Information and Software Technology 54 (2012) 248–256

Contents lists available at SciVerse ScienceDirect

Information and Software Technology

journal homepage: www.elsevier.com/locate/infsof

Transfer learning for cross-company software defect prediction

Ying Ma [a,*], Guangchun Luo [a], Xue Zeng [a,b], Aiguo Chen [a]

[a] School of Computer Science and Engineering, University of Electronic Science and Technology of China, Chengdu 610054, China
[b] Department of Computer Science, University of California, Los Angeles, CA 90095-1596, USA

ARTICLE INFO

Article history:
Received 26 November 2010
Received in revised form 16 August 2011
Accepted 12 September 2011
Available online 19 October 2011

Keywords:
Machine learning
Software defect prediction
Transfer learning
Naive Bayes
Different distribution

ABSTRACT

Context: Software defect prediction studies usually built models using within-company data, but very few focused on the prediction models trained with cross-company data. It is difficult to employ these models which are built on the within-company data in practice, because of the lack of these local data repositories. Recently, transfer learning has attracted more and more attention for building classifier in target domain using the data from related source domain. It is very useful in cases when distributions of training and test instances differ, but is it appropriate for cross-company software defect prediction?
Objective: In this paper, we consider the cross-company defect prediction scenario where source and target data are drawn from different companies. In order to harness cross company data, we try to exploit the transfer learning method to build faster and highly effective prediction model.
Method: Unlike the prior works selecting training data which are similar from the test data, we proposed a novel algorithm called Transfer Naive Bayes (TNB), by using the information of all the proper features in training data. Our solution estimates the distribution of the test data, and transfers cross-company data information into the weights of the training data. On these weighted data, the defect prediction model is built.
Results: This article presents a theoretical analysis for the comparative methods, and shows the experiment results on the data sets from different organizations. It indicates that TNB is more accurate in terms of AUC (The area under the receiver operating characteristic curve), within less runtime than the state of the art methods.
Conclusion: It is concluded that when there are too few local training data to train good classifiers, the useful knowledge from different-distribution training data on feature level may help. We are optimistic that our transfer learning method can guide optimal resource allocation strategies, which may reduce software testing cost and increase effectiveness of software testing process.

1. Introduction

Predicting the quality of software modules is very critical for the high-assurance and mission-critical systems. Within-company defect prediction [1–7] has been well studied in the last three decades. However, there are rarely local training data available in practice, either because the past local defective modules are expensive to label, or the modules developed belong to strange domains for companies. Fortunately, there exist a lot of public data repositories from different companies. But, to the best of our knowledge, very few studies focused on the prediction model trained with these cross-company data.

Cross-company defect prediction is not a traditional machine learning problem, because the training data and test data are under different distributions. In order to solve this problem, Turhan et al. [8] use a Nearest Neighbor Filter (NN-filter) to select the similar data from source data as training data. They discard dissimilar data, which may contain useful information for training. After that, Zimmermann et al. [9] use decision trees to help managers to estimate precision, recall, and accuracy before attempting a prediction across projects. However their method does not yield good results from used across different projects. We consider this a critical transfer learning problem, as defined by Pan and Yang [10].

Unlike these papers, we develop a novel transfer learning algorithm called Transfer Naive Bayes (TNB) for cross-company defect prediction. Instead of discarding some training samples, we exploit information of all the cross-company data in training step. By weighting the instance of training data based on target set information, we build a weighted Naive Bayes classifier. Finally, we perform analysis on publicly available project data sets from NASA and Turkish local software data sets [11]. Our experimental results show that TNB gives better performance on all the data sets when compared with the state-of-the-art methods.

* Corresponding author. Tel.: +86 028 61830557; fax: +86 028 61830580.
E-mail addresses: may@uestc.edu.cn (Y. Ma), gcluo@uestc.edu.cn (G. Luo), snow.zeng@gmail.com (X. Zeng), agchen@uestc.edu.cn (A. Chen).

0950-5849/$ - see front matter © 2011 Elsevier B.V. All rights reserved.
doi:10.1016/j.infsof.2011.09.007

The rest of this paper is organized as follows. Section 2 briefly reviews the background of the transfer learning techniques and software defect prediction algorithms. Section 3 presents our transfer algorithm, and analyzes the theoretical runtime cost of the algorithm. Section 4 describes the software defect data sets, performance metrics used in this study, and shows the experimental results with discussions. Section 5 finalizes the paper with conclusions and future works.

2. Related work

2.1. Transfer learning techniques

Transfer learning techniques allow the domains, tasks, and distributions of the training data and test data to be different, and have been applied successfully in many real-world applications recently. According to [10], transfer learning is defined as follows: Given a source domain D_S and learning task T_S, a target domain D_T and learning task T_T, transfer learning aims to help improve the learning of the target predictive function in D_T using the knowledge in D_S and T_S, where $D_S \neq D_T$ or $T_S \neq T_T$. Cross-company software defect prediction corresponds to inductive transfer learning setting, in which the source and target tasks are the same, while the source and target domains are different, i.e. $T_S = T_T$ and $D_S \neq D_T$.

Inductive transfer learning approaches can be summarized into two categories, instance-transfer and feature-representation-transfer [12]. While the feature-representation-transfer learns low-dimensional representations for reducing the domain divergence, the instance-transfer gives the source instances different weights according to their contribution for building model in target domain. These approaches usually have two steps: (1) estimating the weights of the source domain data; (2) training models on the re-weighted data.

In this research, we mainly focus on instance-transfer approaches. Ref. [13] proposed a kernel mean matching re-weighting process to solve sample selection bias problem. Ref. [14] proposed a logistic regression classifier for differing training and test distributions. Ref. [15] proposed a Kullback–Leibler importance estimation procedure to improve classification accuracy, when the number of training samples are typically limited while test input samples are abundantly available.

Most recently, there are some transfer algorithms extension for existing classifiers. For example, [16] proposed an extension of bagging (called TrBagg). [17] proposed a transfer learning method based on EM algorithm, which can found a locally optimal *a posteriori* hypothesis under the target distribution. [18] proposed a novel framework (called TrAdaBoost) for transferring knowledge from one distribution to another by boosting a basic learner. Transfer learning method proposed in this research, is also the extension for Naive Bayes classifiers.

Transfer learning technique also works well on many fields such as image classification [19], name-entity recognition [20], web page translation [21], natural language processing [22], and email spam filtering [23]. Cross-company defect prediction uses the public software defect data repositories, which are under different distributions. As mentioned above, cross-company defect prediction is also a transfer learning problem. Our experiment results show that the transfer learning algorithm proposed provides a novel method to the cross-company defect prediction.

2.2. Software defect prediction

With the software metric research advanced [24–26], there are a lot of paper using different methods on defect prediction, such as interpretable models [1], decision tree [2], Bayesian networks [3], and neural networks [4]. Menzies et al. [5] compare the performance of learning methods, and also endorse the use of static code attributes for defected-prone models prediction. After that, they report the "ceiling effect" [6], and hold the idea that further progress in learning defect predictors may not come from better algorithms, but come from more information content of the training data. But later, Catal and Diri [7] investigate the effects of data set size, metrics set, and feature selection techniques on performance of prediction model, and show that prediction algorithm is more important than the metrics suite. However, these methods assume sufficient local data for training, which are often difficult to find in reality.

