Int. J. Mach. Learn. & Cyber. (2017) 8:355–370
DOI 10.1007/s13042-015-0328-7

ORIGINAL ARTICLE

Semi-supervised self-training for decision tree classifiers

Jafar Tanha · Maarten van Someren · Hamideh Afsarmanesh

Received: 7 May 2014 / Accepted: 5 January 2015 / Published online: 24 January 2015

Abstract We consider semi-supervised learning, learning task from both labeled and unlabeled instances and in particular, self-training with decision tree learners as base learners. We show that standard decision tree learning as the base learner cannot be effective in a self-training algorithm to semi-supervised learning. The main reason is that the basic decision tree learner does not produce reliable probability estimation to its predictions. Therefore, it cannot be a proper selection criterion in self-training. We consider the effect of several modifications to the basic decision tree learner that produce better probability estimation than using the distributions at the leaves of the tree. We show that these modifications do not produce better performance when used on the labeled data only, but they do benefit more from the unlabeled data in self-training. The modifications that we consider are Naive Bayes Tree, a combination of No-pruning and Laplace correction, grafting, and using a distance-based measure. We then extend this improvement to algorithms for ensembles of decision trees and we show that the ensemble learner gives an extra improvement over the adapted decision tree learners.

Keywords Semi-supervised learning · Self-training · Ensemble learning · Decision tree learning

J. Tanha (✉) · M. van Someren · H. Afsarmanesh
Informatics Institute, University of Amsterdam,
Science Park 904, 1098 XH Amsterdam, The Netherlands
e-mail: jafar.tanha.pnu@gmail.com; jafar.tanha@inl.nl

M. van Someren
e-mail: M.W.vanSomeren@uva.nl

H. Afsarmanesh
e-mail: h.afsarmanesh@uva.nl

1 Introduction

Supervised learning methods are effective when there are sufficient labeled instances. In many applications, such as object detection, document and web-page categorization, labeled instances however are difficult, expensive, or time consuming to obtain, because they require empirical research or experienced human annotators. Semi-supervised learning algorithms use not only the labeled data but also unlabeled data to construct a classifier. The goal of semi-supervised learning is to use unlabeled instances and combine the information in the unlabeled data with the explicit classification information of labeled data for improving the classification performance. The main issue of semi-supervised learning is how to exploit information from the unlabeled data. A number of different algorithms for semi-supervised learning have been presented, such as the Expectation Maximization (EM) based algorithms [30, 35], self-training [25, 33, 34, 45], co-training [6, 37], Transductive Support Vector Machine (TSVM) [23], Semi-Supervised SVM (S3VM) [4], graph-based methods [2, 48], and boosting based semi-supervised learning methods [27, 38, 40].

Self-training is a commonly used method to semi-supervised learning in many domains, such as Natural Language Processing [33, 41, 45] and object detection and recognition [34]. A self-training algorithm is an iterative method for semi-supervised learning, which wraps around a base learner. It uses its own predictions to assign labels to unlabeled data. Then, a set of newly-labeled data, which we call a set of high-confidence predictions, are selected to be added to the training set for the next iterations. The performance of the self-training algorithm strongly depends on the selected newly-labeled data at each iteration of the training procedure. This selection strategy is based on confidence in the predictions and therefore it is vital to self-training that the confidence of prediction, which we will call here probability estimation, is measured correctly. There is a difference between learning algorithms that output a probability distribution, e.g. Bayesian methods, neural networks, logistic regression, margin-based classifiers, and algorithms that are normally seen as only outputting a classification model, like decision trees. Most of the current approaches to self-training utilize the first kind of learning algorithms as the base learner [25, 33, 34, 45]. In this paper we focus on self-training with a decision tree learner as the base learner. The goal is to show how to effectively use a decision tree classifier as the base learner in self-training.

In a decision tree classifier the class distribution at the leaves is normally used as probability estimation for the predictions. We will show that using this as the selection metric in self-training does not improve the classification performance of a self-training algorithm and thus the algorithm does not benefit from the unlabeled data. The reason is that the decision tree classifier cannot produce a good ranking for its predictions using only the class distribution at the leaves of the tree. It can be seen that decision trees as the base learner in self-training involves two main difficulties to produce a good ranking of instances. These include: (1) the sample size at the leaves is almost always small, there is a limited number of labeled data, and (2) all instances at a leaf get the same probability. However, decision trees are the best learning algorithm for the particular domains, see [31, 42]. This has motivated us to look for improvements of the probability estimation for decision tree learning when it is used as the base learner in self-training.

We first make several modifications to the basic decision tree learner that produce better probability estimations when it is used in self-training as the base learner. The proposed modifications are: (a) No-pruning and applying the Laplacian Correction (C4.4) [31], (b) Grafting [44], (c) a combination of Grafting with Laplacian Correction and No-pruning, (d) a Naive Bayes Tree classifier (NBTree) [24], (e) using a distance-based measure combined with the improved decision tree learners. Our hypothesis is that these modified decision tree learners will show classification accuracy similar to the standard decision tree learner when applied to the labeled data only, but will benefit from the unlabeled data when used as the base classifier in self-training, because they make better probability estimates which is vital for the selection step of self-training. We then extend our analysis from single decision trees to ensembles of decision trees, in particular the Random Subspace Method [19] and Random Forests [9]. In this case, probability is estimated by combining the predictions of multiple trees. However, if the trees in the ensemble suffer from poor probability estimation, the ensemble learner will not benefit much from self-training on unlabeled data. Using the modified decision tree learners as the base learner for the ensemble will improve the performance of self-training with the ensemble classifier as the base learner. The results of the experiments on the several benchmark datasets confirm this. We perform several statistical tests to show the effect of the proposed methods.

The rest of this paper is organized as follows. Section 2 reviews related work on semi-supervised learning. Section 3 addresses the decision tree classifiers as the base learner in self-training. Sections 4 and 5 address the improvements for self-training. The experimental setup is presented in Sect. 6. Sections 7 and 8 present the results of the experiments and Sect. 9 addresses the conclusions.

2 Related work

There are several different methods for semi-supervised learning. The generative approach is one of the well-known semi-supervised learning methods. It uses Expectation Maximization (EM) [11] with generative mixture models, for example a mixture of Gaussians [35]. One issue in EM when it is used in semi-supervised learning is that maximizing the likelihood may not lead to the optimal classification performance [34]. Another approach is to extend margin-based methods to semi-supervised learning. Examples of this approach are the Transductive Support Vector Machine (TSVM) [23] and Semi-Supervised SVM method (S3VM) [4]. These methods use the unlabeled data to regularize the decision boundary. However, finding the exact decision boundary is an NP-hard problem. Furthermore, recent approaches to margin-based methods cannot solve semi-supervised classification problems with more than a few hundred unlabeled examples. Recently, new approaches have been presented based on graphs, such as manifold regularization [2], and boosting, like MSAB [40]. The effect of these methods depends strongly on the similarity function, because selecting and tuning a similarity function makes the approach expensive [39, 48].

Self-training has been applied to several natural language processing tasks. Yarowsky [45] uses self-training for word sense disambiguation. A self-training algorithm is used to recognize different nouns in [33]. Maeireizo et al. [26] propose a self-training algorithm to classify dialogues as "emotional" or "non-emotional" with a procedure involving two classifiers. In [41] a semi-supervised self-training approach using a hybrid of Naive Bayes and decision trees is used to classify sentences as subjective or objective.

Rosenberg et al. [34] proposed a self-training approach to object detection using an existing object detector based on the Nearest Neighbor classifier. The study shows that a model trained on a small set of labeled instances can achieve results comparable to a model trained in the supervised manner using a larger set of labeled data.

Li et al. [25] propose a self-training semi-supervised support vector machine algorithm and a selection metric, which are designed for learning from a limited number of training data. Two examples show the validity of the algorithm and selection metric on a data set collected from a P300-based brain computer interface speller. This algorithm is shown to significantly reduce training effort.

In general, self-training is a wrapper algorithm, and is hard to analyze. However, for specific base classifiers, theoretical analysis is feasible, for example [17] showed that the Yarowsky algorithm [45] minimizes an upper-bound on a new definition of cross entropy based on a specific instantiation of the Bregman distance. In this paper, we focus on using a decision tree learner as the base learner in self-training. We show that improving the probability estimation of the decision trees will improve the performance of a self-training algorithm.

3 Semi-supervised self-training with decision trees

In this section, we first define the Semi-supervised setting and then address the semi-supervised self-training algorithm.

3.1 Semi-supervised setting

In semi-supervised learning there is a small set of labeled data and a large pool of unlabeled data. Data points are divided into the points $X_l = (x_1,x_2...,x_l)$, for which labels $Y_l = \{+1,-1\}$ are provided, and the points $X_u = (x_{l+1},x_{l+2},\ldots,x_{l+u})$, the labels of which are not known. We assume that labeled and unlabeled data are drawn independently from the same data distribution.

In this paper we consider datasets for which $n_l \ll n_u$, where n_l and n_u are the number of labeled data and unlabeled data respectively.

