PLAC: Partitioning Based Lazy Classification

Wei Song[1*], He Jiang[2], Fan Ma[3], Qinbao Song[3], Guangtao Wang[3]

[1] Department of Electrical Engineering and Computer Sciences, University of California at Berkeley, USA.
[2] School of Software Technology, Dalian University of Technology, China.
[3] Department of Computer Science and Technology, Xi'an Jiaotong University, China.

* Corresponding author. Tel.: +1 510-693-7897; email: weisong@berkeley.edu
Manuscript submitted November 5, 2018; accepted December 26, 2018.
doi: 10.17706/jsw.14.2.65-91

Abstract: Traditional classification methods cannot well capture the characteristics of complex problems, thus leading to poor performance. In this paper, we propose a new framework named Partition based LAzy Classification (PLAC) to better characterize complex problems by dividing the training data space into smaller and easier-to-learn partitions. In PLAC, only the nearest partition of a new instance is used to train a local classifier that is finally used to classify the new instance. As the partitioning is performed based on information gain before receiving a new instance, the resulting partitions are groups of similar instances and the chance of the nearest instances of the new instance coming from different regions by accident is reduced. Moreover, our method uses only one partition to conduct a prediction and employs the caching mechanism to avoid work replication during classification, thus efficiency is improved. An extensive experimental evaluation on 40 real world data sets shows that PLAC effectively improves the performance of base classifiers and outperforms existing mainstream ensemble methods.

Key words: Classification, eager learning, lazy learning, data partitioning, ensemble learning.

1. Introduction

As an important type of machine learning task, classification has been fully investigated. Most classification methods follow a two-stage process: induction and deduction. In the inductive stage, the classifier(s) is built over the training data; during the deduction stage, the classifier(s) is applied to the new data and the class label is assigned. In literature, many different types of methods have been proposed, including tree-based C4.5 [1], CART [2], and LAZYDT [3]; rule-based RIPPER [4], PART [5], and OneR [6]; instance-based IB1 [7], [8]; probability-based naive Bayes NB [9] and support vector machines SVM [10].

Depending on the time of the classifier(s) was constructed, present classification methods can be roughly divided into two groups: eager learning and lazy learning. An eager learning method uses the training data to build a uniform classifier at the inductive stage and finally the classifier is employed to classify all new instances. However, a uniform classifier cannot well characterize complex problems (see Section 3) and thus may lead to poor performance. In contrast, a lazy learning method does not build a specific classifier with a subset of the training data for each new instance until the deductive stage. Hence, a lazy learning method may better capture the specific characteristics of a new instance and provides an approach to learning classifiers of complex phenomena and dealing with large amounts of data [3], [11], [12]. However, overfitting may occur in lazy learning due to the lack of "global" view of the training data.

Aiming to bridge the gap between eager learning and lazy learning, we propose a new classification framework named Partition based LAzy Classification PLAC. PLAC follows a divide-and-conquer approach by "globally" dividing the training data space into smaller and easier-to-learn partitions, where only the nearest partition of a new instance is used to train the local classifier that is finally used to classify the new instance. As a generic framework, existing classification methods can be incorporated into PLAC to build base classifiers. More specifically, we employ information gain to discriminate the features of the training data and build a hierarchy tree of data partitions. Given a new instance for classification, PLAC sorts it down the hierarchy tree and locates its nearest partition. With this partition, a classifier is built to predict the class label for this new instance. Since some new instances' nearest partition may be the same, we cache new classifiers after they have been built, so as to avoid rebuilding a classifier multiple times. Hence, the classification efficiency of PLAC can be improved.

To evaluate the effectiveness of PLAC, we conduct extensive experiments over 40 real world data sets. In the experiments, we employ five eager learning methods as the base learner of PLAC, including NB, C4.5, SVM, IB1, and OneR. For comparison, we also run a lazy learning algorithm named LAZYDT and two ensemble methods, e.g., Bagging and Boosting. According to the experiments, PLAC greatly improves the accuracy of the five eager learning methods by up to 14.70%. As for most of the base learners, PLAC outperforms LAZYDT and two ensemble methods Bagging and Boosting in terms of accuracy and running time. In addition, as to the performance validation by the nonparametric Friedman test and Nemenyi post-hoc test, PLAC performs best among all the methods in this paper.

In summary, this paper makes the following main contributions: 1) We propose a new method named PLAC to build classifiers for complex problems. As a generic framework, PLAC can employ existing classification algorithms as base learners; 2) We conduct extensive experiments on 40 real world data sets and validate the effectiveness and efficiency of PLAC; 3) All the source codes are publicly available for academic usages.

The rest of this paper is organized as follows: Section 2 provides an overview of the related work; Section 3 presents basic concepts and foundation of our method; Section 4 details our proposed classification method; Section 5 reports and analyzes the experimental results; and Section 6 draws the conclusion.

