to see that the space complexity of our method is $O(k(n+m))$, since for each instance in the training set, a k-length array is maintained. For discussing the time complexity, we distinguish two operations, namely searching for the influenced set and updating the influenced set. The first operation is invoked when we want to find out the weight of a diff-distribution instance, while the second operation is performed when we want to add a weighted diff-distribution instance to the training set. Assume that calculating the distance of two instances takes c_1 time, and comparing two real values takes c_2 time. Note that in common cases, $c_1 >> c_2$. When searching for the influenced set, we need first to calculate the distance between the diff-distribution instance to each instance in the training set, which takes $c_1 O(n+m)$ time. After that, for each instance in the training set, its distance to the diff-distribution instance should be compared with its radius to see whether the instance is influenced, which takes $c_2 O(n+m)$ time. Therefore, the total time complexity for finding the influenced set for one diff-distribution instance is $c_1 O(n+m) + c_2 O(n+m) = c_1 O(n+m)$ (recall that $c_1 >> c_2$).

Then, let us consider the time complexity for updating the influenced set. Note that the updating operation itself contains two sub-operations: first, the influenced set for the diff-distribution instance should be searched (because in co-training, the weighted instance is added to a new training set); second, the arrays of each instance in the influenced set should be updated. The first sub-operation takes $c_1 O(n+m)$ time, as discussed above. For the second sub-operation, since each array has been already sorted, updating one array takes $c_2 O(k)$ time. Let the maximum size of an influenced set be n_A, thus, modifying all arrays take $c_2 O(kn_A)$ time. Combing the time complexity of the two sub-operations together, the time complexity for updating the influenced set is $c_1 O(n+m) + c_2 O(kn_A)$. Let us make another assumption, that is, $n_A < (n+m)/k$. This assumption makes sense, since usually m is very large while k is relatively small (in our experiments, m is 1000, k is no more than 5, and n_A equals to $3k$). Hence the time complexity for updating the influenced set is: $c_1 O(n+m) + c_2 O(kn_A) < c_1 O(n+m) + c_2 O(n+m) = c_1 O(n+m)$. The equation holds because $c_1 >> c_2$. The above analysis shows that, the time complexity of searching for and updating the influenced set both are $c_1 O(n+m)$.

Now we come to the co-training setting, in which one learner first re-weights a diff-distribution instance (this invokes the operation named searching for the influenced set), and adds the weighted instance to the training set of the other learner (this invokes the operation named updating the influenced set). Thus, the time complexity for one learner to teach the other learner once is $c_1 O(n+m)$.

As mentioned above, in [7], an approximation strategy was proposed to process the influenced set. However, the time complexity of their approximation strategy is still $c_1 O(n+m)$ (as can be proved) and the space complexity is also the same as ours, which implies that the approximation is actually not necessary. Also, as stated in [7], in some cases the approximation may bring negative effects. This further demonstrates the advantage of our efficient implementation.

VI. EXPERIMENTS

A. Datasets and preprocessing

In the experiments, we choose eight datasets, four of which (Mushroom[1], Waveform[2], Magic[3], Splice[4]) are directly obtained from UCI Machine Learning Repository, and the other three are generated by adding noise to the four original datasets. The Mushroom dataset has 8124 instances. Each instance has 22 attributes which describe some

[1] http://archive.ics.uci.edu/ml/datasets/Mushroom

[2] http://archive.ics.uci.edu/ml/datasets/Waveform+Database+Generator+(Version+1)

[3] http://archive.ics.uci.edu/ml/datasets/MAGIC+Gamma+Telescope

[4] http://archive.ics.uci.edu/ml/datasets/Molecular+Biology+%28Splice-junction+Gene+Sequences%29