3.2 The self-training algorithm

The self-training algorithm wraps around a base classifier and uses its own predictions through the training process [48]. A base learner is first trained on a small number of labeled examples, the initial training set. The classifier is then used to predict labels for unlabeled examples (prediction step) based on the classification confidence. Next, a subset S of the unlabeled examples, together with their predicted labels, is selected to train a new classifier (selection step). Typically, S consists of a few unlabeled examples with high-confidence predictions. The classifier is then re-trained on the new set of labeled examples, and the procedure is repeated (re-training step) until it reaches a stopping condition. As a base learner, we employ the decision tree classifier in self-training. The most well-known algorithm for building decision trees is the C4.5 algorithm [32], an extension of Quinlan's earlier ID3 algorithm. Decision trees are one of the most widely used classification methods, see [5, 10, 16]. They are fast and effective in many domains. They work well with little or no tweaking of parameters which has made them a popular tool for many domains [31]. This has motivated us to find a semi-supervised method for learning decision trees. Algorithm 1 presents the main structure of the self-training algorithm.

Algorithm 1 Outline of the Self-Training algorithm

```
Initialize: L, U, F, $T$ ; L: Labeled data; U: Unlabeled data;
F : Underlying classifier; $T$ : Threshold for selection;
$Iter_{max}$ : Number of iterations; $\{P_l\}_{l=1}^{M}$: Prior probability;
$t \leftarrow 1$;
while (U != empty) and ($t < Iter_{max}$ ) do
    - $H^{t-1} \leftarrow BaseClassifier(L, F)$;
    for each $x_i \in U$ do
        - Assign pseudo-label to $x_i$ based on classification confidence
    - Sort Newly-Labeled examples based on the confidence
    - Select a set $S$ of the high-confidence predictions according to $n_l \propto P_l$
      and threshold $T$ // Selection Step
    - Update U = U - S; L = L U S;
    - $t \leftarrow t + 1$
    - Re-Train $H^{t-1}$ by the new training set $L$
end while
Output: Generate final hypothesis based on the new training set
```

The goal of the selection step in Algorithm 1 is to find a set unlabeled examples with high-confidence predictions, above a threshold T. This is important, because selection of incorrect predictions will propagate to produce further classification errors. At each iteration the newly-labeled instances are added to the original labeled data for constructing a new classification model. The number of iterations in Algorithm 1 depends on the threshold T and also on the pre-defined maximal number of iterations, $Iter_{max}$.

4 Self-training with improved probability estimates

The main difficulty in self-training is to find a set of high-confidence predictions of unlabeled data. Although for many domains decision tree classifiers produce good classifiers, they provide poor probability estimates [28, 31]. The reason is that the sample size at the leaves is almost always small, and all instances at a leaf get the same probability. The probability estimate is simply the proportion of the majority class at the leaf of a (pruned) decision tree. A trained decision tree indeed uses the absolute class frequencies of each leaf of the tree as follows:

$$p(k|x) = \frac{K}{N} \quad (1)$$

where K is the number of instances of the class k out of N instances at a leaf. However, these probabilities are based on very few data points, due to the fragmentation of data over the decision tree. For example, if a leaf node has subset of 50 examples of which 45 examples belong to one class, then any example that corresponds to this leaf will get 0.9 probability where a leaf with 3 examples of one class get a probability of 1.0. In semi-supervised learning this problem is even more serious, because in applications the size of initial set of labeled data is typically quite small.

Here, we consider several methods for improving the probability estimates at the leaves of decision trees.

4.1 C4.4: No-pruning and Laplacian correction

One candidate improvement is the Laplacian correction (or Laplace estimator) which smooths the probability values at the leaves of the decision tree [31]. Smoothing of probability estimates from small samples is a well-studied statistical problem [36]. Assume there are K instances of a class out of N instances at a leaf, and C classes. The Laplacian correction calculates the estimated probability P(class) as:

$$p(k|x) = \frac{K+1}{N+C} \quad (2)$$

Therefore, while the frequency estimate yields a probability of 1.0 from $K = 10$, $N = 10$ leaf, for a binary classification problem the Laplace estimate produces a probability of $(10+1)/(10+2) = 0.92$. For sparse data, the Laplacian correction at the leaves of a tree yields a more reliable estimation that is crucial for the selection step in self-training. Without it, regions with low density will show relatively extreme probabilities that are based on very few data points. These have a high probability of being used for self-training, which is problematic. Because the misclassified examples can get the high confidence by the classifier.

Another possible improvement is a decision tree learner that does not do any pruning. Although this introduces the risk of "overfitting", it may be a useful method because of the small amount of training data. If there are few training data, then pruning methods can easily produce *underfitting* and No-pruning avoids this. In applications of semi-supervised learning, "underfitting" is a potential problem. Although it produces even fewer data at leave nodes, No-pruning may therefore still provide better probability estimates, especially if combined with the Laplace correction [29, 31]. We therefore include this combination in our analysis as C4.4.

4.2 NBTree

The Naive Bayesian Tree learner, NBTree [24], combines the Naive Bayes Classifier with decision tree learning. In an NBTree, a local Naive Bayes Classifier is constructed at each leaf of decision tree that is built by a standard decision tree learning algorithm like C4.5. NBTree achieves in some domains higher accuracy than either a Naive Bayes Classifier or a decision tree learner.

NBTree classifies a test sample by passing it to a leaf and then using the naive Bayes classifier in that leaf to assign a class label to it. It also assigns the highest $P(k|X)$ for the test example $X \in R^n$ as the probability estimation, where k is a class variable [14].

4.3 Grafted decision tree

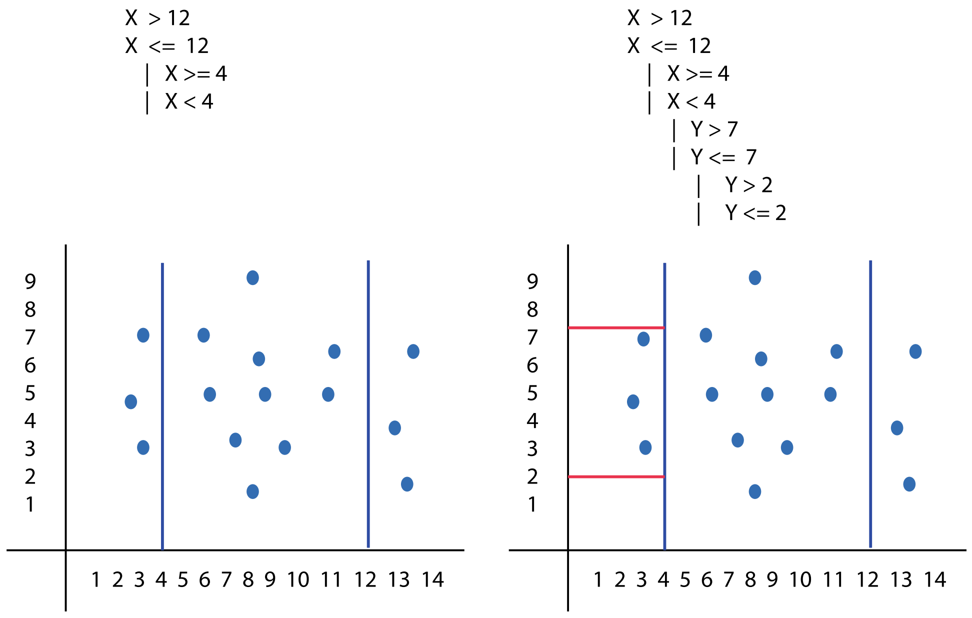
A grafted decision tree classifier generates a decision tree from a standard decision tree. The idea behind grafting is that some regions in the data space are more sparsely populated. The class label for sparse regions can better be estimated from a larger region. The grafting technique [44] searches for regions of the multidimensional space of attributes that are labeled by the decision tree but they contain no or very sparse training data. These regions are then split by the region that corresponds to a leaf and labeling the empty or sparse areas by the label of the majority above the previous leaf node. Consider the example in Fig. 1. It shows the resulting grafted tree. There are two cuts in the decision trees at nodes 12 and 4. After grafting, branches are added by the grafting technique. Grafting performs a kind of local "unpruning" for low-density areas. This can improve the resulting model, see [44]. In fact, grafted decision tree learning often gives better decision trees in case of sparse data and also improves the probability estimates. The probability estimates of the grafted decision tree is computed by (1).

4.4 Combining no-pruning, Laplace correction and grafting

We combine Grafting with the Laplacian correction and No-pruning, which we call C4.4graft. We expect that C4.4graft gives better decision tree than C4.5 in the case of sparse data and it also improves probability estimates due to using the Laplacian correction and No-pruning, see Sect. 7.1. C4.4graft computes the probability estimation as introduced in (2).