2. Related Work

In this section, we present the studies related to our work, including lazy learning and ensemble learning. Our work is related to lazy learning. k-Nearest Neighbor (k-NN) [8] classification is a typical lazy learning method, where the training data are simply stored and the inductive stage is deferred until an unseen instance is given. The lazy decision tree algorithm LAZYDT [3] builds a "best" decision tree for each new instance at classification time. Obviously, LAZYDT is very different from C4.5 [1] and CART [2], which create a single decision tree for all new instances during the inductive stage. The lazy associative classification method [13] focuses on the features that actually occur within the test instance while generating the rules, thus overcoming the large rule-set problem of traditional associative classifiers. The lazy bagging classification method [14] builds bootstrap replicate bags based on the characteristics of the test instances. Upon receiving a test instance, this method trims bootstrap bags by taking into consideration the new instance's nearest neighbors in the training data. The aforementioned methods postpone the classifier construction to the classification time and only store training instances at the inductive stage. Although our method also defers the construction of the classifier, we divide the training data into partitions during the inductive stage, which quickens up classifier construction and avoids the chance of the nearest instances of a new instance coming from different regions by accident.

Our work is also related to ensemble learning, which consists of building base classifiers and combining their predictions. Accurate and diverse base classifiers are critical for gaining a high performing ensemble [15], [16]. Diversity can be obtained by partitioning the training data at instance-level and feature-level, or with mixture strategy (i.e., at both instance and feature levels). Both Bagging [17] and Boosting [18] create diversity at instance-level. The difference is that the former achieves this purpose by generating multiple bootstrap samples, the latter by making the succeeding samples biased towards the misclassified instances of the built classifier(s) via adaptive resampling. Random subspace method [19] and attribute bagging [20]-[21] are examples of the feature-level partitioning, whose effectiveness has been demonstrated by Cherkauer [22] in 1996. Breiman [23] combined Bagging with a random subspace method to improve diversity, thus is a mixture strategy. The resulting partitions of these strategies are similar and generally highly correlated [24]. Our method employs a different mixture strategy, one that guarantees the instances are similar to one another within the same partition and are dissimilar to the instances in other partitions. Moreover, our method uses only the nearest partition to build the classifier, so efficiency can be improved.

3. Preliminaries and Motivation

In this section, we first present some definitions related to classification. Then, we introduce our motivation for the new method PLAC.

3.1. Preliminaries

Suppose D is a data set consisting of n instances $d_1, d_2, \cdots, d_n$, i.e., $D = \{d_1, d_2, \cdots, d_n\}$ $(n \in N^+)$; and Y denotes its target concept with limited discrete values, e.g., for a binary classification problem, $Y = \{+, -\}$, where '+' and '−' are class labels. Each instance $d_i \in D$ is characterized by features $F_1, F_2, \cdots, F_m (m \in N^+)$ and target concept Y, that is, $d_i =< X_i, y_i >$, where X_i and y_i are the value-assignments of $F_1 \times F_2 \times \cdots \times F_m$ and target concept Y, respectively; $X = U_{i=1}^{n} X_i$. With these notations, classification related concepts can be defined as follows.

Definition 1. (Classifier). A classifier is a function $f_D: X \to Y$ built from the given data set D such that for a new instance X_{new} its target concept y_{new} can be obtained via $y_{new} = f_D(X_{new})$ with satisfied classification performance.

Definition 1 indicates that the main task of classification is to build the classifier, which must be evaluated to determine its goodness before using it to make predictions.

Definition 2. (Classifier Error). Let f_D be the classifier learned from data set D, $d_i =< X_i, y_i >$ $(1 \le i \le n)$ be an instance whose target concept is y_i, and $f_D(X_i)$ be the predicted target concept of X_i with f_D, the classification error of f_D upon instance d_i can be defined as

$$\partial(f_D(X_i), y_i) = \begin{cases} 0 & if\ f_D(X_i) = y_i \\ 1 & if\ f_D(X_i) \neq y_i. \end{cases} \quad (1)$$

Summing up the classification error $\partial(f_D(X_i), y_i)$ of each instance $d_i \in D$ gives us the classification error Err of f_D on data set D. That is,

$$Err(f_D) = \sum_{i=1}^{|D|} \partial(f_D(X_i), y_i). \quad (2)$$

Since our proposed method is based on data partitioning, we formally define the related concepts and lay the foundation for further discussion.

Definition 3. (Partition Set). For a given data set D, the non-empty subsets $D_1, D_2, \cdots, D_k (k \in N^+)$

constitute a partition set of data D if and only if

$$D = (\bigcup_{i=1}^{k} D_i) \wedge (D_i \cap D_{j \neq i} = \emptyset, 1 \leq j \leq k), \tag{3}$$

where, $D_i \in D$ *is referred as a partition.*

3.2. Motivation

Although many classification methods perform well over simple problems, their performances on complex matters are usually reduced. Fig. 1 illustrates this situation with a moderate complex binary classification problem.

In Fig. 1, '+' and '−' denote two different types of instances, which lie in the upside and downside of the boundary that can be viewed as the true classifier f_D, respectively; the solid straight line stands for the classifier f'_D learned from the given data with a linear classification method. Obviously, the true classifier f_D is a curve and the learned classifier f'_D is a straight line, the difference between the two is obvious, thus the latter can incur bigger classification error and can be further improved.