Most recently, unlike these within-company methods, Turhan et al. [8] have proposed a NN-filter method to select the similar samples from source data. Without using all the source data, they only use nearest neighbors for each test samples to form training set, which has similar characteristic to the local data. They recommend managers use the NN-filter method in the early phase of software development, when companies have no local defect data. But the runtime cost of this filter is impractical, especially for large data sets, as the neighbors must be searched for each testing samples. Zimmermann et al. [9] identified the effect of the various characteristics on prediction quality with decision trees, by analyzing similarity levels of test data set and the training data set. Liu et al. [27] proposed a search-based approach to predict defects on multiple repositories. Their method has a lower total cost of misclassification than those of non-search-based models. However, since all the data sets come from NASA, it is unclear to what extent the data can be actually considered cross-company. Without discarding any samples, we developed a transfer learning method by sample weighting, while ignoring the similarity between cross-company projects.

3. Transfer learning for software defect prediction

In this section, we present our Transfer Naive Bayes (TNB) algorithm, based on Naive Bayes. Furthermore, we give the theoretical runtime cost analysis for the algorithm. The main idea of TNB is giving weights to the training samples, according to the similarities between the source and target data on feature level. And then, Naive Bayes classifier is built on these weighted training samples.

3.1. Naive Bayes software defect prediction model

As in the standard machine learning formulation, let $L = \{(x_1, c_1), (x_2, c_2), \ldots, (x_n, c_n)\}$ be the training set, where n is the number of instances in the training set, c_i is the class of instance x_i. In cross-company defect prediction problem, $c_i \in$ (true, false), defective modules are labeled as 'true', and defect-free modules are labeled as 'false'. Furthermore, Let $U = \{u_1, u_2, \ldots, u_m\}$ be the target set, where m is the number of instances in the target set. Recently a well-known work [5] reported that using the Naive Bayes classifier [28] with static code metrics could improve the accuracy of within-company defect prediction. In Naive Bayes, the target instance u can be labeled as following formula.

$$c(u) = \arg\max_{c \in C} P(c|u) = \arg\max_{c \in C} \frac{P(c)\prod_{j=1}^{k} P(a_j|c)}{\sum_{c \in C} P(c)\prod_{j=1}^{k} P(a_j|c)} \quad (1)$$

where $u = \{a_1, a_2, \ldots, a_k\}$, a_j is the jth attribute of the target instance u, k is the number of attributes. $P(c)$, $P(a_j|c)$, and $P(c|u)$ are the prior, conditional, and posterior probability for class c.

The Naive Bayes classifiers are known as a simple Bayesian classification algorithm, assuming that the effect of an attribute value on a given class is independent of the values of other attributes.

Our method also has this assumption. We think all the attributes have the same importance, i.e. the same weight for classification.

3.2. Transfer Naive Bayes

In order to transfer the information of the test data, firstly, the information of the test set (target data set) is collected. Next, every feature of the data in training set is compared with this information. And then, the weight of each training data is calculated through an analogy with gravitation. Finally, the predict model is built, based on the weighted training data.

3.2.1. Collecting target set information

In our method, an instance can also be written as $x_i = \{a_{i1}, a_{i2}, \ldots, a_{ik}\}$, a_{ij} is the jth attribute of x_i. In order to get the information of target set, we compute the maximum value and minimum value of jth attribute on the test set, i.e.

$$max_j = \max\{a_{1j}, a_{2j}, \ldots, a_{mj}\} \quad (2)$$

$$min_j = \min\{a_{1j}, a_{2j}, \ldots, a_{mj}\} \quad (3)$$

where $j = 1,2,\ldots,k$, k is the number of the attributes, and m is the number of the test data. Then, we have two vectors, $Max = \{max_1, max_2, \ldots, max_k\}$, $Min = \{min_1, min_2, \ldots, min_k\}$. These two vectors contain the information of the test set.

We assume each similar attribute has the same contribution to classifying the target set. For each training sample, the degree of similarity to the test set is computed by calculating the location of attributes between these maximum values and minimum values. For each instance x_i in training set, we compute the number of similar attributes,

$$s_i = \sum_{j=1}^{k} h(a_{ij}) \quad (4)$$

where $h(a_{ij}) = \begin{cases} 1 & \text{if } min_j \leqslant a_{ij} \leqslant max_j \\ 0 & \text{else} \end{cases}$, a_{ij} is the jth attribute of the instance x_i.

For example, we have three original training data: $x_1 =$ (2, 1, 3, 'false'); $x_2 =$ (1, 2, 2, 'false'); $x_3 =$ (1, 3, 4, 'true'). The test data are: $u_1 =$ (2, 1, 3); $u_2 =$ (1, 2, 3).

We can compute the Max, Min vectors on the test data, $Max = (2, 2, 3)$; $Min = (1, 1, 3)$. Then we can compute s_i on the training data. According to Eq. (4), we have $s_1 = 3$, because $1 \leqslant 2 \leqslant 2$; $1 \leqslant 1 \leqslant 2$; $3 \leqslant 3 \leqslant 3$. Similarly we have, $s_2 = 2$; $s_3 = 1$.

3.2.2. Data gravitation

In order to transfer the target set information, we introduce data gravitation in our method. Data gravitation simulates the universal gravitation in data analysis. Many studies make data gravitation applicable to machine learning, such as [29–31]. Peng et al. [29] presented a data gravitation based classification, by simulating the gravitation. Wang et al. [30] improved the Nearest Neighbor classifier using a simulated gravitational collapse algorithm to trim the boundaries of the distribution of each class to reduce overlapping. Indulska et al. [31] introduced the center of gravity concept to optimize the end clustering result. To draw an analogy with gravitation, we calculate the weights (just as the force F) of the training data to the test data set.

According to Newton's Universal Gravitation law [32], the universal gravitation exists between any two objects. The gravitation is proportional to the product of the two masses and inversely proportional to the square of the distance between them:

$$F = G\frac{m_1 m_2}{r^2} \quad (5)$$

where G is the universal gravitational constant. m_1 is the mass of one of the objects. m_2 is the mass of the other object. r is the radius of separation between the center of masses of each object. F is the force of attraction between the two objects. In our method, we also weight the training data, by simulating the form of gravitation equation, as shown in the next section.

3.2.3. Weighting for training data

We simulate the universal gravitation between objects to a 'force' (i.e. $W \propto \frac{1}{r^2}$, as shown in Eq. (5)) between our training data and test set. Suppose one feature has mass M, then the mass of the test set is kmM and the mass of each training data is s_iM (the mass of the features located between the Min and Max). Thus, the weight w_i of the training instance x_i is directly proportional to ks_imM^2 and inversely proportional to $r_i^2 = (k - s_i + 1)^2$, which correspond to the product of two masses and the square of the distance in the gravitation formula, respectively. So, the weight of instance can be defined as follows.

$$w_i = \frac{m_1 m_2}{r_i^2} = \frac{ks_imM^2}{(k - s_i + 1)^2} \propto \frac{s_i}{(k - s_i + 1)^2} \quad (6)$$

According to this formula, the data x_i are more similar to the test set, the more weight w_i are given. If the similarity s_i is equal to the number of attributes k, all the attributes locate between the maximum values and minimum values, then the dissimilarity $k - s_i + 1$ is equal to 1, the greatest weight is assigned. We can also normalize them using summation $\sum_{j=1}^{n} w_i$.