4.5 Global distance-based measure

In Algorithm 1, only the probability estimation is used to select high-confidence predictions, which may not always



(a) Standard Decision Tree **(b)** Grafted Decision Tree

Fig. 1 Grafted decision tree

be optimal because of some misclassified examples with high-confidence probability estimation. Another way to select from the unlabeled examples is to use a combination of distance-based approach and the probability estimation. We propose a selection metric based on a rather drastic step against the problem of data fragmentation for sample selection. There are several approaches to sample selection using decision trees [43]. We use all data for a global distance-based measure to sample selection. Specifically we subtract the difference in average Mahalanobis distance between an unlabeled data point and all positively labeled data, p_i, from that of the negatively labeled data, q_i, see Fig. 2 and Algorithm 2. The Mahalanobis distance measure differs from Euclidean distance in that it takes into account the correlation of the data. It is defined as follows:

$$D(X) = \sqrt{(X - \bar{X})^T S^{-1} (X - \bar{X})} \tag{3}$$

where $X = (X_1, \ldots, X_n) \in U$ is a multivariate vector, $\bar{X} = (\bar{X_1}, \ldots, \bar{X_n})$ is the mean, and S is *covariance matrix*. Then, the absolute value of the difference between p_i and q_i is calculated as a score for each unlabeled example, see Algorithm 2. These scores are used for selection metric. Algorithm 2 shows the procedure for selection metric. The labeled data are used to calculate the covariance. This measure uses all (labeled) data and thereby avoids errors in the probability estimates that are caused by the instability due to data fragmentation. Note that when $M = S^{-1}$ is the identity matrix, $D(X)$ gives the Euclidean distance. Otherwise, we can address M as $L^T L$ such that $L \in R^{k \times n}$ where k is the rank of M . Then (3) is reformulated as

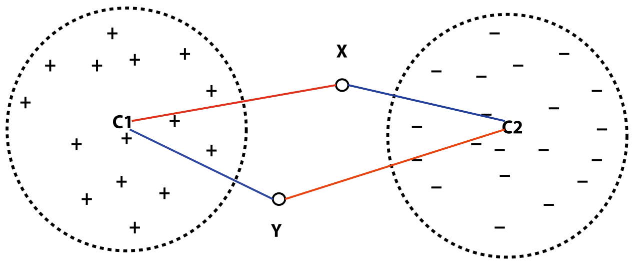


Fig. 2 Distance of unlabeled examples X and Y and positive and negative examples based on the Mahalanobis distance

$$D(X) = \sqrt{(LX - L\bar{X})^T M (LX - L\bar{X})} \tag{4}$$

which emphasizes that a Mahalanobis distance implicitly corresponds to computing the Euclidean distance after the linear projection of the data defined by the transformation matrix L. One important point in Mahalanobis distance is that if M is low-rank ($rank(M) = r < n$), then it transforms a linear projection of the data into a space of lower dimension, i.e. r [3]. However, there are some practical issues using Mahalanobis distance, for example the computation of the covariance matrix can cause problems. When the data consist of a large number of features and the limited number of the data points, they can contain much redundant or correlated information, which leads to a singular or nearly singular covariance matrix. This problem can be solved by using feature reduction or using pseudo-covariance [21].

Following the general self-training algorithm, a set of examples with highest score is selected from unlabeled examples according to Algorithm 2. This subset is then assigned "pseudo-labels" as in Algorithm 1. Next, the high-confidence predictions from this subset is added along with their "pseudo-labels" to the training set. This procedure is repeated until it reaches a stopping condition.

Algorithm 2 Selection Metric

```
Initialize: U, P;
U : Unlabeled examples; P: Number of selected examples;
for each $x_i \in U$ do
    - $p_i \leftarrow$ Calculate distance between $x_i$ and the mean of the positive examples;
    - $q_i \leftarrow$ Calculate distance between $x_i$ and the mean of the negative examples
    - $w_i \leftarrow |p_i - q_i|$ as score for each example;
$SubSet \leftarrow$ Select P% of the highest score examples;
return $SubSet$
```

5 Self-training for ensembles of decision trees

In this section we extend the analysis from decision trees to ensembles of decision trees. An ensemble combines many, possibly weak, classifiers to produce a (hopefully) strong classifier [13]. Ensemble methods differ in their base learner and in how classifiers are combined. Examples of the ensemble methods are bagging [8], boosting [15], Random Forest (RF) [9], and Random Subspace Method (RSM) [19]. In an ensemble classifier, probability estimation is estimated by combining the confidences of their components. This tends to improve both the classification accuracy and the probability estimation. However, if a standard decision tree learner is used as the base learner, then the problems, that we noted above, carry over to the ensemble. We therefore expect that improving the probability estimates of the base learner will enable the ensemble learner to benefit more from the unlabeled data than if the standard decision tree learner is used. Algorithm 3 shows the generation of the trees in an ensemble classifier. In the experiments we use Random Forest and the Random Subspace Method to generate ensembles of trees.

The RSM and RF ensemble classifiers are well-suited for data that are high-dimensional. A single decision tree minimizes the number of features that are used in the decision tree and does not exploit features that are correlated and all have little predictive power. The ensemble classifiers do not suffer from this problem because the random component and the ensemble allow including more features.

5.1 Random forest

A Random Forest (RF) is an ensemble of n decision trees. Each decision tree in the forest is trained on different subsets of the training set, generated from the original labeled data by bagging [8]. Random Forest uses randomized feature selection while the tree is growing. In the case of multidimensional datasets this property is indeed crucial, because when there are hundreds or thousand features, for example in medical diagnosis and documents, many weakly relevant features may not appear in a single decision tree at all. The final hypothesis in Random Forest is produced by using majority voting method among the trees. Many semi-supervised algorithms have been developed based on the Random Forest approach, such as Co-Forest [47].

There are several ways to compute the probability estimation in a random forest such as averaging class probability distributions estimated by the relative class frequency, the Laplace estimate and the m-estimate respectively. The standard random forest uses the relative class frequency as its probability estimation which is not suitable for self-training as we discussed. As a result we use the improved base classifier C4.4graft as a base learner in random forest.

Let's denote the ensemble H as $H = \{h_1, h_2, \ldots, h_N\}$, where N is the number of trees in the forest. Then, the probability estimation for predicting example x is defined as

$$\underset{k}{\operatorname{argmax}} \; P(k|x) \tag{5}$$

where

$$P(k|x) = \frac{1}{N}\sum_{i=1}^{N} P_i(k|x) \tag{6}$$

and k is the class label and $P_i(k|x)$ is the probability estimate of the i-th tree for sample x which is computed by (2).

5.2 The random subspaces method

A Random Subspace method [19] is an ensemble method that combines randomly chosen feature subspaces of the original feature space. In the Random Subspaces method instead of using a subsample of data points, subsampling is performed on the feature space. The Random Subspaces method constructs a decision forest that maintains highest accuracy on training data and improves on generalization accuracy as it grows in complexity.

Assume that the ith tree of the ensemble be defined as $h_i(X, S_i) : X \mapsto K$, where X are the data points, K are the labels, and the S_i are independent identically distributed (i.i.d) random vectors. Let's define the ensemble classifier H as $H = \{h_1, h_2, \ldots, h_N\}$, where N is the number of trees in the forest. The probability estimation is then computed as in (5), where $P_i(k|x)$ is the probability estimate of the i-th tree for sample x. This probability estimation cab be computed by NBTree, C4.4, or C4.4graft.

5.3 The ensemble self-training algorithm

Beside modifying the decision tree learner, probability estimates can be improved by constructing an ensemble of decision trees and using their predictions for probability estimates. We use the modified decision tree learners as base learners and construct the ensemble following $EnsembleTrees(L, F, N)$. This is then used as the base learner in Algorithm 1, where N is the number of trees. In the self-training process, first the decision trees are generated by bagging or the random subspace method using our proposed decision trees, see Algorithm 3. The ensemble classifier then assigns "pseudo-labels" and confidence to the unlabeled examples at each iteration. Labeling is performed by using different voting strategies, such as majority voting or average probability as in (5). The training process is given in Algorithm 1. The selection metric in this algorithm is based on the ensemble classifier, which improves the probability estimation of the trees.

Algorithm 3 Ensemble of Decision Trees

```
Initialize: L, F, N ; L: Labeled data;
F : Base classifier;// This base classifier is one of the proposed method in this paper.
N: Number of trees;
for i=1 to N do
    - $L_i$ ←BootstrapSample(L)// or RandomSubSpaceSample(L);
    - $h_i \leftarrow F(L_i)$
    - $H \leftarrow H + h_i$
Output: Generate ensemble $H$
```

6 Experiments

In the experiments we compare the performance of the supervised decision tree learning algorithms to self-training. We use the following decision tree classifiers as the base classifier in self-training: J48 (the Java implementation of C4.5), C4.4, NBTree, C4.4graft, J48graft, and ensemble of trees. For our experiments we use the WEKA [18] implementation of the classifiers in Java. We expect to see that the decision tree learners that give better probability estimates for their predictions will benefit more from unlabeled examples. We then make the same comparison for ensemble learners. We further compare the performance of self-training with distance-based selection metric to other algorithms.

In the experiments, for each dataset 30 % of the data are kept as test set, and the rest is used as training data. Training data in each experiment are first partitioned into 90 % unlabeled data and 10 % labeled data, keeping the class proportions in all sets similar to the original data set. We run each experiment 10 times with different subsets of training and testing data. The results reported refer to the test set. To provide a statistical basis for the main comparisons we use the following statistical tests.