properties of a mushroom. All instances can be classified into two classes - poisonous or non-poisonous. We use the mechanism in [3] to construct the training and test set: for each instance, if its attribute *stalk-shape* is enlarging, then this instance is put into the same-distribution or test data set; otherwise, it is put into the diff-distribution data set. In all datasets, we fix the size of the diff-distribution data set to be 1000, the size of the test data set to be 1000, and the maximum size of the same-distribution data set is 500 (note in experiments, the size of the same-distribution data set has several different values). To show that the construction mechanism is effective, we can calculate the "diff-test error" by setting the diff-distribution data as the training data and calculate the prediction error on the test data (the underlying learner is k NN with $k = 3$). For the Mushroom dataset, the diff-test error is 0.659, which implies the large distribution difference in the same-distribution data and the diff-distribution data (note the test data and the same-distribution data are under the same distribution). In the experiments, we are also interested in the robustness of a learning method. Here the robustness evaluates its sensitiveness to noisy instances. Therefore, for each of the four datasets, we also construct its corresponding noisy dataset (for example, the Mushroom-Noise dataset corresponds to the Mushroom dataset). The mechanism to construct the noisy dataset is to flip a diff-distribution instance's label with the probability of 0.015. Note that we only add noise to the diff-distribution data. This makes sense, because usually the same-distribution instances are just a few, which are carefully labeled by humans. The diff-distribution data, however, are often large in amounts, which tend to contain noise. The diff-test error of the Mushroom-Noise dataset is 0.666, slightly larger than that of the Mushroom dataset.

The Waveform dataset has 5000 instances, each of which contains 21 attributes. All instances can be classified into three classes; however, in the experiments we only choose instances from the first two classes. For each chosen instance, if its first attribute value is larger than 0.15 and its second attribute value is larger than 0, it is put into the same-distribution or test data set; otherwise, it is put into the diff-distribution data set. The diff-test error is 0.301. Similarly, we construct its corresponding noisy dataset by introducing labeling noise with the probability of 0.015. The diff-test error of the Waveform-Noise dataset is 0.405.

The Magic dataset has 19020 instances, each with 10 attributes. All instances belong to two classes. For each instance, if its first attribute value is larger than 100, it is added to the same-distribution or test data set; otherwise, it is added to the diff-distribution data set. The diff-test error of the Magic dataset and Magic-Noise dataset are 0.123 and 0.204, respectively.

The Splice dataset has 3190 instances, each containing 60 attributes. Each instance belongs to one of three classes ('EI', 'IE' and 'Neither'). Since in this paper, only binary classification problem is addressed, we combine the class 'EI' and 'IE' to form a single class. Then, for each instance, if the first attribute value is 'A' or 'G', we add it to the same-distribution or test data set; otherwise, we add it to the diff-distribution data set. The diff-test error of the Splice dataset and Splice-Noise dataset are 0.263 and 0.287, respectively.

Table II lists the information of the eight datasets.

TABLE II. THE INFORMATION OF THE EIGHT DATASETS

dataset	# attr.	size of T_d	max size of T_s	size of E	diff-test error
Mushroom	22	1000	500	1000	0.659
Mushroom-Noise					0.666
Waveform	21	1000	500	1000	0.301
Waveform-Noise					0.405
Magic	10	1000	500	1000	0.123
Magic-Noise					0.204
Splice	60	1000	500	1000	0.263
Splice-Noise					0.287

Finally, complete content and organizational editing before formatting. Please take note of the following items when proofreading spelling and grammar:

B. Experimental results

In the experiments, four learning methods are chosen to be compared. They are k NN, COITL, Aux-k NN [5] and TrAdaBoost [3]. For k NN, k is set to be 3. For COITL, the two k values are set to be 3 and 5, respectively. The parameter settings for Aux-k NN and TrAdaBoost follow those proposed in literatures. To test the performance of each learning method in different conditions, the size of the same-distribution data varies from 10 to 500. Table III shows the experimental results (test error) on the Mushroom dataset. We can see that, when the size of the same-distribution data set is smaller than 100, transfer learning is effective, as it reduces the generalization error. When the size of the same-distribution data set is larger than 100, the effects of transfer learning are not apparent.