The prior probability is computed based on the weighted data. The prior formula is changed to reflect the distribution of the class of the test data. If a training sample is similar to the test data, more weight is given to the training data. And more weight is given to the class of this training data, because this class can be consider to exist in the test set. According to [33], the weighted prior probability for class c can be re-written as follows.

$$P(c) = \frac{\sum_{i=1}^{n} w_i \delta(c_i, c) + 1}{\sum_{i=1}^{n} w_i + n_c} \quad (7)$$

where w_i is the weight of the training instance x_i, c_i is class value of the training instance x_i, n is total number of training instances, n_c is total number of classes, $n_c = 2$ in the prediction model, $\delta(x,y)$ is the indicator function. $\delta(x,y) = 1$ if $x = y$, zero otherwise.

For the class c, the more samples with the same class in the training data, the larger prior probability is given. Give a test instance x, by using sample weighting method described in [33], the conditional probability for jth attribute a_j is

$$P(a_j|c) = \frac{\sum_{i=1}^{n} w_i \delta(a_{ij}, a_j)\delta(c_i, c) + 1}{\sum_{i=1}^{n} w_i \delta(c_i, c) + n_j} \quad (8)$$

where a_{ij} is the value of jth attribute in ith training instance, and n_j is the number of different jth attribute values. In this formula, the conditional probability $P(a_j|c)$ is computed, when the training data has the same attribute value a_j and class c with x.

Since the features of the software defect data are numeric, all the numeric features are discretized using Fayyad and Irani's MDL-based discretization scheme [34]. And then the results are treated as nominal attributes. By combining Eqs. (1), (7), and (8), the test data can be classified based on the predict model.

In the above example, suppose we want to classify the data $u_1 = (2, 1, 3, ?)$. According to Eq. (6), we have

$$w_1 = \frac{3}{(3 - 3 + 1)^2} = 3; \; w_2 = 0.5; \; w_3 = 1/9.$$

(1) $P(c)$ can be computed according to Eq. (7). In the above case, there are three training data and two classes, so $n = 3$ and $n_c = 2$. Then, we have

$$P(\text{'false'}) = \frac{w_1 * 1 + w_2 * 1 + 1}{(w_1 + w2 + w_3) + n_c} = \frac{3 * 1 + 0.5 * 1 + 1}{(3 + 0.5 + 1/9) + 2} = 0.804.$$

$$P(\text{'true'}) = 0.196.$$

(2) $P(a_j|c)$ can also be computed according to Eq. (8). In the above case, $n_1 = 2$, $n_2 = 3$, $n_3 = 3$. We can get

$$P(a_1 = 2|\text{'false'}) = \frac{w_1 * 1 * 1 + 1}{w_1 * 1 + n_1} = \frac{3 * 1 * 1 + 1}{3 * 1 + 2} = 0.8.$$

Similarly we have

$$P(a_2 = 1|\text{ 'false'}) = 2/3; P(a_3 = 3|\text{'false'}) = 5/14.$$
$$P(a_1 = 2|\text{ 'true'}) = 0.5; P(a_2 = 1|\text{'true'}) = 1/3; P(a_3 = 3|\text{ 'true'}) = 1/3.$$

For u_1, according to Eq. (1), we have

$$P(\text{'false'}|u_1) = \frac{P(\text{'false'})\prod_{j=1}^{3} P(a_j|\text{'false'})}{P(\text{'false'})\prod_{j=1}^{3} P(a_j|\text{'false'}) + P(\text{'true'})\prod_{j=1}^{3} P(a_j|\text{'true'})} = 0.933.$$
$$P(\text{'true'}|u_1) = 0.067.$$

Because 0.933 > 0.067, we classify u_1 to class 'false'.

3.2.4. Analysis for TNB

Algorithm 1 presents the pseudo-code of the TNB classifier. Let n be the number of the training set size, m be the number of the test set size, and k be the number of the attributes. Minimum and maximum values of each attribute in the test set must be found with a runtime of $O(km)$. The theoretical runtime of the set weight step is $O(kn)$. Next, a weighted Naive Bayes classification is also built with a runtime of $O(kn)$. Finally, the test step cost runtime of $O(km)$. Therefore, the total theoretical runtime of TNB is $O(k(n + m))$. As the K nearest neighbors for each test sample must be searched, the theoretical runtime of NN-filter method is $O(kmn)$. Since the number of attributes is much smaller than the number of the test samples, the runtime of TNB is much less than that of NN-filter method.

Algorithm 1. Transfer Naive Bayes (TNB).

Require
The set of labeled samples, L;
The set of unlabeled samples, U;
Ensure
TNB classifier, M;
1: compute *Max*, *Min* of U;
2: compute r_i of each instance x_i;
3: **for** each instance $x_i \in L$ **do**
4: according to Eq. (6), set w_i to x_i;
5: **end for**
6: Build weighted Naive Bayes;
7: **for** each instance $u_i \in U$ **do**
8: use Eq. (1) to predict u_i;
9: **end for**
10: **return** M;

This classifier has different advantages, due to the inductive bias of the specific target set data as well as the distributional differences among the source data. In [8], the distances from each test sample to all training samples are computed. While the K nearest neighbors of each test sample are used to train the model, the further samples are discarded. In our method, we use all the training data. The discarded samples in NN-filter method also contain useful information. We still use them to build classifier by giving different weights. This strategy enable the learning model be able to use more information available on the feature level. The experiment results also show a good performance of our method. On the other hand, we only compute the maximum values and minimum values of the test data, when scanning the test data. And then we compare these values with training data only once. So, the time-complexity is a linear function of the number of training data and test data.

4. Experiments

In this section we evaluate TNB algorithm empirically. We use Naive Bayes classifier in Weka [35] to conduct the CC method. And we implement the NN-filter and TNB methods in Weka environment. We focus on cross-company defect-prone software modules prediction problems in this experiment. As we will show later, TNB significantly improves prediction performance with less time over the sample selecting method when applied to defect data sets from different companies.

4.1. Data set

In this study, we use multiple sets of software defect records from different companies as source data, to predict an unlabeled defect set, which is regarded as target data from another company. Since the source data and target data are collected from different companies, they are under different distributions. All our source data sets come from NASA data sets, our target data sets come from Turkish software company (SOFTLAB), both of which can also be obtained from PROMISE [11], as shown in Table 1. The feature numbers are not the same in all data sets, so we use all shared features in these data sets, as shown in Table 2.