6.1 Statistical test

We compare the performance of the proposed algorithms by the method in [12]. We first apply Friedman's test as a nonparametric test equivalent to the repeated measures ANOVA. Under the null hypothesis Friedman's test states that all the algorithms are equal and the rejection of this hypothesis implies differences among the performance of the algorithms. It ranks the algorithms based on their performance for each dataset separately, it then assigns the rank 1 to the best performing algorithm, rank 2 to the second best, and so on. In case of ties average ranks are assigned to the related algorithms. Let's define r_i^j as the rank of the jth algorithm on the ith datasets. The Friedman test compares the average ranks of algorithms, $R_j = \frac{1}{N}\sum_i r_i^j$, where N is the number of datasets and k is the number of the classifiers. Friedman's statistic

$$\chi_F^2 = \frac{12N}{k(k+1)}\left[\sum_j R_j^2 - \frac{k(k+1)^2}{4}\right] \tag{7}$$

is distributed according to χ_F^2 with $k-1$ degrees of freedom. Iman and Davenport [22] showed that Friedman's χ_F^2 is conservative. Then they present a better statistic based on Friedman's test:

$$F_F = \frac{(N-1)\chi_F^2}{N(k-1)-\chi_F^2} \tag{8}$$

This statistic is distributed according to the *F-distribution* with $k-1$ and $(k-1)(N-1)$ degrees of freedom.

As mentioned earlier, Friedman's test shows whether there is a significant difference between the averages or not. The next step for comparing the performance of the algorithms with each other is to use a post-hoc test. We use Holm's method [20] for post-hoc tests. This sequentially checks the hypothesis ordered by their performance. The ordered p-values are denoted by $p_1 \le p_2 \le \ldots \le p_{k-1}$. Then each p_i is compared with $\frac{\alpha}{(k-i)}$, starting from the most significant p-value. If p_1 is less than $\frac{\alpha}{(k-1)}$, then the corresponding hypothesis is rejected and p_2 is compared with $\frac{\alpha}{(k-2)}$. As soon as a certain hypothesis cannot be rejected, all remaining averages are taken as not significantly different. The test statistic for comparing two algorithms is

$$z = \frac{R_i - R_j}{\sqrt{\frac{k(k+1)}{6N}}} \tag{9}$$

The value z is used to find the corresponding probability from the normal distribution and is compared with the corresponding value of α. We use $\alpha = 0.05$.

6.2 UCI datasets

We use a number of UCI datasets [1] to evaluate the performance of our proposed methods. Recently the UCI datasets have been extensively used for evaluating semi-supervised learning methods. Consequently, we adopt UCI

Table 1 Overview of UCI datasets

Dataset (classes)	Attributes	Size	Perc.
Breath-cancer (1,2)	10	286	70
Bupa (1,2)	6	345	58
Car (1,2)	7	1,594	76
Cmc (1,3)	10	1,140	55
Colic (1,2)	22	368	63
Diabetes (1,2)	6	768	65
Heart statlog (1,2)	13	270	55
Hepatitis (1,2)	19	155	79
Ionosphere (1,2)	34	351	36
Liver (1,2)	7	345	58
Sonar (1,2)	61	208	53
Tic-tac-toe (1,2)	9	958	65
Vote (1,2)	16	435	61
Wave (1,2)	41	3,345	51

datasets to assess the performance of the proposed algorithms. Fourteen datasets from the UCI repository are used in our experiments. We selected these datasets, because they differ from the number of features and examples, and distribution of classes. Information about these datasets is in Table 1. All sets have two classes and Perc. represents the percentage of the largest class.

6.3 Web-pages datasets

The datasets from the UCI Repository have a relatively small number of attributes. We performed additional experiments on image classification data with a somewhat larger set of attributes. The task here is to learn to classify web pages by their visual appearance as in [7]. Specifically the task is to recognize *aesthetic* value a (*ugly* vs. *beautiful*) and the *recency* of a page (*old* vs. *new*) from simple color and edge histograms, Gabor and texture attributes. These labels were assigned by human evaluators with a very high inter-rater agreement. Information about these datasets is in Table 2. In these data ensembles of decision trees are more effective than single decision trees because typically a large number of attributes is relevant and the decision tree learners minimize the number of attributes in the classifier. All datasets have two classes and Prc. represents the percentage of the largest class label.

7 Results

Tables 3, 6, 7, and 8 give the classification accuracies for the experiments. For each base classifier, the performance of the supervised learning only on labeled data and self-training on both labeled and unlabeled data are reported.

7.1 Self-training with a single classifier

Table 3 compares the results of the standard decision tree learner J48 (DT) to its self-training version (ST-DT) and the same for the C4.4, grafting (GT), the combination of grafting and C4.4 (C4G), and the Naive Bayes Decision trees (NBTree). We expect that the modified algorithms show results similar to J48 when only labeled data are used, but improved classification accuracies when they are used as the base learner in self-training.

7.1.1 Self-training with J48 decision tree learner

In Table 3, the columns DT and ST-DT show the classification accuracy of J48 base learner and self-training respectively. As can be seen, self-training does not benefit from unlabeled data and there is no difference in accuracy between learning from the labeled data only and self-training from labeled and unlabeled data. The average improvement over all datasets is 0.15 %.

7.1.2 Self-training with C4.4, grafting, and NBTree

Table 3 gives the classification accuracy of C4.4 and ST-C4.4. As can be seen, unlike the basic decision tree learner, C4.4 enables self-training to become effective for nearly all the datasets. The average improvement over the used datasets is 1.9 %. The reason for improvement is that using Laplacian correction and No-pruning give better rank for probability estimation of the decision tree, which leads to select a set of high-confidence predictions.

In Table 3, we can see the same observation for J48graft (grafted decision tree). When using only labeled data we see no difference with the standard algorithm but self-training improves the performance of the supervised J48graft on all datasets. The average improvement over all datasets is 1.6 %. Next, we combine Laplacian correction and No-pruning in J48graft, C4.4graft. This modification in grafted decision tree gives better results when it is used as the base learner in self-training. The results show that self-training outperforms the supervised C4.4graft classifier on all datasets. A t-test on the results shows that self-training significantly improves the classification performance of C4.4graft classifier on 10 out of 14 datasets and the average improvement over all datasets is 3.5 %. Finally, the results of experiments on NBTree classifier as the base learner in self-training show that it improves the performance of NBTree classifier on 13 out of 14 datasets and the average improvement is 2.7 %.

7.1.3 *Statistical analysis*

In this section, we analyze the results in Table 3. Table 4 shows the rank of each algorithm for each dataset according to Friedman's test. Average ranks are reported in the last row. Friedman's test checks whether the measured average ranks are significantly different from the mean rank $R_j = 2.96$ expected under the null-hypothesis. Then, according to the Eqs. (7) and (8), $\chi^2_F = 39.54$ and $F_F = 31.24$.

With five algorithms and 14 datasets, F_F follows the *F-distribution* with $5-1=4$ and $(5-1)(14-1)=39$ degrees of freedom. The critical value of $F(4,39)$ for $\alpha = 0.05$ is 2.61, so we reject the null-hypothesis. Next we apply Holm's test, see Table 5.

Table 5 shows the results. The Holm procedure rejects the first, second, third, and then fourth hypotheses, since the corresponding p values are smaller than the adjusted $\alpha's$ (0.05). We conclude that the performance of C4.4graft, NBTree, J48graft, and C4.4 as the base learner in self-training algorithm are significantly different from standard decision tree (J48).

7.2 Self-training with single classifier and global distance-based measure

To evaluate the impact of the global distance-based selection metric for self-training (as in Algorithm 2), we run another set of experiments. Table 6 shows the results of the experiments. The columns ST-DT, ST-C4.4, ST-GT, ST-C4G, and ST-NB show the classification performance of self-training with J48, C4.4, J48graft, C4.4graft, and NBTree as the base learners respectively.

The results show that in general using the Mahalanobis distance as the selection metric improves the classification performance of all self-training algorithms. For example, comparing the results of self-training in Tables 3 and 6 when the base classifier is C4.4graft, we observe that self-training in Table 6 except for web-pages datasets, using distance-based selection metric, outperforms the self-training in Table 3 on 10 out of 14 datasets. The same results are seen for J48 as the base learner.

The results on the web page classification task show the same pattern. Note that Self-Training with web page classification starts with only six labeled examples. The results do not seem to depend on the number of variables or the nature of the domain.

7.3 Self-training with an ensemble of trees

In this experiment, we expect that the ensemble learner will perform somewhat better than the basic decision tree learner, when used on the labeled data only. More interesting thought, we expect that the ensemble, with improved base learner, can improve the probability estimation and therefore if it is used as the base learner in self-training it will benefit even more from the unlabeled data than a single modified decision tree learner.