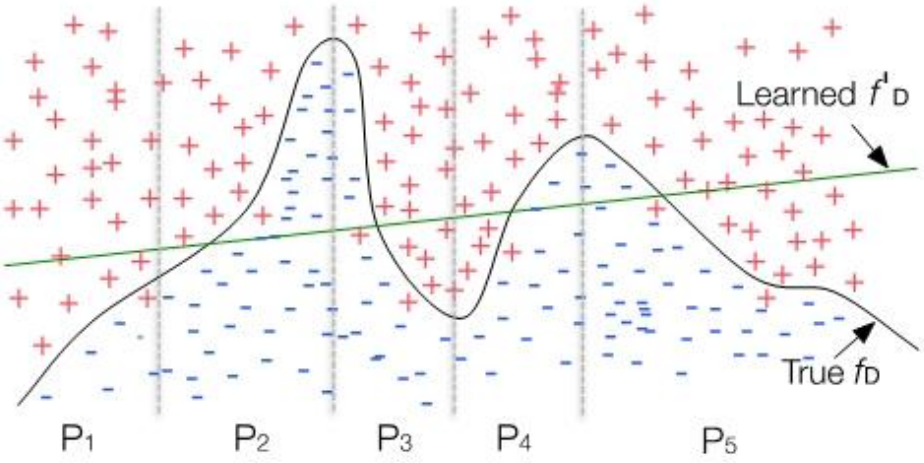


Fig. 1. A complex classification problem.

Generally, in order to get better classification performance, the learned classifier f'_D should approximate the true classifier f_D as much as possible, since the latter effectively characterizes the underlying structure of the given data. Unfortunately, limited to both the internal rationale of classification algorithms and the data sets themselves, it is hard to directly learn the intrinsic characteristics of complex problems.

However, taking a closer look at Fig. 1, a complicated problem may consist of various simple problems. In Fig. 1, the four dashed vertical lines divide the problem space into five regions that constitute a partition set of the original data; each partition of the set is relatively simple and can be easily handled with a classification algorithm.

Hence, it is reasonable to learn multiple classifiers over the partition set of the given data and use the learned classifiers to approximate the true classifier. Since the partition set consists of small pieces of data whose complexity is much smaller than that of the original data, and learning classifiers from small simple data is much easier than that from bigger more complicated data, our proposed classification method can be based on data partitioning. Since every partition in our method should contain enough information such that it is able to distinguish different types of instances within itself. Therefore, a suitable data partitioning method is needed to guarantee the effectiveness of the partition set, and further to lay a solid foundation for

our proposed classification method.

4. PLAC

4.1. Framework

From the above analysis, we now have a rough idea of the framework of our method: to split the data into multiple partitions and only use one of them to train a classifier, with the classifier training delayed until a classification task is issued. Everything seems to be in place, but how do we implement this in a more specific way? Let's now look into the k-NN classification algorithm [7] for some inspirations.

As we know, k-NN is one type of lazy learning, where the classifier is approximated locally only with the instances in the nearest neighborhood of a new instance. This means that if we only view one of the multiple partitions as the nearest neighbor of an unseen instance, the underlying mechanism of k-NN can help make our method work. More specifically, we first divide the training data into a set of partitions that can well characterize the training data, for obvious reasons; then upon receiving a new instance to classify, we build the classifier on its nearest partition, and eventually use this classifier for classifying the new instance. This forms the framework of our proposed classification method, Fig. 2 shows the details.

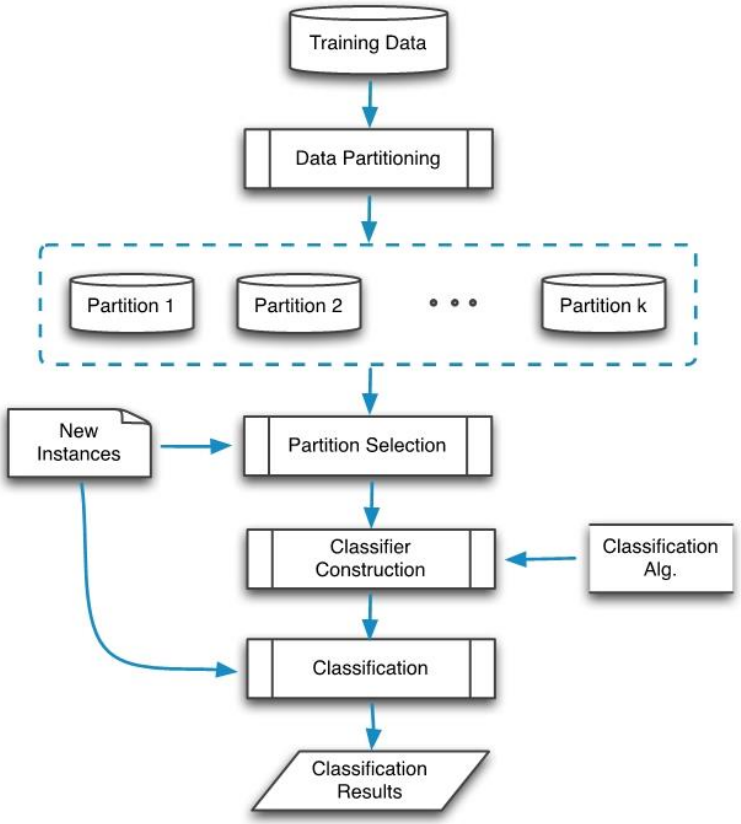


Fig. 2. Framework of the proposed classification method.

From Fig. 2 we know that our proposed classification method is a generic framework, which is not limited to any specific classification algorithm. In the following subsections, we will focus on introducing the data partitioning technique and the classification strategy, respectively.