TABLE III. EXPERIMENTAL RESULTS (TEST ERROR) ON THE MUSHROOM DATASET

Learning Method	size of the same-distribution data set							
	10	20	40	60	80	100	250	500
kNN	0.359	0.208	0.122	0.064	0.057	0.038	0.004	0.003
COITL	0.274	0.120	0.087	0.054	0.054	0.033	0.004	0.003
TrAdaBoost	0.375	0.148	0.070	0.070	0.039	0.025	0.004	0.003
Aux-kNN	0.343	0.168	0.105	0.085	0.053	0.046	0.004	0.003

It may be difficult to assess the rank of each learning method; therefore we provide a meaningful criterion to compare the performance of different methods. The criterion is named the Average Test Error Reduction (ATER), which is the mean of the reduced test error compared to k NN when the size of the same-distribution data set varies. Formally, for a learner L^* (COITL, Aux-

k NN or TrAdaBoost), its ATER is obtained using the following equation:

$$ATER(L^*) = \frac{1}{8}\sum_{i=1}^{8}(e_{k\text{NN}} - e_{L^*})$$

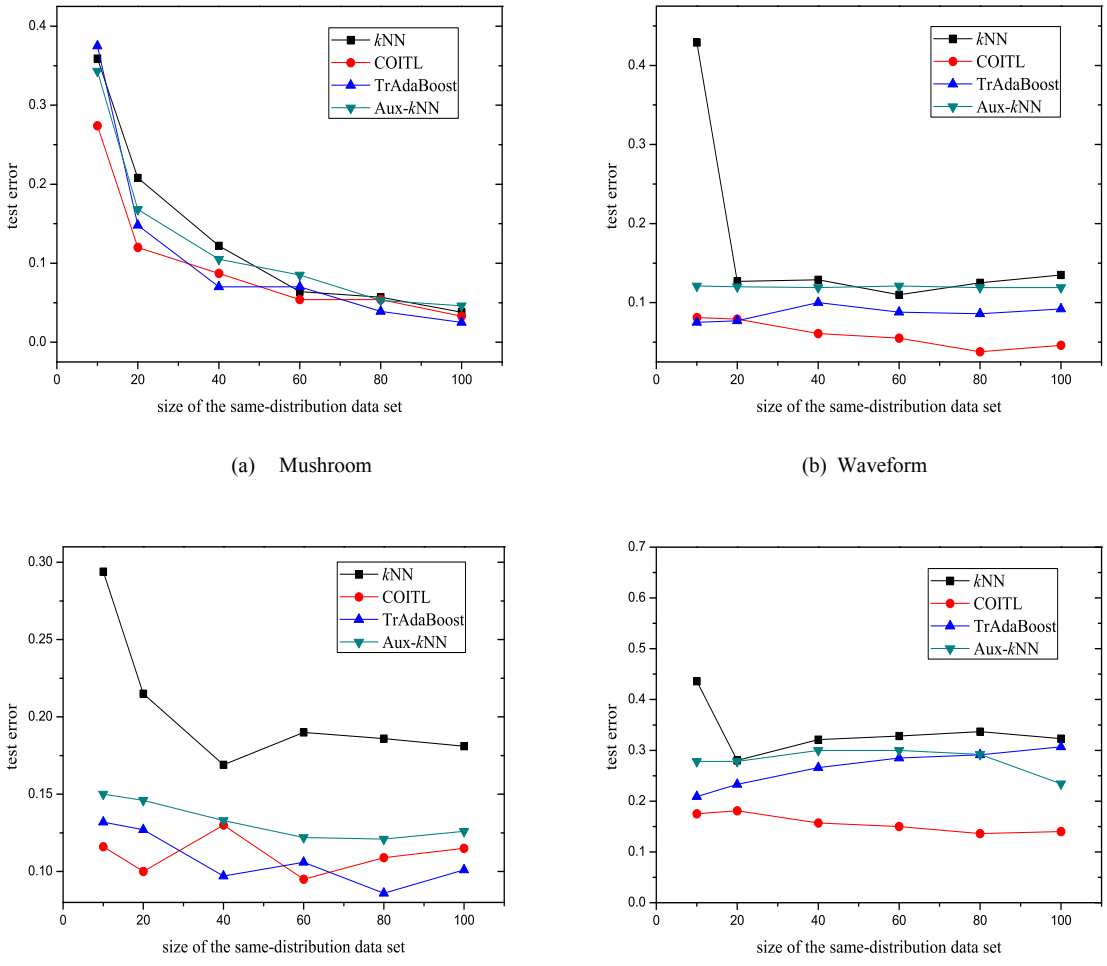
where $e_{k\text{NN}}$ is the test error of k NN and e_{L^*} is the test error of L^*. As can be seen, the ATER uses k NN as the baseline to evaluate the performance of other learning methods. Also, it's easy to see that the ATER reveals the average generalization ability of a learning method.