We use C4.4graft as the base classifier in RF. REPTree, NBTree, and C4.4graft classifiers are also used in a RSM ensemble classifier as the base classifiers. REPTree is a fast decision tree learner. It builds a decision tree using information gain and prunes it using reduced-error pruning

Table 2 Overview of web-pages datasets

Dataset	Attributes	Size	Perc.
Aesthetic	192	60	50
Recency	192	60	50

Table 3 Average classification accuracy of supervised learning and self-training with different base classifiers

Dataset	DT	ST-DT	C4.4	ST-C4.4	GT	ST-GT	C4G	ST-C4G	NB	ST-NB
Breath-cancer	68.00	69.00	66.25	67.00	68.00	70.01	66.25	70.12	72.50	75.75
Bupa	58.62	57.09	58.62	58.68	58.62	59.25	58.62	61.40	58.20	58.20
Car	86.08	86.04	85.48	87.48	86.08	87.28	85.48	88.28	85.08	87.68
Cmc	57.00	58.25	56.75	59.05	57.00	59.13	56.75	60.12	54.25	58.00
Colic	72.83	72.36	70.56	73.70	72.84	74.80	70.56	75.03	74.60	76.71
Diabetes	67.82	67.83	67.51	69.18	68.46	69.40	68.14	71.79	71.14	72.59
Heart	67.27	67.27	68.63	70.50	67.27	69.10	68.63	72.12	71.81	73.85
Hepatitis	76.00	75.60	76.00	76.40	76.00	76.60	76.00	80.60	78.40	82.40
Ionoshere	70.47	70.67	70.47	71.46	70.37	71.56	70.37	73.72	79.97	82.57
Liver	56.80	56.60	57.00	60.80	57.00	59.80	57.00	59.98	57.00	59.90
Sonar	63.40	63.40	63.40	63.76	63.40	64.92	63.40	65.40	59.60	63.60
Tic-tac-toe	66.40	68.20	63.40	68.80	66.40	70.10	63.80	69.60	65.20	68.60
Vote	89.08	89.08	89.05	90.30	89.08	89.80	88.05	90.48	90.00	92.74
Wave	83.10	83.63	82.85	85.13	84.10	85.25	83.60	86.25	84.75	88.00

Table 4 Statistical rank (Friedman's test)

Datasets	Decision tree	C4.4	J48graft	C4.4graft	NBTree
Breath-cancer	4	5	3	2	1
Bupa	5	3	2	1	4
Car	5	3	4	1	2
Cmc	5	3	2	1	4
Colic	5	4	3	2	1
Diabetes	5	4	3	2	1
Heart	5	3	4	2	1
Hepatitis	5	4	3	2	1
Ionosphere	5	4	3	2	1
Liver	5	3	4	1	2
Sonar	5	3	2	1	4
Tic-tac-toe	5	3	1	2	4
Vote	5	3	4	2	1
Wave	5	4	3	2	1
Average rank	4.93	3.50	2.93	1.64	1.93

Table 5 Statistical rank (Holm's test)

i	Classifier	Z	P-value	$\alpha/(k-i)$
1	C4.4graft	5.498051603	0.00000004	0.0125
2	NBTree	4.780914437	0.00000174	0.016666667
3	j48graft	3.82473155	0.00013092	0.025
4	C4.4	2.031888636	0.0421658	0.05

(with backfitting). REPTree is the default decision tree learner for the Random Subspace Method in WEKA.

Table 7 gives the classification accuracies of the all experiments. The columns RFG, RREP, RG, and RNB show the classification performance of supervised classifiers Random Forest with C4.4graft and RSM with REPTree, C4.4graft, and NBTree respectively and their corresponding self-training algorithms. Using RF with C4.4graft as the base learner in self-training improves the classification performance of the supervised classifier RF, on 13 out of 14 datasets. As can be seen, the results are better than a single decision tree, but in most cases the differences are not significant. We suspect that this is due to using the bagging method for generating different training set in Random Forest. In the case of a small set of labeled data, bagging does not work well [37], because the pool of labeled data is too small for re-sampling. However the average improvement is 1.9 % over all datasets.

In the second experiment, we use the RSM ensemble classifier with improved versions of decision trees. We observe that using C4.4graft and NBTree as the base classifiers in RSM is more effective than using REPTree, when RSM is used as the base learner in self-training. The results show that RSM with C4.4graft as the base learner in self-training improves the classification performance of RSM on 13 out of 14 datasets and the average improvement over all datasets is 2.7 %. The same results are shown in Table 7 for RSM, when the NBTree is the base learner.

7.4 Self-training with ensemble classifier and distance-based measure

Table 8 shows the results of the experiments using distance-based selection metric. The columns ST-RFG, ST-RREP, ST-RG, and ST-RNB show the classification performance of self-training with RF (ST-RFG as the base learner), RSM when the base learners are REPTree, C4.4graft, and NBTree respectively. The results, as in single classifier, show that in general using the Mahalanobis distance along with the probability estimation as the selection metric improves the classification performance of all self-training algorithms and emphasizes the effectiveness of this selection metric.

Results in Table 8 also show that ensemble RSM classifier with NBTree and C4.4graft, as the base learners, achieves the best classification performance on web-pages datasets. Finally, comparing Tables 6–8, shows that ensemble methods outperform the single classifier especially for web-page datasets. The results also verify that improving both the classification accuracy and the probability estimates of the base learner in self-training are effective for improving the performance.

7.5 Sensitivity to the amount of trees

We also study the effect on performance of the number of trees when using ensemble methods RF and RSM as the base learner of self-training. Figure 3 shows the accuracy of self-training with varying numbers of trees on two web-pages datasets. As we expect overall the classification accuracy is improved with increasing the number of trees.

7.6 Sensitivity to the amount of unlabeled data

To study the sensitivity of the proposed algorithms to the number of labeled data, we run a set of experiments with different proportions of labeled data which vary from 10 to 40 % on web-pages datasets. We expect that the difference between the supervised algorithms and the semi-supervised methods decreases when more labeled data are available. Figure 4 shows the performance of self-training with different base learners on two web-pages datasets. In this experiment, we use RF and RSM with C4.4graft as the base learner in self-training. For fair comparison we include a set of experiments with single classifiers, J48 and C4.4graft, as

Table 6 Average performance of supervised learning and self-training using the global Mahalanobis distance selection metric

Dataset	Supervised learning					Self-training				
	DT	C4.4	GT	C4G	NB	ST-DT	ST-C4.4	ST-GT	ST-C4G	ST-NB
Breath-cancer	68.00	66.25	68.00	66.25	72.50	70.07	70.12	70.25	71.27	76.14
Bupa	58.62	58.62	58.62	58.62	58.20	61.60	61.60	61.80	62.37	61.76
Car	86.08	85.48	86.08	85.48	85.08	87.04	87.56	87.28	89.01	88.04
Cmc	57.00	56.75	57.00	56.75	54.25	59.00	60.01	60.77	61.01	60.00
Colic	72.83	70.56	72.84	70.56	74.60	74.29	75.17	75.49	76.25	77.34
Diabetes	67.82	67.51	68.46	68.14	71.14	69.80	70.80	70.90	71.20	72.40
Heart	67.27	68.63	67.27	68.63	71.81	68.50	71.61	70.27	71.63	74.00
Hepatitis	76.00	76.00	76.00	76.00	78.40	77.12	77.29	78.71	80.97	82.40
Ionosphere	70.47	70.47	70.37	70.37	79.79	72.62	71.61	72.29	73.43	82.15
Liver	56.80	57.00	57.00	57.00	57.00	57.28	60.88	61.10	60.80	58.32
Sonar	63.40	63.40	63.40	63.40	59.60	64.12	65.00	65.10	66.43	64.31
Tic-tac-toe	66.40	63.40	66.40	63.80	65.20	67.40	66.02	68.40	67.33	67.17
Vote	89.08	89.05	89.08	88.05	90.00	90.43	91.37	92.12	92.18	92.15
Wave	83.10	82.85	84.10	83.60	84.75	84.63	85.13	85.67	86.75	88.00
Aesthetics	53.47	54.12	54.40	54.48	54.40	57.70	57.87	56.70	59.22	60.45
Recency	65.27	66.01	66.70	67.50	67.00	68.08	70.71	70.27	72.47	74.36

Table 7 Average performance of supervised learning and self-training with ensemble classifiers

Dataset	RFG	ST-RFG	RREP	ST-RREP	RG	ST-RG	RNB	ST-RNB
Breath-cancer	66.25	68.75	68.50	69.50	68.50	71.50	74.50	75.50
Bupa	57.34	60.32	56.45	57.72	55.04	59.38	58.40	58.40
Car	87.00	88.32	77.60	80.40	80.60	83.02	78.00	80.80
Cmc	60.25	63.25	58.75	59.63	59.50	63.70	57.25	59.38
Colic	75.00	74.90	67.32	71.65	77.50	79.60	76.77	79.26
Diabetes	69.56	71.66	70.66	70.75	67.82	70.56	70.04	72.29
Heart	74.99	76.22	72.04	74.58	73.18	76.09	70.91	73.40
Hepatitis	80.00	80.80	80.00	80.00	80.40	81.80	79.60	82.00
Ionoshere	80.00	82.00	71.20	73.80	73.04	77.76	78.31	81.10
Liver	56.60	58.14	56.00	58.40	61.40	63.00	56.80	58.00
Sonar	63.60	67.20	59.20	60.60	63.40	64.80	59.80	61.20
Tic-tac-toe	70.00	71.40	67.20	67.60	69.60	69.60	68.20	70.40
Vote	91.25	93.25	88.78	90.25	89.00	93.00	88.78	92.50
Wave	86.00	88.50	85.75	86.75	87.25	89.50	88.75	89.75

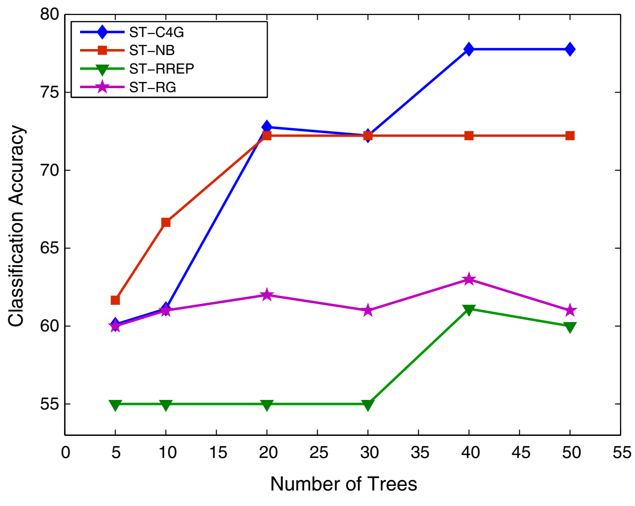
well. Figure 4 shows the performance obtained by self-training algorithms and supervised classifiers. Figure 4a, b show the performance of self-training with ensemble classifiers and Fig. 4c, d give the performance of self-training with single classifiers on web-pages datasets. Consistent with our hypothesis we observe that difference between supervised algorithms and self-training methods decreases when the number of labeled data increases. Another interesting observation is that RF improves the classification performance of self-training when more labeled data are available, because with more labeled data the bagging approach, used in RF, generates diverse decision trees.