4.2. Data Partitioning

Data partitioning is the process of dividing the given data into multiple small partitions that constitute a partition set. The true complex classifier can be approximated with the simpler classifiers learned from the

partition set. Obviously, it is easier to train classifiers from small partitions than from the entire data. Therefore, the effectiveness of these partitions is a must. For this purpose, we need to address the following three problems: (1) What should be used to partition the data? (2) How to perform the partitioning? and (3) When to stop the partitioning process?

As we know, each instance of a data set is characterized by a set of features, generally the feature values of different instances are dissimilar. This means features can be used to differentiate distinct instances, and the richer the information a feature contains, the stronger its discriminability. This kind of ability can be computed with *information gain*, which is based on *entropy*.

Definition 4. (Entropy). Let D be a data set and $Y = \{y_1, y_2, \cdots, y_r\} (r \in N^+)$ be its target concept, the entropy of D relative to Y is defined as

$$Entropy(D) = \sum_{i=1}^{r} -\frac{|D_{y_i}|}{|D|} log_2 \frac{|D_{y_i}|}{|D|} \tag{4}$$

where D_{y_i} is the subset of D for which Y takes value y_i, and $\frac{|D_{y_i}|}{|D|}$ is the proportion of the instances whose target concept is y_i in D.

Entropy characterizes the impurity of a data set, a greater value implies higher impurity. Based on entropy, the information gain, which indicates how well a feature separates the instances according to their target concepts, can be defined.

Definition 5. (Information Gain). The information gain Gain (D, F) of feature F relative to data D is define as

$$Gain(D, F) = Entropy(D) - \sum_{v \in Vals(F)} \frac{|D_v|}{|D|} Entropy(D_v) \tag{5}$$

where $Vals(F)$ is the set of all possible values[2] for feature F, and D_v is the subset of D for which F has value v.

From Definition 5 we observe that the first term of the expression $Gain(D, F)$ is the entropy of data D, and the second term is the expected value of the entropy after D is partitioned with F. Therefore, $Gain(D, F)$ represents the expected reduction in entropy caused by partitioning D using F; greater values indicate purer/simpler partitions, which are what we desire.

From the above introduction we know that $Gain(D, F)$ can help us with choosing the appropriate features for getting simple partitions of the given data. Once a feature is selected, the current data is partitioned into smaller pieces. Thus, the feature selection process is also a data partitioning process, in which the current best feature F_{split} is chosen according to

$$F_{split} = \underset{F \in F_{curr}}{argmax}\, Gain(D_{curr}, F) \tag{6}$$

where D_{curr} is the current data set and is characterized with feature set F_{curr}.

Feature selection is just one of the many things we need to do, the complete data partitioning process is as follows:

1. for current data D_{curr}, the information gain $Gain(D_{curr}, F)$ of each feature $F \in F_{curr}$ is computed, and the maximum one is chosen as the current best feature F_{split}.
2. use F_{split} to divide current data D_{curr} into multiple partitions D_1, D_2, ..., D_k according to its values, each partition D_i(i=1,2,...,k) results from a distinct feature value. This means within each partition, F_{curr} has an identical value.
3. view each partition D_i(i=1,2,...,k) as current data D_{curr}, repeat steps 1 and 2 until the termination

condition is met. The final partitions consist of the partition set.

This process can be viewed as the process of building a hierarchical tree, illustrated in Fig. 3. Firstly, $F_{root} \in F_{curr}$ is chosen as the split feature from current data D_{curr}, and the root of the tree is created. As feature F_{root} has m values $v_1,v_2,\cdots,v_m$, m branches are grown, and each one corresponds to a temporary partition of D_1, D_2, ..., D_m, respectively. Because partition D_m meets the termination condition, leaf L_m is formed. For each of the remaining partitions D_i(i=1,2,...,m-1), the split feature F_i is chosen as the root of a subtree. Repeat the above process for each root until all subtrees meet the termination condition, finally the set of leaf nodes $\{L_1, L_2, \cdots, L_m\}$ constitutes the partition set we want.

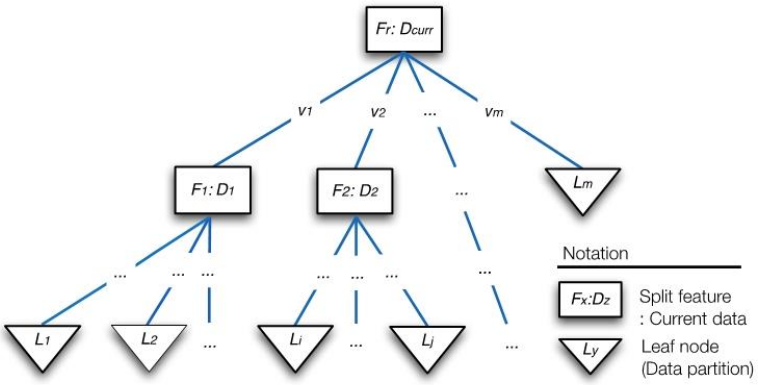


Fig. 3. Data partitioning.

Since only one partition of $\{L_1, L_2, \cdots, L_m\}$ will be used to train a classifier, the number of instances in a partition should be big enough to avoid overfitting. In our method, the size of a partition is controlled with a threshold. Specifically, if the number of instances in a partition is less than the threshold, the partitioning on this partition will be terminated.

4.3. Classification

In our proposed method, the classification of a new instance is very similar to that of the k-NN method. That is, the nearest partition is first located for the new instance, then a single classification algorithm is applied to that nearest partition to train the classifier, which will finally be used to classify the new instance.