TABLE IV. THE ATER OF EACH LEARNING METHOD ON THE CLEAR DATASETS.

Learning Method	ATER			
	Mushroom	Waveform	Magic	Splice
COITL	**0.028**	**0.100**	**0.077**	**0.161**
TrAdaBoost	0.015	0.058	0.056	0.065
Aux-kNN	0.006	0.042	0.037	0.035

Table IV lists the ATER of each learning method on the clear datasets. It is evident that COITL significantly outperforms TrAdaBoost and Aux-kNN on all of the four datasets. Also, TrAdaBoost performs better than Aux-kNN on each dataset.

Figure 2 shows the performance comparisons of different learning methods on clear datasets. Note that when the number of the same-distribution data is quite small (i.e. equals to 10), transfer learning methods can significantly improve the test error. For example, in Waveform dataset, the test error is reduced from over 0.4 to about 0.1 when transfer learning is performed. Also note, the curves of the test error are not monotonously descending, because the same-distribution data are so insufficient that adding just a few same-distribution instances may not help to train a better learner.



(a) Mushroom (b) Waveform

(c) Magic (d) Splice

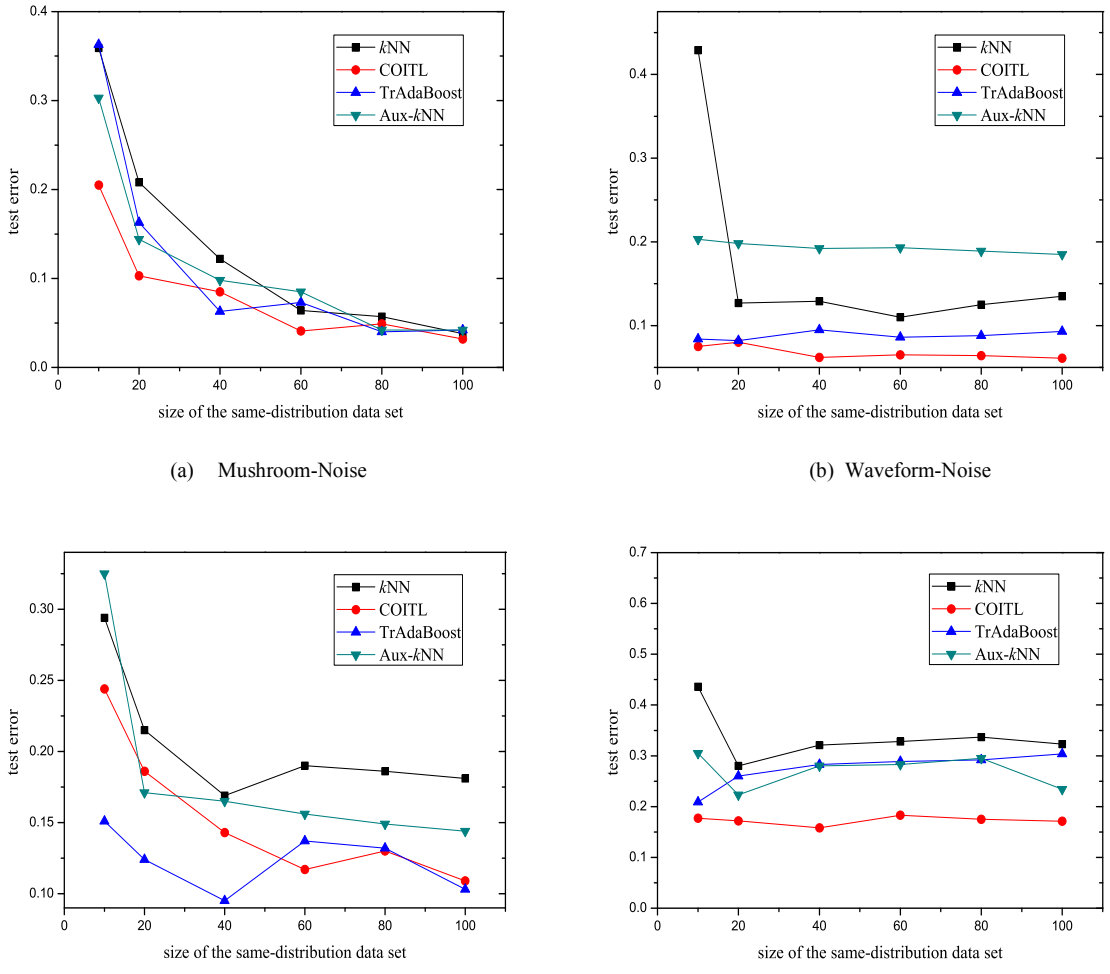
Figure 2. Performance comparisons on clear datasets