7.7 Sensitivity to the threshold parameter

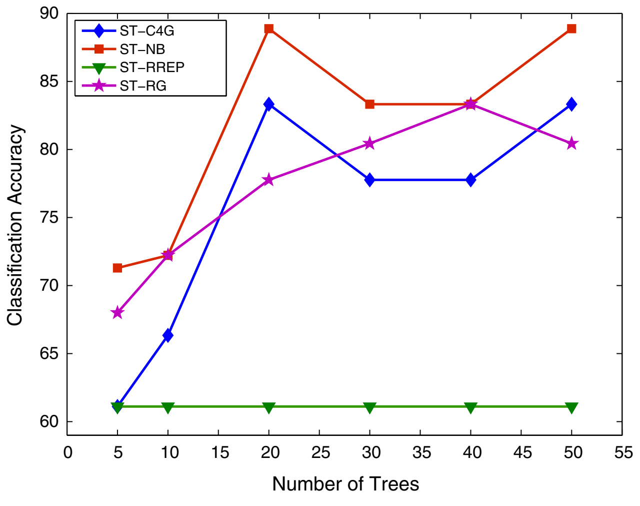
As mentioned in Sect. 3, at each iteration of the self-training algorithm a set of informative newly-labeled examples is selected for the next iterations. Therefore there is a need for a selection metric. We have used the probability estimation of the decision tree as the selection criterion. However in order to be able to select this subset, we need to introduce a threshold (T in Algorithm 1) regarding the probability estimation of the base learner. In fact, this threshold is a tuning parameter and needs to be tuned for each dataset.

Table 8 Average performance of supervised learning and self-training using the Mahalanobis distance along with the probability estimation as the selection metric

Dataset	Supervised learning				Self-training			
	RFG	RREP	RG	RNB	ST-RFG	ST-RREP	ST-RG	ST-RNB
Breath-cancer	66.25	68.50	68.50	74.50	69.80	70.50	71.95	76.93
Bupa	57.34	56.45	55.04	58.40	61.60	60.76	61.06	60.04
Car	87.00	77.60	80.60	78.00	89.27	81.40	84.29	81.18
Cmc	60.25	58.75	59.50	57.25	64.00	60.00	64.12	60.10
Colic	75.00	67.32	77.50	76.77	75.21	71.15	80.60	79.32
Diabetes	69.56	70.66	67.82	70.04	71.80	70.98	71.80	72.00
Heart	74.99	72.04	73.18	70.91	77.01	74.15	76.62	74.27
Hepatitis	80.00	80.00	80.40	79.60	81.81	81.25	82.15	83.00
Ionoshere	80.00	71.20	73.04	78.31	82.34	74.79	78.56	81.91
Liver	56.60	56.00	61.40	56.80	59.00	57.92	63.60	58.52
Sonar	63.60	59.20	63.40	59.80	69.07	64.15	69.23	63.14
Tic-tac-toe	70.00	67.20	69.60	68.20	72.01	69.34	70.67	71.33
Vote	91.25	88.78	89.00	88.78	93.25	91.53	93.17	93.79
Wave	86.00	85.75	87.25	88.75	88.15	86.97	89.50	89.86
Aesthetics	58.55	60.88	63.88	60.41	61.91	61.04	68.91	65.01
Recency	68.49	70.37	70.37	71.29	71.87	72.3	78.74	75.57



(a) Aesthetics dataset



(b) Recency dataset

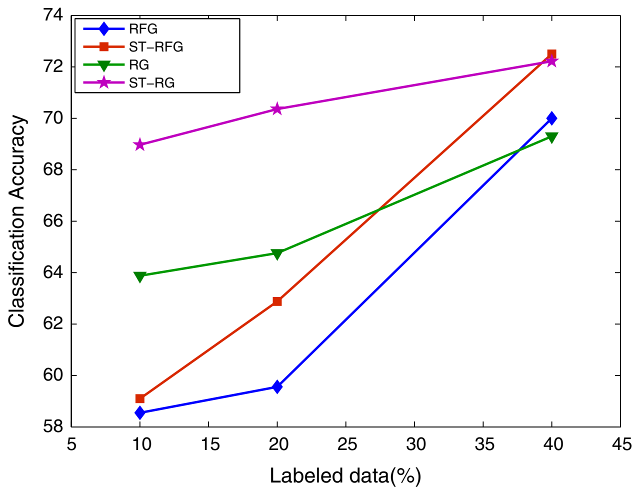
Fig. 3 Average performance of self-training with increasing the number of trees on web-pages datasets

We perform several experiments on *colic* dataset to show the effect of this threshold on the performance of self-training. Figure 5 shows the results of the experiments.

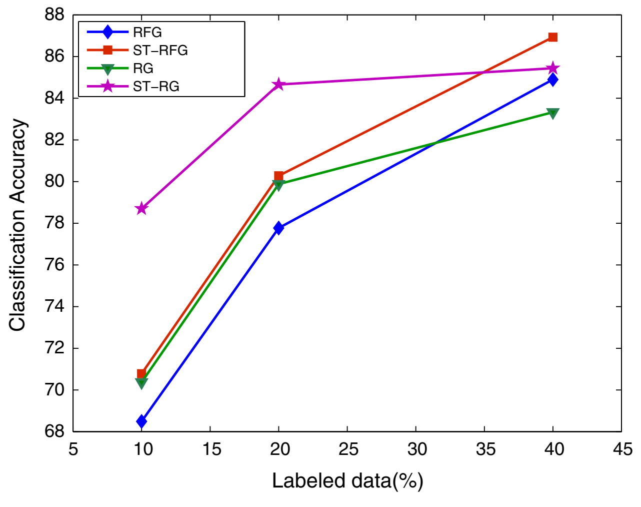
As can be seen, the performance of the self-training with NBTree as the base classifier changes when the different threshold for selection metric is used in the training procedure. The same results are seen for self-training when the base learner is RSM with NBTree. As a result the selection of the threshold has a direct impact on the performance. In the experiment we selected 10 % of the high-confidence predictions and used the mean of the probability estimation of these predictions as threshold.

8 Multiclass classification

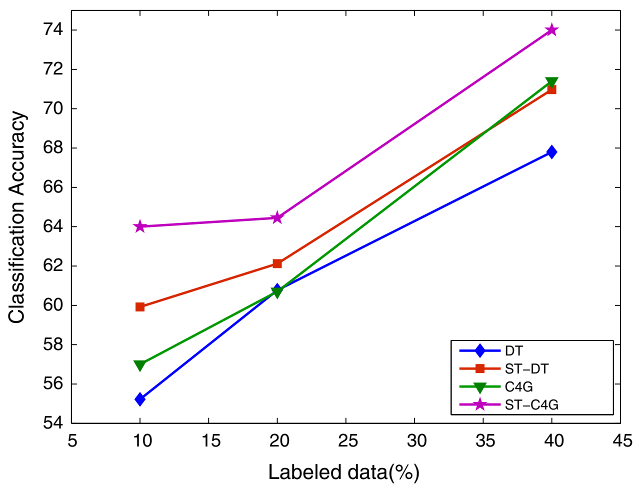
In order to generalize the proposed methods to multiclass classification problem, we perform several experiments to show the effect of the improved probability estimation of the decision trees. Since the decision tree classifiers are basically