The search of the nearest partition is very straightforward. PLAC traverses down the hierarchy tree that was built in the data partitioning process with respect to the feature values of the new instance until a leaf (partition) is located. Due to the fact that the nearest partition of some unseen instances can be identical, it is possible to build the same classifier multiple times, thus lowering the classification efficiency. We avoid this situation by caching new classifiers after they have been built. In this way, next time when the same partition is located by a new instance, the cached classifier is invoked to make a prediction.

4.4. Pseudo Code of PLAC

Our proposed classification method is implemented as a three-step process. In the first step, the training data is partitioned into a set of partitions, with the method presented in Section 4.2. The second step searches for the nearest partition from the partition set for a new instance waiting to classify. Finally, in the third step, the classifier specialized to the new instance is built over the nearest partition and used to classify the new instance. Algorithm 1 shows the details.

Algorithm 1. PLAC

INPUT: *dataset* – the training data set
featureSet – the features characterizing the *dataset*
classificationAlg - the algorithm used to classify new instances
partitionSize - the predefined minimum size of a partition
newInstance - the instance to be classified
OUTPUT*: classLabel* - the predicted class label of *newInstance*

```
//== Step 1: Data partitioning ==
1.  tree ← Partitioning(dataSet, featureSet, partitionSize);
    /* Leaves of the tree consist of the partition set*/
2.  //== Step 2: The nearest partition search ==
3.  currNode ← tree.root; /* Get the current split feature currNode*/
4.
5.  While currNode is NOT a leaf do
6.      v ← Value of the feature newInstance.currNode;
7.      currNode ← Root of the subtree whose parent feature currNode has value v;
8.  nearPart ← The partition corresponding to currNode;
9.  if nearPart has NOT been used to build a classifier then
10.     classifier ← Train classifier on nearPart with classificationAlg;
11.     Cache the classifier classifier;
12. Else
13.     classifier ← Cached classifier for the partition nearPart;
14. classLabel ← Classify newInstance with classifier;
15. return classLabel;
```

Function Partitioning

INPUT: *dataset* –the given data set
featureSet –the features characterizing the *dataset*
partitionSize - the minimum size of a partition
newInstance - the instance to be classified
OUTPUT*: treeRoot* - the tree whose leaves consist of the partition set

```
1.  if | dataset | ≤ partitionSize then
2.      return treeRoot
3.  else
4.      best ← argmax_{F∈featureSet} Gain(dataSet, F);
5.      treeRoot ← Create a new tree with root best;
6.      for each value v of the feature best do
7.          d ← { Instances of dataSet with best=v } ;
8.          subTree ← Partitioning(d, featureSet-best,partitionSize);
9.          Add subTree as a subtree to treeRoot;
10.     return treeRoot
```

Line 1 of Algorithm 1 partitions the training data *dataSet*, which is characterized by *featureSet*, into a partition set using Function Partitioning, and the size of each partition in the set is no less than the predefined *partitionSize*. Function Partitioning uses information gain based method to split the training data *dataset* into small pieces, where the smaller ones have higher purity. Here, partitioning is actually the process of constructing a hierarchy tree, where the internal nodes are split features and the leaf nodes are made up of the partition set.

Line 2 to 6 search for the nearest partition *nearPart* for the unseen instance *newInstance* from the hierarchy tree returned by Function Partitioning. Starting at the root node, the algorithm traverses the tree according to the feature values of the instance *newInstance*, the leaf node encountered is the nearest partition *nearPart*.

Line 7 to 12 classify the new instance *newInstance*. Specifically, if the nearest partition *nearPart* of the new instance *newInstance* has been used to build a classifier, the classifier must have been cached, the cached classifier *classifier* is then invoked to classify the new instance. Otherwise, the specified algorithm *classificationAlg* is applied to the nearest partition *nearPart* to train the classifier, which will be employed to

classify the unseen instance *newInstance* and finally be cached for use of next time.

Time complexity. The computational cost of data partitioning is $O(\tilde{m} \times n \times log(n))$ for data with n instances and m features, where $\tilde{m} \ll m$ is the number of features used to partition the data. $O(h)$ and $O(classificationAlg(nearPart))$ are the computational cost of locating the nearest partition for a new instance from the hierarchy tree whose maximum depth is h and classifying the new instance with classification algorithm $classificationAlg$, respectively. Since $\tilde{m}$ and h do not exceed 25 and 10 according to our experiments on 40 real world data sets, respectively, the time complexity of this algorithm is $O(n \times log(n)) + O(classificationAlg(nearPart))$. Usually $|nearPart| \ll n$, therefore, this algorithm can be very efficient in classifying large data sets. Moreover, if the complexity of algorithm $classificationAlg$ is less than $O(n \times log(n))$, our proposed method is even faster than $classificationAlg$ as a single method, this can be demonstrated by the experiments in Section 5.

5. Experimental Results and Analysis

A large variety of experiments was performed to evaluate the performance of our proposed PLAC and other representative classification methods. In this section, we present the data and the experimental procedure, and report and analyze the experimental results.

5.1. Data Description

To evaluate the performance of PLAC, verifying whether or not it is potentially useful in practice, and allowing other researchers to confirm our results, 40 publicly available data sets were employed to compare PLAC and its competitors. The statistics of these data sets are summarized in Table 1.