The source data sets are seven subsystems taken from NASA, and the target data sets are three systems related to the embedded controller for white-goods from SOFTLAB. Although, these projects have some relation within the two organizations, these companies are very different across the two organizations. That is, the projects from the companies of these different organizations are very different. Therefore, using these source data to predict defects on each target data set is a transferring prediction from cross-company projects to another.

4.2. Performance measures

To evaluate the performance of the prediction model, we can use the confusion matrix of the predictor from Weka [35]. In this matrix,

True positive is: the number of defective modules predicted as defective;
False negative is: the number of defective modules predicted as non-defective;

Table 1
Source data and target data.

Project	Examples	% Defective	Description
Source data (NASA)			
pc1	1109	6.94	Flight software
kc1	1212	26.00	Storage management
kc2	522	20.49	Storage management
kc3	458	9.38	Storage management
cm1	498	9.83	Spacecraft instrument
mw1	403	7.69	A zero gravity experiment
mc2	161	32.30	Video guidance system
Target data (SOFTLAB)			
ar3	63	12.7	Embedded controller
ar4	107	18.69	Embedded controller
ar5	36	22.22	Embedded controller

Table 2
All shared features in NASA and SOFTLAB projects.

Type	#	Feature
Mccabe	2	Cyclomatic complexity; design complexity
Loc	4	Code and comment loc; comment loc Executable loc; total loc
Halstead	10	Halstead difficulty; halstead effort Halstead error; halstead length Halstead time; halstead volume Unique operands; unique operators Total operands; total operators
Other	1	Branch count
Total	17	

False positive is: the number of non-defective modules predicted as defective;
True negative is: the number of non-defective modules predicted as non-defective, as shown in Table 3.

In our study, we use four performance metrics, *pd*, *pf*, $F-measure$, and AUC, which are commonly used in defect prediction. They are defined as follows.

Probability of Detection (*pd*) [5], also called recall and true positive rate in [36]. It is a measure of completeness, defines the probabilities of true defective modules in comparison to the total number of defective modules:

$$pd = recall = \frac{TP}{TP + FN} \tag{9}$$

Precision, a measure of exactness, defines the probabilities of true defective modules to the number of modules predicted as defective:

$$precision = \frac{TP}{TP + FP} \tag{10}$$

As [37] states "the $F-measure$ combines *recall* and *precision* with an equal weight", $F-measure$ can be defined the as follows:

$$F-measure = \frac{(\alpha+1) * recall * precision}{recall + \alpha * precision} \tag{11}$$

where $\alpha \in (0,+\infty)$, is the weight of recall metric. In this research, we use $\alpha = 1$. Probability of False alarm (*pf*) [5], also called false positive rate. It defines the probabilities of false positive modules in comparison to the total number of non-defective modules,

$$pf = \frac{FP}{FP + TN} \tag{12}$$

Receiver operating characteristics (ROC) graph depicts the relative tradeoff between benefit and cost [36]. ROC graph is two-dimensional graph in which *pd* is plotted on the *Y* axis and *pf* is plotted on the X axis. In order to compare the performances of the different prediction models, ROC performances can be reduced to a single scalar value. The area under the ROC curve (AUC) [36] is the portion of the area of unit square. In this unit, the diagonal line between (0,0) and (1,1), which has an area of 0.5. An AUC less than 0.5, it means very low *pd* and very high *pf*. We seek a predictor with high *pd* and low *pf*, In other words, the high AUC is the ideal case.

Table 3
Confusion matrix of the predictor.

	Classified true	Classified false
Real true	True positive (TP)	False negative (FN)
Real false	False positive (FP)	True negative (TN)

Table 4
Experimental results: AUC, *pd*, *pf*, and $F-measure$ comparisons, using all the NASA data sets as training data and each SOFTLAB data set as test data. The bold results in last column indicate the performances of TNB significantly outperform NN-filter.

Metric	CC	NN-filter	TNB
	ar3		
AUC	0.5009 ± 0.0000	0.6511 ± 0.0001	**0.7139** ± 0.0002
pd	1.0000 ± 0.0000	0.8750 ± 0.0000	0.8750 ± 0.0000
pf	0.9982 ± 0.0000	0.5727 ± 0.0003	**0.4473** ± 0.0007
F-measure	0.2257 ± 0.0000	0.3012 ± 0.0000	**0.3540** ± 0.0002
	ar4		
AUC	0.5264 ± 0.0000	0.6170 ± 0.0000	**0.6920** ± 0.0001
pd	1.0000 ± 0.0000	0.8500 ± 0.0000	0.8300 ± 0.0006
pf	0.9471 ± 0.0001	0.6161 ± 0.0001	**0.4460** ± 0.0008
F-measure	0.3268 ± 0.0000	0.3753 ± 0.0000	**0.4405** ± 0.0001
	ar5		
AUC	0.5036 ± 0.0001	0.7839 ± 0.0001	**0.8268** ± 0.0002
pd	1.0000 ± 0.0000	1.0000 ± 0.0000	1.0000 ± 0.0000
pf	0.9929 ± 0.0002	0.4321 ± 0.0004	**0.3464** ± 0.0008
F-measure	0.3653 ± 0.0000	0.5696 ± 0.0001	**0.6232** ± 0.0004

As accuracy is considered a poor performance measure for imbalanced defect data sets, we used AUC to estimate the performance of each classifier. These metrics have the range, $0 \leqslant pd, pf, F-measure, AUC \leqslant 1$. The models perform better with the higher values of *pd*, $F-measure$, AUC, and the lower *pf* values.

4.3. Experiment on data sets from very different companies

In order to investigate the performance of our algorithm, we conduct the experiment on the data sets from very different companies, i.e. NASA aerospace software companies and Turkish domestic appliances company. Using the projects from these two organizations as source data and target data, can be actually considered as cross-company circumstances.

The performance of TNB are compared with that of CC (use all training data to build prediction model directly) and NN-filter (select similar training data to build prediction model, in our experiment $K = 10$ as in [8]).

First, we merge all the NASA data sets as source data. Each target set from SOFTLAB is used as test data. Note that we only use all the available common features (i.e. 17 attributes in total, details in Table 2). Then, we transform the original independent variable values using log filter. For each method, the processing is repeated 10 times considering that the sampling of subsets introduces randomness. Details can be seen from [5,8].

CC: random select 90% source data for training.

NN-filter [8]: For each test data, K nearest neighbors are selected from candidate training set. The total of the similar data is $K \times N$ (N is the number of the test data). And then, unique ones are used for training a predictor.

TNB: first, random select 90% source data as training data and then, use the TNB algorithm on these training data.

Note that, the defective ratios of the training data are similar to those of the original source data in the three methods.

Finally, for comparing the results for these methods, we conducted Mann–Whitney U-Test[1] with level of significance: 5% ($P = 0.05$). That is, we speak of two results for a data set as being "significantly different" only if the difference is statistically significant at the 0.05 level according to the Mann–Whitney U-Test.

We calculate the means and variances of these 10 times' results for the three methods. The *pd*, *pf*, AUC and $F-measure$ values are

[1] Mann–Whitney U test is a non-parametric statistical hypothesis test to compare two independent groups of sampled data, which is without an assumption of a normal distribution. For details see [38].