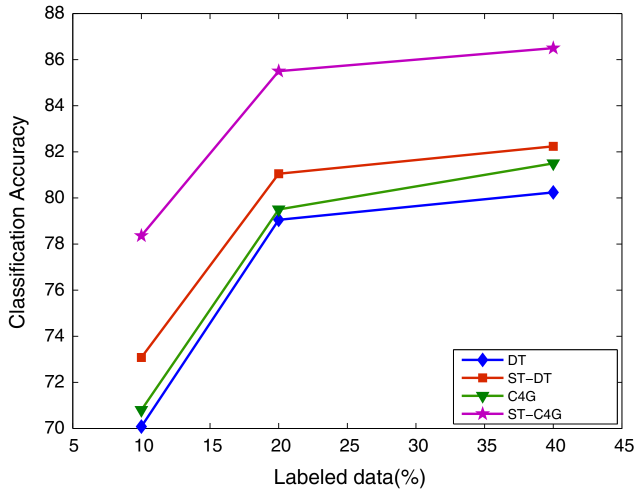
(a) Aesthetics dataset



(b) Recency dataset

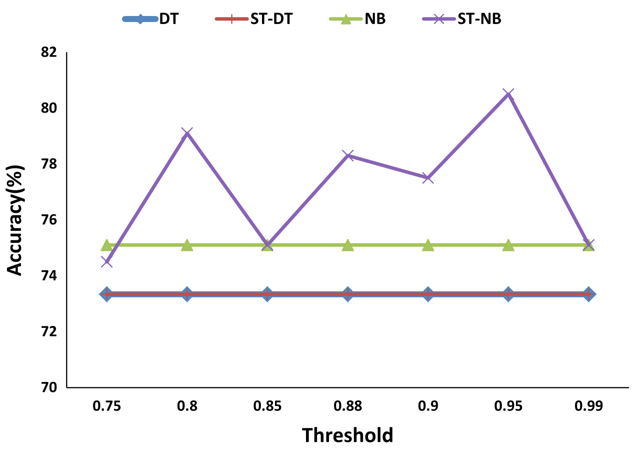


(c) Aesthetics dataset

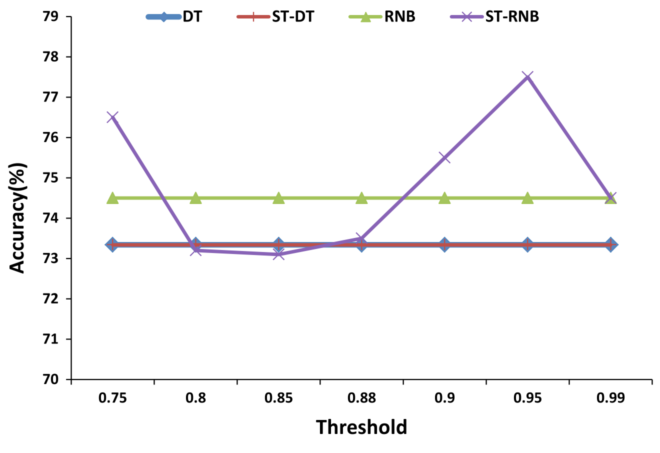


(d) Recency dataset

Fig. 4 Average performance of self-training [using ensemble classifier (**a**, **b**) and single classifier (**c**, **b**) as the base learner] with increasing proportions of labeled data on web-pages datasets



(a) Single classifier



(b) Ensemble classifier

Fig. 5 The classification accuracy of self-training (using single classifier (**a**) and ensemble classifier (**b**) as the base learner) with different value for T parameter on *colic* dataset

Table 9 Overview of multiclass datasets

Dataset	# Samples	# Attributes	# Classes
Balance	625	4	3
Car	1,728	6	4
Cmc	1,473	9	3
Iris	150	4	3
Vehicle	846	19	4

multiclass classifiers [46]. Therefore they can directly handle the multiclass classification problems, which is one of the key advantages of using the decision trees. Most of the classifiers often convert the original multiclass problem into several binary classification problems using one-vs-one or one-vs-all methods, which may lead to several problems like imbalance data [38, 40].

In this section, we use five multiclass UCI datasets in the experiments as shown in Table 9. Table 10 compares the results of the standard decision tree learner (DT) to its self-training version (ST-DT) and the same for C4.4graft (C4G), and the Naive Bayes Decision trees (NB). It also compares the results of the DT to ensemble classifiers RF with C4.4graft (RFG) and RSM with C4.4graft (RG) and NBTree (RNB) as the base learners.

As can be seen in Table 10, consistent with the performance of the binary classification, the proposed methods outperform the performance of the standard decision tree. Figure 6 shows the average improvement of the used methods in the comparison.

We further present the precision (P), recall (R), and the area under curve (AUC) of some datasets in terms of each class to show the improvements in more details. In this experiment *Balance* and *Cmc* datasets are evaluated using self-training with single and ensemble classifiers as in the pervious experiment. Table 11 shows the results in more details.

9 Conclusions and discussions

The main contribution of this paper is the observation that when a learning algorithm is used as the base learner in self-training, it is very important that the confidence of prediction is correctly estimated, probability estimation. The standard technique of using the distribution at the leaves of decision tree as probability estimation does not enable self-training with a decision tree learner to benefit from unlabeled data. The accuracy is the same as when the decision tree learner is applied to only the labeled data. If a modified decision tree learner is used which has an improved technique for estimating probability, then self-training with the modified version does benefit from the unlabeled data. Although to a lesser extent, the same is true when the modified decision tree learners are used as the base learner in an ensemble learner.

Based on the results of the experiments we conclude that improving the probability estimation of the tree classifiers leads to better selection metric for the self-training algorithm and produces better classification model. We observe that using Laplacian correction, No-pruning, grafting, and NBTree produce better probability estimation in tree classifiers. We also observed that Mahalanobis distance method for sampling is effective and guides a decision tree learner to select a set of high-confidence predictions. The best result based on our experiments with a small amount of labeled instances (10 %), which is the most relevant for semi-supervised settings, is obtained by a combination of grafting, No-pruning, and Laplacian correction. This was useful in the Random Subspace Method as well. Random Forest suffers from the small amount of labeled data and therefore does not work well. Better probability-based ranking and high classification accuracy could select the high-confidence predictions in the selection step of self-training and therefore, these variations improved the performance of self-training.

Table 10 Average classification accuracy of supervised learning and self-training using single and ensemble classifiers

Self-training with ensemble classifiers								
Datasets	DT	ST-DT	RFG	ST-RFG	RG	ST-RG	RNB	ST-RNB
Balance	63.59	63.11	67.96	69.41	68.44	70.38	66.99	66.99
Car	76.94	77.29	79.23	81.34	72.71	73.50	72.00	75.18
Cmc	42.77	44.42	45.46	47.32	46.28	50.00	44.70	48.02
Iris	77.08	77.08	77.08	81.25	89.58	95.83	91.67	95.83
Vehicle	53.99	54.71	59.42	61.96	59.42	64.13	64.85	65.22
Self-training with single classifiers								
Datasets	DT	ST-DT	NB	ST-NB	C4G	ST-C4G		
Balance	63.59	63.11	67.47	71.84	65.53	67.96		
Car	76.94	77.29	76.76	78.52	74.64	76.23		
Cmc	42.77	44.42	42.36	45.04	42.96	45.46		
Iris	77.08	77.08	89.58	91.67	75.00	75.00		
Vehicle	53.99	54.71	61.59	62.68	53.99	55.70		

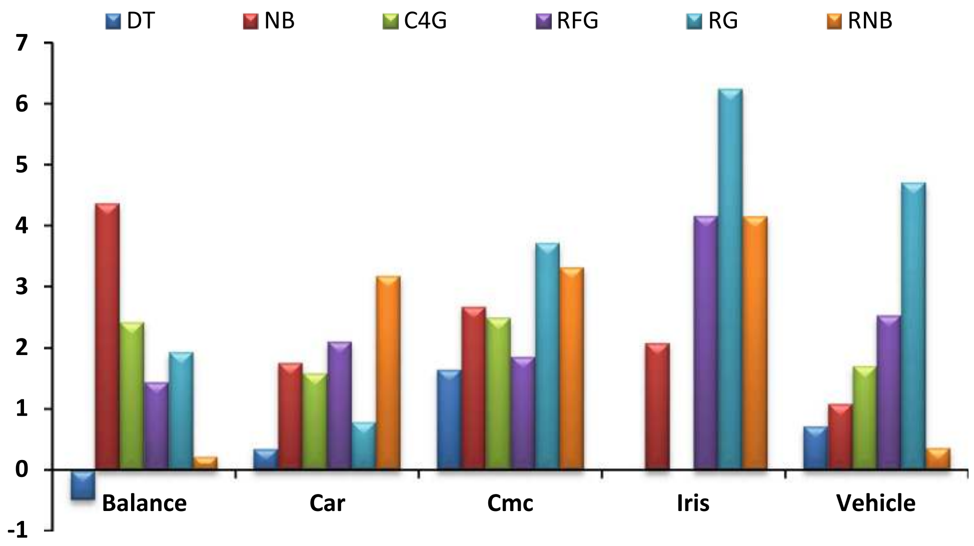


Fig. 6 The improvement in the classification performance of the used methods in Table 10

Table 11 Detailed classification accuracy by classes for *Balance* and *Cmc* datasets using self-training with single and ensemble classifiers