TABLE V. THE ATER OF EACH LEARNING METHOD ON THE NOISY DATASETS

Learning Method	ATER			
	Mushroom-Noise	Waveform-Noise	Magic-Noise	Splice-Noise
COITL	**0.042**	**0.095**	**0.043**	**0.145**
TrAdaBoost	0.015	0.057	0.028	0.057
Aux-*k*NN	0.017	-0.022	0.013	0.051

Table V lists the ATER of each learning method on the noisy datasets. Again, COITL achieves much higher generalization than the other learning methods. The results also demonstrate the high robustness of COITL. By contrast, the robustness of TrAdaBoost and Aux-*k*NN are worse. Note that although TrAdaBoost performs better than Aux-*k*NN on the Mushroom dataset, it makes larger test error on the Mushroom-Noise dataset. Also note, Aux-*k*NN has negative ATER on the Waveform-Noise dataset, meaning that its transfer learning brings bad effects. Those observations reflect that TrAdaBoost and Aux- *k*NN are more sensitive to noise than COITL.

Figure 3 shows the performance comparisons of different learning methods on noisy datasets. It is noteworthy that adding noise to the diff-distribution data may reduce the test error of the transfer learning methods. For instance, in the Mushroom-Noise dataset, the three transfer learning methods have better performance than that on the Mushroom dataset. This is not strange because those methods always select useful diff-distribution instances based on the same-distribution data. However, in many cases, the introduction of noisy instances can have bad effects. Comparing Table IV with Table V, we can see that TrAdaBoost and Aux-*k*NN perform worse on the Waveform-Noise and Magic-Noise datasets than on the Waveform and Magic datasets.



(a) Mushroom-Noise (b) Waveform-Noise

(c) Magic-Noise (d) Splice-Noise

Figure 3. Performance comparisons on noisy datasets

The outstanding performance of COITL can be roughly explained as follows. First, it utilizes the co-training mechanism which helps to generalize well through maximizing the agreement on the diff-distribution data by two learners. Second, the different k values for each kNN allow them to have some compensated effects on different datasets. For example, kNN with a very small value of k tends to perform well when there is little noise in the dataset, while kNN with a relatively large value of k may be more robust on noisy dataset. In our experiments, we found that although k values are sensitive to different datasets, setting them to 3 and 5 may be a robust choice, as it gives good results on each dataset. In practice, one can also use cross-validation to choose more proper k values.

VII. CONCLUSION

Inductive transfer learning and semi-supervised learning are two different branches of machine learning. In this paper, we bridge them by showing that many semi-supervised learning methods can be extended for inductive transfer learning, if the step of labeling of an unlabeled instance is replaced by re-weighting a diff-distribution instance. Based on this recognition, we develop the COITL algorithm which extends the co-training method in semi-supervised learning. COITL employs two k NN learners with different values of k . In every learning iteration, each learner re-weights a diff-distribution instance for the other one, where the weight of the instance is determined according to its influence on the training error. The final prediction is made by linearly combining the predictions of both learners. Moreover, an efficient implementation of COITL with relatively low time complexity is discussed. Experimental results show that, compared with two state-of-the-art methods in inductive transfer learning, COITL can have less generalization error and stronger robustness.