Of the 40 data sets, 36 come from UCI Machine Learning Repository [25] that cover a range of application domains such as life sciences, physical sciences, social sciences, engineering, game, and business. The remaining 4 are KDD Cup data sets, which involve areas of fund raising and computer network. The number of instances, features, and target concepts of these data sets vary from 303 to 67557, from 5 to 61, and from 2 to 26, respectively.

Table 1. Summary of the 40 Publicly Available Data Sets

ID	Data Set	I	F	T	ID	Data Set	I	F	T
1	adult-census	32561	15	2	21	kdd-ipums-99	8844	61	9
2	anneal	898	39	6	22	kdd-Japanese	4274	15	9
3	anneal-ORIG	898	39	6	23	kr-vs-kp	3196	37	2
4	balance-scale	625	5	3	24	letter	20000	17	26
5	breast-w	699	10	2	25	liver-disorders	345	7	2
6	car	1728	7	4	26	monks-1	432	7	2
7	cmc	1473	10	3	27	monks-2	432	7	2
8	colic	368	23	2	28	monks-3	432	7	2
9	connect-4	67557	43	3	29	nursery	12960	9	5
10	credit-a	690	16	2	30	pendigits	10992	17	10
11	credit-g	1000	21	2	31	poker-hand	25010	11	10
12	cylinder-bands	540	40	2	32	segment	2310	20	7
13	dermatology	366	35	6	33	solar-flare-1	323	13	2
14	diabetes	768	9	2	34	solar-flare-2	1066	13	3
15	ecoli	336	8	8	35	soybean	683	36	19
16	EEG Eye State	14980	15	2	36	tic-tac-toe	958	10	2
17	heart-disease	303	14	5	37	vehicle	846	19	4
18	ionosphere	351	35	2	38	vote	435	17	2
19	kdd-ipums-97	7019	61	9	39	vowel	990	14	11
20	kdd-ipums-98	7485	61	10	40	wdbc	569	31	2

Note: I, F, and T denote the number of instances, the number of features, and the number of classes, respectively.

5.2. Experimental Setup

5.2.1. Benchmark methods

Since PLAC is a generic framework, existing classification algorithms could be used as its subordinate algorithm. For the purpose of thoroughly evaluating our proposed method PLAC, a series of classification algorithms are incorporated into PLAC respectively, including probability-based naive Bayes NB [9], tree-based C4.5 [1], support vector machines SVM [10], rule-based OneR [6], and instance-based IB1 [7]-[8]. In such a way, we could evaluate the effectiveness of PLAC against its subordinate algorithms. Since ensembling methods also employ some subordinate classification algorithms, for fair comparison, we also compare PLAC against Bagging [17] and Boosting [18]. Meanwhile, we also employ the lazy decision tree algorithm LAZYDT as a comparative algorithm, in which a path in a ``best'' tree for a given new instance is constructed to predict the label of the new instance.

5.2.2. Experimental procedure

In order to make the best use of the data and obtain stable results, a (M=10)×(N=10)-cross-validation is used for estimating the performance of each classification method. That is, each data set is first divided into N bins, after that, a classifier is trained using (N-1) bins, and then tested on the remaining one. This procedure is repeated for the N folds, so that each bin is used for training and testing while minimizing sampling bias. To overcome any ordering effect and to achieve reliable statistics, each holdout experiment is also repeated M times, and in each repetition the data sets are randomized. Hence in general, M×N classifiers are all built during the period of evaluation, thus M×N results are obtained on each data set about the performance of each classification method. The Experimental Process above shows the details of the experimental procedure.

Function Experimental Process

```
1.  M ← 10: /*the number of repetitions */
2.
3.  N ← 10: /*the number of repetitions */
4.
5.  DATA ← {D1, D2, ..., D40};
6.  Learners ← {SVM, OneR, NB, C4.5, IB1,LAZYDT};
7.  for each data ∈ DATA do
8.     for each times ∈ [1, M] do
9.        data' ← randomize instance-order for data;
10.       binData ← generate numFolds bins from data';
11.       for each fold ∈ [1,N] do
12.          testData ← binData[fold];
13.          trainingData ←data'- testData;
14.          for each learner ∈ Learners do classifier ← learner(trainingData);
15.             Accuracy ← classifier(testData);
```

5.3. Accuracy Comparison

Tables 2, 3, 4, 5, and 6 show the classification accuracy and the corresponding Win/Draw/Loss records of single learning methods NB, C4.5, SVM, IB1, OneR, and LAZYDT, and our PLAC and ensemble learning Bagging and Boosting with these single methods excluding LAZYDT as the base learner(s), respectively.

From Tables 2 - 6 we observe that

1) The classification accuracy of our PLAC varies from 82.06% to 86.8% with different single learning methods as its base learner. When SVM is chosen (as the base learner), PLAC achieves the best accuracy 86.8% that outperforms all the other classification methods over the 40 data sets. Since PLAC is not specific to any individual classification algorithm, SVM should be selected as its base learner if accuracy is most concerned.