Classes	DT			ST-DT			NB			ST-NB			C4G			ST-C4G		
	P	R	AUC	P	R	AUC	P	R	AUC	P	R	AUC	P	R	AUC	P	R	AUC
Balance																		
Class 1	0.74	0.63	0.80	0.68	0.71	0.77	0.63	0.87	0.74	0.68	0.85	0.85	0.71	0.76	0.76	0.75	0.74	0.79
Class 2	0.07	0.13	0.42	0.07	0.13	0.44	0.0	0.0	0.56	0.0	0.0	0.47	0.0	0.0	0.44	0.04	0.06	0.60
Class 3	0.73	0.73	0.76	0.78	0.64	0.74	0.79	0.62	0.77	0.77	0.71	0.83	0.70	0.72	0.76	0.69	0.66	0.79
	RFG			ST-RFG			RFG			ST-RFG			RNB			ST-RNB		
Class 1	0.71	0.77	0.77	0.72	0.75	0.83	0.68	0.78	0.84	0.68	0.83	0.85	0.76	0.59	0.73	0.78	0.60	0.86
Class 2	0.0	0.0	0.46	0.09	0.13	0.60	0.0	0.0	0.53	0.0	0.0	0.55	0.0	0.0	0.47	0.0	0.0	0.44
Class 3	0.80	0.74	0.81	0.79	0.71	0.85	0.72	0.71	0.86	0.78	0.70	0.87	0.62	0.86	0.72	0.65	0.86	0.85
Classes	DT			ST-DT			NB			ST-NB			C4G			ST-C4G		
Cmc																		
Class 1	0.47	0.63	0.62	0.49	0.75	0.61	0.46	0.45	0.54	0.49	0.6	0.57	0.48	0.56	0.58	0.54	0.47	0.57
Class 2	0.32	0.27	0.64	0.37	0.60	0.68	0.35	0.40	0.64	0.47	0.32	0.67	0.29	0.27	0.58	0.37	0.32	0.64
Class 3	0.38	0.27	0.56	1.0	0.02	0.60	0.42	0.39	0.54	0.39	0.41	0.56	0.43	0.36	0.61	0.42	0.52	0.61
	RFG			ST-RFG			RFG			ST-RFG			RNB			ST-RNB		
Class 1	0.50	0.48	0.60	0.50	0.70	0.61	0.48	0.63	0.62	0.48	0.86	0.65	0.49	0.55	0.59	0.56	0.58	0.61
Class 2	0.36	0.31	0.67	0.37	0.18	0.56	0.37	0.23	0.66	0.58	0.18	0.69	0.42	0.28	0.59	0.43	0.31	0.65
Class 3	0.45	0.51	0.60	0.47	0.39	0.59	0.45	0.40	0.60	0.57	0.29	0.62	0.40	0.42	0.57	0.44	0.5	0.58

Future work can consider extending the proposed selection metric to multiclass classification. Most of the measures we used are for binary classification and can be extended to the muticlass case. Another line is to consider the probability estimates of other learning algorithms that are not designed to output probabilities. Also co-training algorithms rely on good probability estimates. We expect that similar issues play a role in that setting.

References

1. Asuncion A, David N (2007) UCI machine learning repository
2. Belkin M, Niyogi P, Sindhwani V (2006) Manifold regularization: a geometric framework for learning from labeled and unlabeled examples. J Mach Learn Res 7:2399–2434
3. Bellet A, Habrard A, Sebban M (2013) A survey on metric learning for feature vectors and structured data. arXiv:13066709
4. Bennett K, Demiriz A (1999) Semi-supervised support vector machines. In: Proceedings of the 1998 conference on advances in neural information processing systems (NIPS), vol 11. MIT Press, Cambribge, pp 368–374
5. Blockeel H, Raedt LD (1998) Top-down induction of first-order logical decision trees. Artif Intell 101:285–297
6. Blum A, Mitchell T (1998) Combining labeled and unlabeled data with co-training. In: Computational learning theory, pp 92–100
7. Boer V, Someren M, Lupascu T (2011) Web page classification using image analysis features. Web Inf Syst Technol 75:272–285
8. Breiman L (1996) Bagging predictors. Mach. Learn. 24:123–140
9. Breiman L (2001) Random forests. Mach. Learn. 45:5–32
10. Caruana R, Niculescu-Mizil A (2006) An empirical comparison of supervised learning algorithms. In: ICML, pp 161–168
11. Dempster AP, Laird NM, Rubin DB (1977) Maximum likelihood from incomplete data via the em algorithm. J R Stat Soc 39(1):1–38
12. Demšar J (2006) Statistical comparisons of classifiers over multiple data sets. J Mach Learn Res 7:1–30
13. Dietterich T (2000) An experimental comparison of three methods for constructing ensembles of decision trees: bagging, boosting, and randomization. Mach Learn 40:139–157
14. Domingos P, Pazzani M (1997) On the optimality of the simple bayesian classifier under zero-one loss. Mach Learn 29(2–3):103–130
15. Freund Y, Schapire R, Abe N (1999) A short introduction to boosting. J Jpn Soc Artif Intell 14:1612–1635
16. Friedl M, Brodley C, Strahler A (1999) Maximizing land cover classification accuracies produced by decision trees at continental to global scales. IEEE Trans Geosci Remote Sens 37:969–977
17. Haffari G, Sarkar A (2007) Analysis of semi-supervised learning with the yarowsky algorithm. In: Uncertainty in Artificial Intelligence (UAI), pp 159–166
18. Hall M, Frank E, Holmes G, Pfahringer B, Reutemann P, Witten IH (2009) The weka data mining software: an update. SIGKDD Explor Newsl 11:10–18
19. Ho T (1998) The random subspace method for constructing decision forests. IEEE Trans Pattern Anal Mach Intell 20(8):832–844
20. Holm S (1979) A simple sequentially rejective multiple test procedure. Scand J Stat 65–70
21. Hoyle DC (2011) Accuracy of pseudo-inverse covariance learninga random matrix theory analysis. Pattern Anal Mach Intell IEEE Trans 33(7):1470–1481
22. Iman R, Davenport J (1980) Approximations of the critical region of the fbietkan statistic. Comm Stat Theory Methods 9:571–595
23. Joachims T (1999) Transductive inference for text classification using support vector machines. In: ICML, pp 200–209
24. Kohavi R (1996) Scaling up the accuracy of naive-bayes classifiers: a decision-tree hybrid. In: ACM SIGKDD, pp 202–207
25. Li Y, Guan C, Li H, Chin Z (2008) A self-training semi-supervised svm algorithm and its application in an eeg-based brain computer interface speller system. Pattern Recognit Lett 29:1285–1294
26. Maeireizo B, Litman D, Hwa R (2004) Co-training for predicting emotions with spoken dialogue data. In: ACL on Interactive poster and demonstration sessions, pp 28–36
27. Mallapragada P, Jin R, Jain A, Liu Y (2009) Semiboost: boosting for semi-supervised learning. IEEE Trans Pattern Anal Mach Intell 31(11):2000–2014
28. Mease D, Wyner AJ, Buja A (2007) Boosted classification trees and class probability/quantile estimation. J Mach Learn Res 8:409–439
29. Moret SL, Langford WT, Margineantu DD (2006) Learning to predict channel stability using biogeomorphic features. Ecol Model 191(1):47–57
30. Nigam K, McCallum A, Thrun S, Mitchell T (2000) Text classification from labeled and unlabeled documents using em. Mach Learn 39:103–134
31. Provost FJ, Domingos P (2003) Tree induction for probability-based ranking. Mach Learn 52:199–215
32. Quinlan JR (1993) C4.5: Programs for Machine Learning, vol 1. Morgan kaufmann
33. Riloff E, Wiebe J, Phillips W (2005) Exploiting subjectivity classification to improve information extraction. In: Proceedings of the National Conference On Artificial Intelligence, pp 1106–1114
34. Rosenberg C, Hebert M, Schneiderman H (2005) Semi-supervised self-training of object detection models. In: WACV/MOTION, pp 29–36
35. Shahshahani B, Landgrebe D (1994) The effect of unlabeled samples in reducing the small sample size problem and mitigating the hughes phenomenon. IEEE Trans Geosci Remote Sens 32(5):1087–1095
36. Simonoff J (1996) Smoothing methods in statistics. Springer, New York
37. Tanha J, van Someren M, Afsarmanesh H (2011) Disagreement-based co-training. In: Tools with Artificial Intelligence (ICTAI), pp 803–810
38. Tanha J, van Someren M, Afsarmanesh H (2012) An adaboost algorithm for multiclass semi-supervised learning. In: International Conference on Data Mining (ICDM), pp 1116–1121
39. Tanha J, Saberian M, van Someren M (2013) Multiclass semi-supervised boosting using similarity learning. In: Data Mining (ICDM), 2013 IEEE 13th International Conference on, pp 1205–1210
40. Tanha J, Van Someren M, Afsarmanesh H (2014) Boosting for multiclass semi-supervised learning. Pattern Recognit Lett 37:63–77
41. Wang B, Spencer B, Ling CX, Zhang H (2008) Semi-supervised self-training for sentence subjectivity classification. In: Proceedings of 21st conference on Advances in artificial intelligence, pp 344–355
42. Wang XZ, Zhang SF, Zhai JH (2007) A nonlinear integral defined on partition and its application to decision trees. Soft Comput 11(4):317–321
43. Wang XZ, Dong LC, Yan JH (2012) Maximum ambiguity-based sample selection in fuzzy decision tree induction. Knowl Data Eng IEEE Trans 24(8):1491–1505
44. Webb GI (1999) Decision tree grafting from the all tests but one partition. In: IJCAI, pp 702–707
45. Yarowsky D (1995) Unsupervised word sense disambiguation rivaling supervised methods. In: ACL, pp 189–196
46. Yi W, Lu M, Liu Z (2011) Multi-valued attribute and multi-labeled data decision tree algorithm. Int J Mach Learn Cybern 2(2):67–74
47. Zhou Z, Li M (2010) Semi-supervised learning by disagreement. Knowld Inf Syst 24:415–439
48. Zhu X (2005) Semi-Supervised Learning Literature Survey. Tech. Rep. 1530, Computer Sciences, University of Wisconsin-Madison