There are three research issues to be explored in the future: first, theoretically analyzing the properties of COITL; second, extending other semi-supervised learning approaches, like the graph methods, for inductive transfer learning; third, extending inductive transfer learning approaches for semi-supervised learning. The third issue is quite interesting, as we believe it will bring more insightful understanding of the relationship between semi-supervised learning and transfer learning.

ACKNOWLEDGMENT

We thank the anonymous reviewers for their helpful comments.

REFERENCES

[1] Zhu X., "Semi-supervised Learning Literature Survey", *Technical Report* 1530, University of Wisconsin-Madison, 2008.

[2] Sinno J. P., and Qiang Y., "A Survey on Transfer Learning", *Technical Report*, Hong Kong University of Science and Technology, 2008.

[3] Dai W., Qiang Y., Xue G., and Yong Y., "Boosting for Transfer Learning", In *Proceedings of the 24th International Conference on Machine Learning (ICML)*, 2007.

[4] Argyriou, A., Evgeniou, T., and Pontil, M., "Multitask Feature Learning", In *Proceedings of the 19th Annual Conference on Neural Information Processing Systems (NIPS)*, 2007.

[5] Wu P., and Thomas G. D., "Improving SVM Accuracy by Training on Auxiliary Data Sources", In *Proceedings of the 21st International Conference on Machine Learning (ICML)*, 2004.

[6] Sarkar, A., "Applying Co-training Methods to Statistical Parsing", In *Proceedings of the 2nd Annual Meeting of the North American Chapter of the Association for Computational Linguistics*, 2001.

[7] Zhou, Z. H., and Ming L., "Semi-supervised Regression with Co-training Style Algorithms", *IEEE Transaction on Knowledge and Data Engineering*, 2007, pp. 1479-1493.

[8] Liao, X., Ya X. and Lawrence C., "Logistic Regression with An auxiliary Data Source", In *Proceeding s of the 21st International Conference on Machine Learning (ICML)*, 2004.

[9] Blum, A. and Mitchell, T., "Combining Labeled and Unlabeled Data with Co-training", In *Proceedings of the 11th Annual Conference on Computational Learning Theory (COLT)*, 1998.

[10] Dasgupta, S., Littman, M., and McAllester, D., "PAC Generalization Bounds for Co-training", In *Advances in Neural Information Processing Systems*, MIT Press, 2002, pp. 375-382.

[11] Hwa R., Osborne, M., Sarkar, A., and Steedman, M., "Corrected Co-training for Statistical Parsers", In *Working Notes of the ICML'03 Workshop on Continuum from Labeled to Unlabeled Data in Machine Learning and Data Mining*, 2003.

[12] Pierce, D., and Cardie, C., "Limitations of Co-training for Natural Language Learning from Large Data Sets", In *Proceedings of Empirical Methods in Natural Language Processing*, 2001.

[13] Goldman, S., and Zhou, Y., "Enhancing Supervised Learning with Unlabeled Data", In *Proceedings of the 17th International Conference on Machine Learning (ICML)*, 2000.

[14] Wang, W. and Zhou Z. H., "Analyzing Co-training Style Algorithms", In *Proceedings of the 18th European Conference on Machine Learning (ECML)*, 2007.

[15] Culp M. and Michailidis G., "An Iterative Algorithm for Extending Learners to a Semi-supervised Setting. *Journal of Computational Graphics and Statistics*, 2008.

[16] Zhu, X., Ghahramani, Z., and Lafferty, J., "Semi-supervised Learning Using Gaussian Fields and Harmonic Functions", In Proceeding s of the 20th International Conference on Machine Learning (ICML), 2003.

[17] Belkin, M., Niyogi, P., and Sindhwani, V., "Manifold Regularization: A Geometric Framework for Learning from Labeled and Unlabeled Examples", *Journal of Machine Learning Research*, 2006.

[18] Dai, W., Jin, O., Xue, G., Yang, Q., and Yu, Y., "Eigen-Transfer: A Unified Framework for Transfer Learning", In *Proceedings of the 26th International Conference on Machine Learning (ICML)*, 2009.