2) Compared to the five eager learning methods, our PLAC performs best over the 40 data sets. Specifically, PLAC improved the classification accuracy of NB, C4.5, SVM, IB1, and OneR by 6.81%, 1.93%, 4.20%, 3.66%, and 14.70% on average, respectively. Surprisingly, the accuracy of C4.5 was enhanced as well. This is because C4.5 induces a single decision tree for all the new instances with the entire training data, which means it does not take into account the specific characteristics of a new instance. When using C4.5 as the base learner, PLAC builds a classifier for each new instance with the most relevant part of the training data. Since PLAC customizes the classifier for each new instance, it could be more accurate.

Table 2. Accuracy of LAZYDT, NB, and Bagging, Boosting and PLAC with NB as the Base Learner over the 40 Data Sets

Data	LAZYDT	NB	Bagging	Boosting	PLAC
adult-census	84.74	83.43	83.35	83.43	83.37
anneal	98.40	86.61	87.64	93.85	98.31
anneal-ORIG	90.29	75.34	78.17	80.02	94.92
balance-scale	77.86	90.62	90.08	91.81	90.62
breast-w	94.96	96.05	96.05	95.51	96.40
car	92.34	85.6	85.20	90.35	89.63
cmc	51.12	50.42	49.83	50.42	54.89
colic	84.61	78.65	79.03	77.17	83.38
connect-4	80.91	72.14	72.21	72.14	75.34
credit-a	84.41	77.83	77.77	81.36	84.78
credit-g	71.42	75.30	75.46	75.26	75.30
cylinder-bands	57.89	73.96	72.81	77.59	76.19
dermatology	93.94	97.49	97.43	96.77	97.49
diabetes	74.69	75.68	75.83	75.89	76.00
ecoli	82.86	85.48	85.59	85.48	85.48
EEG Eye State	57.58	46.77	47.05	46.77	74.56
heart-disease	76.14	83.29	83.35	83.63	83.29
ionosphere	92.66	82.40	82.34	90.61	91.34
kdd-ipums-97	77.11	70.17	70.93	69.58	76.78
kdd-ipums-98	76.26	72.68	73.01	66.49	76.32
kdd-ipums-99	86.02	83.03	83.45	80.10	85.95
kdd-Japanese	86.86	83.74	83.69	86.13	91.43
kr-vs-kp	99.44	99.44	99.37	99.61	99.44
letter	86.50	64.12	64.30	64.12	83.97
liver-disorders	64.60	54.80	54.96	65.23	67.98
monks-1	97.13	75.01	75.01	73.80	89.22
monks-2	67.13	66.16	65.28	65.70	66.16
monks-3	100.00	97.22	97.22	99.81	100.00
nursery	97.16	90.32	90.23	91.82	94.05
pendigits	96.56	85.75	85.74	85.75	96.21
poker-hand	57.94	49.88	49.81	49.88	54.22
segment	95.23	80.12	80.29	80.12	92.75
solar-flare-1	97.84	92.95	93.88	96.41	97.59
solar-flare-2	99.54	97.58	97.75	99.20	99.61
soybean	91.33	93.00	92.82	92.97	93.47
tic-tac-toe	84.97	69.52	70.19	81.57	83.63
vehicle	69.90	44.42	45.39	44.42	73.03
vote	96.55	90.02	89.88	94.89	94.21
vowel	80.59	62.59	62.95	79.90	82.81
wdbc	93.50	93.19	93.26	95.78	95.08
Mean	83.72	78.32	78.47	80.28	85.13
Win/Draw/Loss	18/2/20	1/7/32	3/0/37	8/1/31	

Note: A Win/Draw/Loss record indicates how many data sets the learner in a column has accuracy higher than/equal to/lower than that of PLAC.

Table 3. Accuracy of LAZYDT, C4.5, and Bagging, Boosting and PLAC with C4.5 as the Base Learner over the 40 Data Sets

Data	LAZYDT	C4.5	Bagging	Boosting	PLAC
adult-census	84.74	86.23	86.12	83.58	86.49
anneal	98.40	98.51	98.84	99.69	98.89
anneal-ORIG	90.29	91.85	94.26	95.19	95.28
balance-scale	77.86	77.89	83.13	78.58	78.21
breast-w	94.96	95.02	96.25	95.91	95.11
car	92.34	92.34	93.32	95.93	92.34
cmc	51.12	51.57	51.93	50.41	53.57
colic	84.61	85.10	85.42	81.95	85.15
connect-4	80.91	80.90	82.71	82.73	80.97
credit-a	84.41	85.88	86.38	84.41	86.41
credit-g	71.42	71.30	73.42	70.20	74.70
cylinder-bands	57.89	57.78	58.04	57.78	70.48
dermatology	93.94	94.11	96.01	95.90	94.11
diabetes	74.69	74.85	75.42	72.13	75.05
ecoli	82.86	83.34	84.29	82.63	85.01
EEG Eye State	57.58	84.50	90.51	91.70	85.13
heart-disease	76.14	76.97	79.59	78.29	78.67
ionosphere	92.66	89.63	92.54	92.60	89.80
kdd-ipums-97	77.11	77.11	77.11	68.22	77.11
kdd-ipums-98	76.26	76.25	76.25	76.25	76.25
kdd-ipums-99	86.02	86.02	86.02	83.23	86.11
kdd-Japanese	86.86	87.37	93.22	96.75	87.88
kr-vs-kp	99.44	99.44	99.37	99.61	99.44
letter	86.50	87.98	92.57	95.54	88.11
liver-disorders	64.60	66.38	70.61	69.03	67.25
monks-1	97.13	97.13	100.00	100.00	97.13
monks-2	67.13	67.14	56.95	59.40	67.14
monks-3	100.00	100.00	100.00	100.00	100.00
nursery	97.16	97.05	97.31	99.51	97.05
pendigits	96.56	96.58	98.07	99.07	96.73
poker-hand	57.94	54.65	58.70	57.56	55.19
segment	95.23	96.76	97.22	98.18	96.92
solar-flare-1	97.84	97.84	97.65	96.29	97.84
solar-flare-2	99.54	99.53	99.53	98.97	99.53
soybean	91.33	91.33	92.56	92.53	91.68
tic-tac-toe	84.97	84.97	92.90	96.43	85.30
vehicle	69.90	71.87	74.66	75.93	74.30
vote	96.55	96.55	96.27	95.25	96.55
vowel	80.59	80.04	89.33	93.03	80.34
wdbc	93.50	93.11	95.68	95.64	94.27
Mean	83.72	83.51	86.25	85.90	85.44
Win/Draw/Loss	6/7/27	0/12/28	23/4/13	22/2/16	