Table 5
The average running time (in seconds) for 10 times.

Data set	CC	NN-filter	TNB
ar3	0.3250	27.1578	0.3438
ar4	0.3375	59.9438	0.3578
ar5	0.3141	10.8359	0.3296

summarized in Table 4. It shows that CC method not only has both highest *pd* and *pf* values (as described in [8]), but also has the lowest AUC and $F-measure$ values. The high *pf* values are reduced dramatically by using NN-filter method in all cases.

On all target sets, TNB achieves higher AUC values than NN-filter significantly. Similar to the results obtained using the AUC performance metric, TNB outperforms NN-filter with respect to the $F-measure$ and *pf*. TNB method only has one lower *pd* value than NN-filter, but it is still comparable. Although, our result cannot achieve the within company model results (*pd* = 0.75, *pf* = 0.25 in [5]), we still consider it a good result. Note that within company defect prediction is relative simpler than the cross-company defect prediction, since the training data and test data have the same distribution in the former situation, but have different distributions in the latter situation.

The testing time is crucial for large-scale projects with a large number of test data. All of our experiments are performed on an IBM PC (2.99 GHz CPU with 1.50 GB RAM). In Table 5, we compare TNB with other methods in terms of the average total time on the three data sets. We can see that the computational time cost of TNB is close to that of the CC method. The runtime cost for NN-filter is dramatically higher than other comparative algorithms especially when the size of the data set is larger, i.e. ar4. This is because that NN-filer iterates across the entire data space repeatedly until all the distances from test data to training data have been computed.

4.4. Experiment on data sets with different sizes

In order to investigate the effect of the training size on these methods, we also change the numbers of training samples in the experiment. In Figs. 1–3, all the four performance results are recorded when the training size is gradually increased from 10% to 100%. From these three figures, we can see that TNB always improves the AUC, $F-measure$ and *pf* performances over CC and NN-filter. Although the *pd* values of CC are very high (close to 1.0 on all data sets), it is not a good prediction model. The use of the CC method is not practical, because CC method also has too high *pf* values, which means that this method predicts almost all the non-defective modules as defective. This performance can also be seen from the AUC and $F-measure$ curves, which are very low (i.e. below 0.55, 0.55, 0.65 and 0.22, 0.32, 0.35 on ar3, ar4, ar5, respectively).

In Fig. 1, we focus on the ar3 data set. For all the training example rates, the *pd* values of TNB method are close to NN-filter method, as the training data rate gradually increases. This means that the TNB can use more information of the training data, when more training samples are used. When the ratios are less than 0.9, TNB has little lower pd values than NN-filter, but still comparable. What is more, TNB always has better *pf* performances than NN-filter. Considering all the four metrics from Fig. 1, we can conclude that TNB has better performances than NN-filter on ar3 data set.

From Fig. 2, we can see that the *pd* values of the TNB are slightly lower than that of the NN-filter on ar4. But TNB improves the AUC, $F-measure$ and *pf* performance dramatically, i.e. the *pf* values decrease from around 0.6 to around 0.4, the AUC values increase from around 0.6 to around 0.7, and the $F-measure$ values increase from around 0.36 to around 0.46.

For further studying the performance of the compared methods, the median performance values on ar5 are depicted in Fig. 3. We can see that the TNB has *pd* values close to 1.0 and *pf* values below 0.4. This means that, although TNB method does not achieve the

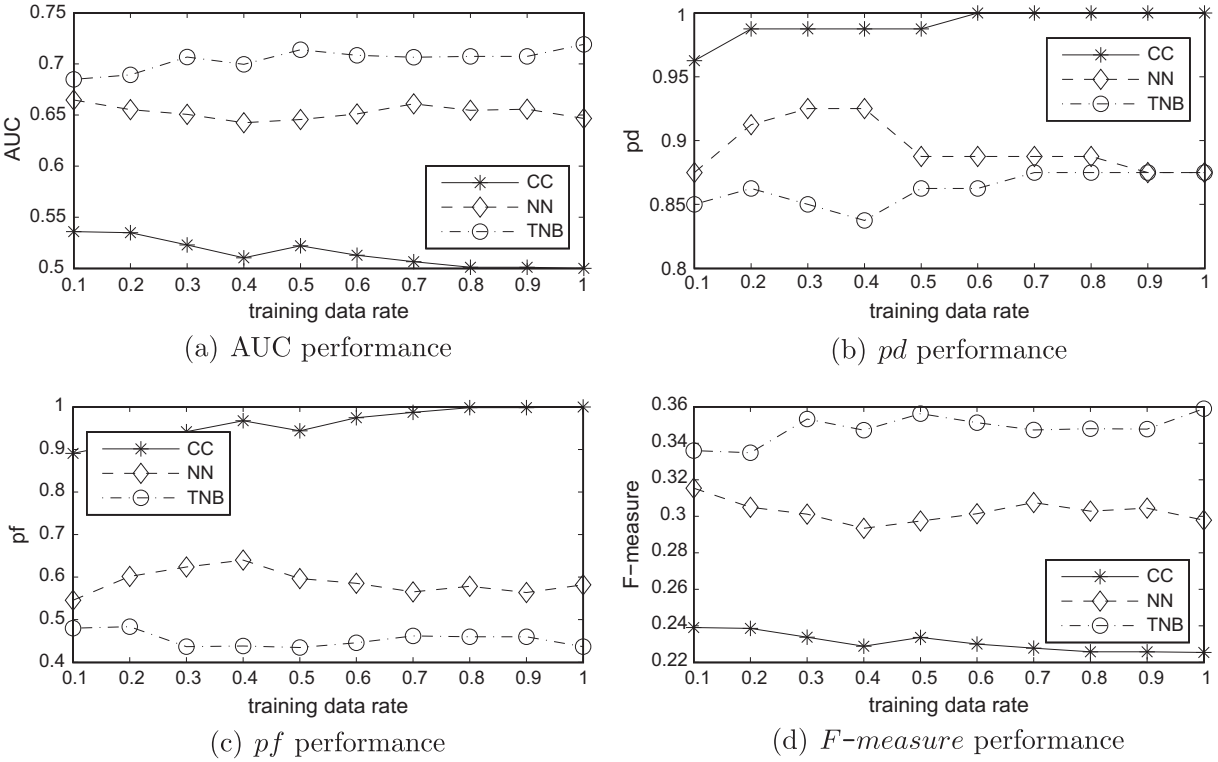
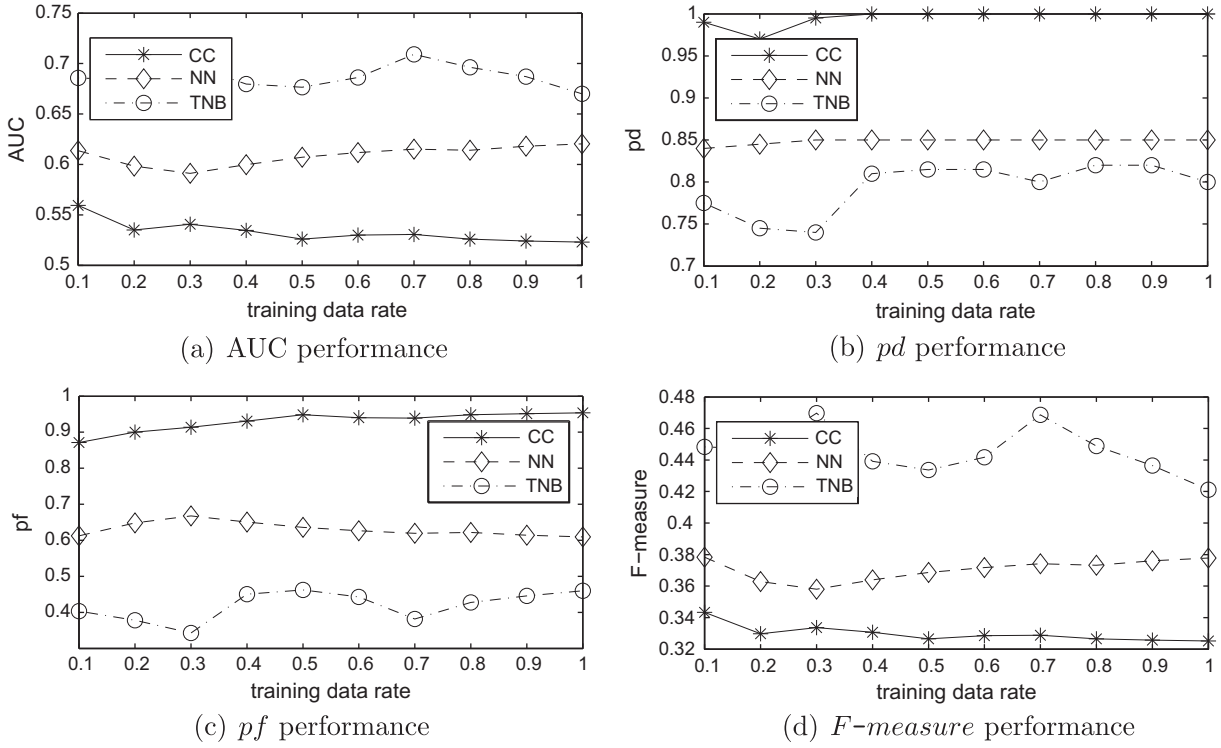


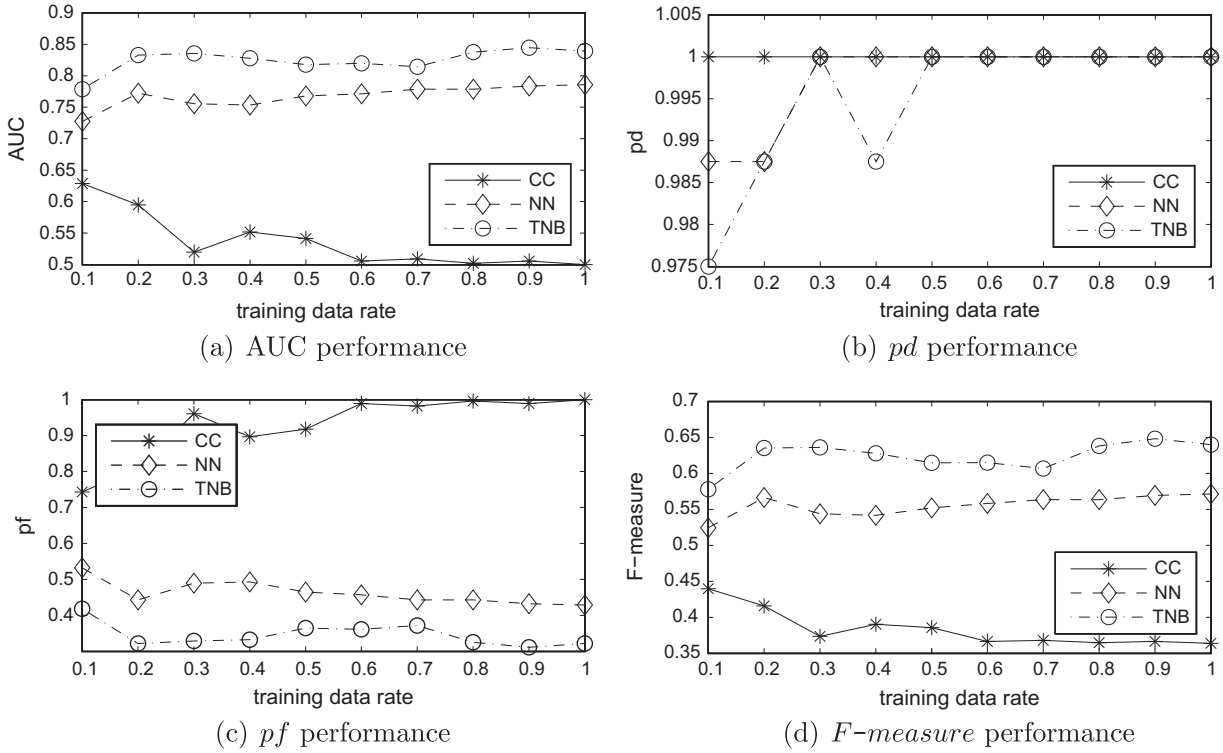
Fig. 1. Performances compared on ar3.



(a) AUC performance (b) pd performance

(c) pf performance (d) F-$measure$ performance

Fig. 2. Performances compared on ar4.



(a) AUC performance (b) pd performance

(c) pf performance (d) F-$measure$ performance

Fig. 3. Performances compared on ar5.

performance within company defect prediction model [5], it still has a good performance across-company defect prediction. We are optimistic that this method can provide an alternative technique guiding the corrective action.

4.5. Experiment on data sets from related companies

The above experiments are conducted on the very different data sets, i.e. the training sets are from NASA and test sets are from

Table 6
Experimental results: AUC and $F-measure$ comparisons for each NASA data set. The line $w/t/l$ means that the algorithm at the corresponding TNB wins in w data sets, ties in t data sets, and loses in l data sets, compared with the algorithm at the corresponding column.

Data	CC	NN-filter	TNB
	AUC		
kc1	0.6170 ± 0.0014	0.6130 ± 0.0021	0.6236 ± 0.0017
mc2	0.5745 ± 0.0243	0.6171 ± 0.0118	0.6252 ± 0.0199
kc3	0.7021 ± 0.0321	0.6406 ± 0.0117	0.7377 ± 0.0074
mw1	0.6506 ± 0.0028	0.6349 ± 0.0244	0.6777 ± 0.0114
kc2	0.7612 ± 0.0030	0.7641 ± 0.0023	0.7787 ± 0.0024
pc1	0.6066 ± 0.0082	0.5981 ± 0.0060	0.5796 ± 0.0100
cm1	0.6440 ± 0.0148	0.6071 ± 0.0296	0.6594 ± 0.0117
w/t/l	3/4/0	2/5/0	
	F-measure		
kc1	0.4567 ± 0.0031	0.4286 ± 0.0052	0.4701 ± 0.0030
mc2	0.5053 ± 0.0280	0.4688 ± 0.0264	0.5385 ± 0.0315
kc3	0.2655 ± 0.0155	0.2695 ± 0.0169	0.2579 ± 0.0140
mw1	0.1817 ± 0.0071	0.1987 ± 0.0085	0.1981 ± 0.0056
kc2	0.5097 ± 0.0098	0.5372 ± 0.0053	0.5601 ± 0.0059
pc1	0.1686 ± 0.0030	0.1407 ± 0.0016	0.1427 ± 0.0031
cm1	0.2564 ± 0.0118	0.2242 ± 0.0113	0.2704 ± 0.0127
w/t/l	2/5/0	1/6/0	

SOFTLAB companies. In this section, further experiment results are showed on the projects from related companies, i.e. NASA companies. NASA projects were all developed by contractors under NASA (mainly for American aerospace software applications), had to follow stringent ISO-9001 industrial practices imposed by NASA [5]. That is, these projects can be considered from the companies in similar application domain to some extent. In the extensive experiment, each date set from the NASA data sets is used as test data, and the rest data sets are used as training data.

Table 6 shows the results of the average and standard deviation of AUC and $F-measure$ values. Each method runs 10 times on the NASA data sets. And then, we conduct Mann–Whitney test with significantly different probability of 0.95 to compare our algorithm with other algorithms. The line w/t/l summarized at the bottom of the table, means that TNB wins in w data sets, ties in t data sets, and loses in l data sets, compared with the algorithm at the corresponding column. We summarize the highlights as follows:

TNB outperforms the CC and NN-filter method in the aspect of AUC. Compared with CC method, TNB wins 3 data sets, ties 4 data sets, loses 0 data set. And compared with NN-filter method, TNB wins 2 data sets, ties 5 data sets, loses 0 data set.

TNB outperforms the CC and NN-filter method in the aspect of $F-measure$. Compared with CC method, TNB wins 2 data sets, ties 5 data sets, loses 0 data sets. And compared with NN-filter method, TNB wins 1 data set, ties 6 data sets, loses 0 data set.

In terms of the average AUC and $F-measure$, TNB is the best among the three methods compared. From Tables 4 and 6, we can see that the TNB achieves bigger improvements on SOFTLAB data sets than on NASA data sets. These different improvements may be caused by a variety of reasons, such as the scale of the project, similarity of the data sets, standard of the projects and so on. The experiments show that TNB can get better performances on the very different data sets from different companies. When there are too few same-distribution training data to train good classifiers, the useful knowledge from different-distribution training data on feature level may help.

5. Threats to validity

As every empirical experiment, our results are subject to some threats to validity.

5.1. Construct validity

First of all, we covered only a small number of data sets from specific sources, namely, NASA MDP and SOFTLAB. We cannot necessarily generalize to other data sets from the current study, since the attributes of these data sets may not be the same. Second, there are some potential issues about the defects can be raised, for example, whether the defects are incompletely fixed, whether the defects are recorded. Since we know that a large number of researchers used the open data sets studied here, we consider that the defects were revealed and fixed adequately in our study.

5.2. Internal validity

Many structural measures have been found to be strongly correlated with other measures, and their additional benefits in characterizing defect-prone modules have been questioned. In our method, we suppose the feature is independent with each other, and has the same importance to building the defect model. The traditional Naive Bayes classifier also assumes the effect of an attribute value on a given class is independent of the values of other attributes, but still has good performance in with-in company defect prediction. Therefore, the internal validity threat caused by this attribute correlation should be minor. A second issue affecting internal validity is that developers' skills and expertise training could affect defect proneness. Similarly to the earlier studies mentioned above, such data were not available in this study.

5.3. External validity

In this study, we validated our findings on open data sets with different characteristics, from two different organizations, i.e. NASA and SOFTLAB. By doing so, we have gained more confidence in the external validity of the results. However, one could still think more about the ability of application of our method in industrial practice, due to the differences in different systems developed for different domains with different complexity. Surely, the replicated studies examining our method on other software systems will be useful to generalize our findings and improve our method.

5.4. Statistical validity

The Mann–Whitney U test, is non parametric test. It does not rest on any assumption concerning the underlying distributions. We do not known whether the performances appear any distributions, so we use Mann–Whitney U test in our study. Furthermore, Mann–Whitney U test is frequently used in many data mining articles. To the best of our knowledge, there is little current negative criticism for Mann Whitney U test as a statistical test for data miners, but many for others.

The domain of the systems and development teams of the projects in our study will be different to that of many other companies. Thus it might be possible that our results do not generalize to the projects from other companies. Although, it is may be misunderstood as a criticism of empirical studies, this study shows encouraged results with transfer learning method. Our method should encourage more researchers to run similar studies, deepen the understanding of the field, and develop more practical predict models. Cross-company prediction research has a lot of serious challenges. Our study would be replicated with more projects, different metrics, and replaced by more sophisticated method.

6. Conclusion and future work

In this paper, we addressed the issue of how to predict software defects using cross-company data. In our setting, the labeled training data are available but have a different distribution from the unlabeled test data. We have developed a sample weighting algorithm based on Naive Bayes, called Transfer Naive Bayes.

The TNB algorithm applies the weighted Naive Bayes model by transferring information from the target data to the source data. First, it calculates the each attribute information of the target data. Then, the degree of similarity for each sample in the source data is computed by comparing with the information collected in the first step. After giving each training instance a weight, based on the similarity, this method build the TNB model on the weighted training instances. Experiments are conducted to show that TNB can give good performances among the comparative methods on the test data sets. Moreover, TNB also shows excellent runtime cost property. We are optimistic that this method can guide optimal resource allocation strategies, which may reduce software testing cost and increase effectiveness of software testing process.

There are several areas in which we can improve this work. First, when we try to transfer the target data information, we only use simple information of the target data, i.e. the max and min values of each attribute. However, such information can not reflect the precise characters of target data, how to get more information of the target data should be well studied in our following work. Second, in the future we will try to investigating other transfer algorithms for cross-company software defect prediction on more software defect data sets.

Acknowledgements

This research was partially supported by National High Technology Research and Development Program of China (No. 2007AA01Z443), Research Fund for the Doctoral Program of Higher Education (No. 20070614008), and the Fundamental Research Funds for the Central Universities (No. ZYGX2009J066). We thank the anonymous reviewers for their great helpful comments.

References

[1] L. Briand, V. Basili, C. Hetmanski, Developing interpretable models with optimized set reduction for identifying high risk software components, IEEE Transactions on Software Engineering 19 (11) (1993) 1028–1044.
[2] T.M. Khoshgoftaar, E.B. Allen, J.P. Hudepohl, S.J. Aud, Application of neural networks to software quality modeling of a very large telecommunications system, IEEE Transactions on Neural Networks 8 (4) (1997) 902–909.
[3] N. Fenton, M. Neil, P. Hearty, W. Marsh, D. Marquez, P. Krause, R. Mishra, Predicting software defects in varying development lifecycles using Bayesian nets, Information and Software Technology 49 (1) (2007) 32–43.
[4] S. Kanmani, V. Rhymend Uthariaraj, V. Sankaranarayanan, P. Thambidurai, Object-oriented software fault prediction using neural networks, Information and Software Technology 49 (5) (2007) 483–492.
[5] T. Menzies, J. Greenwald, A. Frank, Data mining static code attributes to learn defect predictors, IEEE Transactions on Software Engineering 33 (11) (2007) 2–13.
[6] T. Menzies, B. Turhan, A. Bener, G. Gay, B. Cukic, Y. Jiang, Implications of ceiling effects in defect predictors, in: Proceedings of the 4th International Workshop on Predictor Models in Software Engineering, 2008, pp. 47–54.
[7] C. Catal, B. Diri, Investigating the effect of dataset size, metrics sets, and feature selection techniques on software fault prediction problem, Information Sciences 179 (8) (2009) 1040–1058.
[8] B. Turhan, T. Menzies, A.B. Bener, J.D. Stefano, On the relative value of cross-company and within-company data for defect prediction, Empirical Software Engineering 14 (5) (2009) 540–578.
[9] T. Zimmermann, N. Nagappan, H. Gall, E. Giger, Cross-project defect prediction: a large scale experiment on data vs. domain vs. process, in: Proceedings of the 7th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering (ESEC/FSE), 2009, pp. 91–100.
[10] S.J. Pan, Q. Yang, A survey on transfer learning, Technical Report HKUST-CS 08-08, Department of Computer Science and Engineering, Hong Kong University of Science and Technology, 2008.
[11] G. Boetticher, T. Menzies, T. Ostrand, The PROMISE Repository of Empirical Software Engineering Data, 2007 <http://promisedata.org/repository>.
[12] Y. Shi, Z. Lan, W. Liu, W. Bi, Extending semi-supervised learning methods for inductive transfer learning, In: Ninth IEEE International Conference on Data Mining, 2009, pp. 483–492.
[13] J. Huang, A. Smola, A. Gretton, K.M. Borgwardt, B. Scholkopf, Correcting sample selection bias by unlabeled data, in: Proceedings of the 19th Annual Conference on Neural Information Processing Systems, 2007, pp. 601–608.
[14] S. Bickel, M. Bruckner, T. Scheffer, Discriminative learning for differing training and test distributions, in: Proceedings of the 24th International Conference on Machine Learning, 2007, pp. 81–88.
[15] M. Sugiyama, S. Nakajima, H. Kashima, P.V. Buenau, M. Kawanabe, Direct importance estimation with model selection and its application to covariate shift adaptation, in: Proceedings of the 20th Annual Conference on Neural Information Processing Systems, 2008, pp. 1433–1440.
[16] T. Kamishima, M. Hamasaki, S. Akaho, TrBagg:A Simple Transfer Learning Method and Its Application to Personalization in Collaborative Tagging, in: Proceedings of the IEEE International Conference on Data Mining (ICDM), 2009, pp. 219–228.
[17] W. Dai, G. Xue, Q. Yang, Y. Yu, Transferring naive bayes classifiers for text classification, in: Proceedings of the 22rd AAAI Conference on Artificial Intelligence, 2007, pp. 540–545.
[18] W. Dai, Q. Yang, G. Xue, Y. Yu, Boosting for transfer learning, in: Proceedings of the 24th International Conference on Machine Learning, 2007, pp. 193–200.
[19] P. Wu, T.G. Dietterich, Improving svm accuracy by training on auxiliary data sources, in: Proceedings of the 21st International Conference on Machine Learning, 2004, pp. 871–878.
[20] A. Arnold, R. Nallapati, W.W. Cohen, A comparative study of methods for transductive transfer learning, in: Proceedings of the IEEE International Conference on Data Mining (ICDM), 2007, pp. 77–82.
[21] X. Ling, G.-R. Xue, W. Dai, Y. Jiang, Q. Yang, Y. Yu, Can Chinese web pages be classified with english data source? in: Proceedings of the 17th International Conference on World Wide Web, 2008, pp. 969–978.
[22] J. Blitzer, R. McDonald, F. Pereira, Domain adaptation with structural correspondence learning, in: Proceedings of the Conference on Empirical Methods in Natural Language Processing, 2006, pp. 120–128.
[23] X. Zhang, W. Dai, G. Xue, Y. Yu, Adaptive Email Spam Filtering based on Information Theory, in: Proceedings of the Eighth International Conference on Web Information Systems Engineering (WISE), 2007, pp. 59–170.
[24] A.R. Gray, S.G. MacDonnell, A Comparison of techniques for developing predictive models of software metrics, Information and Software Technology 39 (6) (1997) 425–437.
[25] V.K. Vaishnavi, S. Purao, J. Liegle, Object-oriented product metrics: a generic-framework, Information Sciences 177 (2) (2007) 587–606.
[26] J. Al Dallal, L. Briand, An Object-Oriented High-Level Design-Based Class Cohesion Metric, TR, Simula Research Laboratory, 2009.
[27] Y. Liu, T.M. Khoshgoftaar, N. Seliya, Evolutionary optimization of software quality modeling with multiple repositories, IEEE Transactions on Software Engineering 36 (6) (2010) 852–864.
[28] R. Duda, P. Hart, N. Nilsson, Subjective bayesian methods for rule-based inference systems, Technical Report 124, Artificial Intelligence Center, SRI International, 1976.
[29] L. Peng, B. Yang, Y. Chen, A. Abraham, Data gravitation based classification, Information Sciences 179 (6) (2009) 809–819.
[30] C. Wang, Y.Q. Chen, Improving nearest neighbor classification with simulated gravitational collapse, Advances in Natural Computation, Lecture Motes in Computer Science 3612 (2005) 845–854.
[31] M. Indulska, M.E. Orlowska, Gravity based spatial clustering, in: Proceedings of the 10th ACM International Symposium on Advances in Geographic Information Systems, 2002, pp. 125–130.
[32] I. Newton, Philosophiae Naturalis Principia Mathematica, first ed., Royal Society, London, 1687.
[33] E. Frank, M. Hall, B. Pfahringer, Locally Weighted Naive Bayes, in: Proceedings of the Conference on Uncertainty in Artificial Intelligence, 2003, pp. 249–256.
[34] U.M. Fayyad, K.B. Irani, Multi-interval discretization of continuous-valued attributes for classification learning, In: Proceedings of the Thirteenth International Joint Conference on Articial Intelligence, 1993, pp. 1022–1027.
[35] I.H. Witten, E. Frank, Data Mining: Practical Machine Learning Tools and Techniques, second ed., Morgan Kaufman, San Francisco, 2005.
[36] T. Fawcett, An introduction to ROC analysis, Pattern Recognition Letters 27 (8) (2006) 861–874.
[37] C.J. van Rijsbergen, Information Retireval, second ed., Butterworth, London, 1979.
[38] F. Wilcoxon, Individual comparisons by ranking methods, Biometrics Bulletin 1 (6) (1945) 80–83.