Table 4. Accuracy of LAZYDT, SVM, and Bagging, Boosting and PLAC with SVM as the Base Learner over the 40 Data Sets

Data	LAZYDT	SVM	Bagging	Boosting	PLAC
adult-census	84.74	84.91	84.50	84.89	82.74
anneal	98.40	97.46	97.84	99.36	99.24
anneal-ORIG	90.29	87.46	89.06	90.00	91.91
balance-scale	77.86	87.78	87.42	87.68	89.31
breast-w	94.96	96.71	96.74	96.65	97.00
car	92.34	93.68	93.50	94.29	95.74
cmc	51.12	48.56	49.22	48.68	55.52
colic	84.61	82.49	83.09	79.13	85.11
connect-4	80.91	NA	NA	NA	79.16
credit-a	84.41	85.01	85.42	83.45	86.78

credit-g	71.42	75.06	75.44	75.06	75.08
cylinder-bands	57.89	81.52	80.41	82.04	82.81
dermatology	93.94	96.11	97.1	96.06	96.11
diabetes	74.69	76.85	77.37	76.85	76.85
ecoli	82.86	83.57	84.28	85.00	83.93
EEG Eye State	57.58	55.13	55.95	55.77	84.44
heart-disease	76.14	83.69	83.49	83.82	83.69
ionosphere	92.66	88.10	88.50	89.23	90.37
kdd-ipums-97	77.11	69.10	75.29	69.64	72.96
kdd-ipums-98	76.26	70.97	75.26	70.85	73.31
kdd-ipums-99	86.02	82.79	85.80	82.05	84.89
kdd-Japanese	86.86	94.9	94.15	94.57	93.24
kr-vs-kp	99.44	95.76	96.07	97.00	99.50
letter	86.50	82.34	81.05	82.34	89.01
liver-disorders	64.60	58.04	58.80	64.09	67.96
monks-1	97.13	75.01	75.01	85.14	90.51
monks-2	67.13	67.14	67.14	72.47	78.99
monks-3	100.00	100.00	99.81	100.00	100.00
nursery	97.16	93.08	93.00	93.07	98.46
pendigits	96.56	97.91	97.73	98.20	96.62
poker-hand	57.94	46.93	46.44	46.93	48.15
segment	95.23	92.94	92.88	92.98	96.09
solar-flare-1	97.84	97.53	97.41	95.98	97.53
solar-flare-2	99.54	99.51	99.50	99.08	99.51
soybean	91.33	93.03	93.03	92.76	93.53
tic-tac-toe	84.97	98.33	98.33	98.14	98.33
vehicle	69.9	74.37	74.73	74.35	77.09
vote	96.55	95.72	96.18	95.67	96.23
vowel	80.59	70.71	70.38	78.38	86.59
wdbc	93.50	97.54	97.65	97.19	97.54
Mean	83.72	82.60	1900/3/23	84.33	86.80
Win/Draw/Loss	11/1/28	3/8/29	11/1/28	6/2/32	

Table 5. Accuracy of LAZYDT, IB1, and Bagging, Boosting and PLAC with IB1 as the base Learner over the 40 Data Sets

Data	LAZYDT	IB1	Bagging	Boosting	PLAC
adult-census	84.74	79.42	80.04	78.36	81.12
anneal	98.40	99.11	98.71	99.26	99.51
anneal-ORIG	90.29	95.50	95.25	96.55	97.21
balance-scale	77.86	78.4	81.95	72.00	80.13
breast-w	94.96	95.37	95.45	95.28	95.79
car	92.34	77.37	87.59	88.67	82.59
cmc	51.12	43.93	44.92	43.92	48.61
colic	84.61	79.52	80.01	79.14	82.17
connect-4	80.91	66.35	77.58	NA	69.71
credit-a	84.41	81.71	82.70	80.52	82.06
credit-g	71.42	71.98	71.76	67.76	71.98
cylinder-bands	57.89	75.41	74.07	73.93	78.37
dermatology	93.94	94.69	94.8	92.23	94.69
diabetes	74.69	70.88	71.74	67.91	72.13
ecoli	82.86	80.78	82.98	79.47	81.56
EEG Eye State	57.58	83.65	90.97	81.88	94.53
heart-disease	76.14	76.25	76.65	73.67	78.15
ionosphere	92.66	87.24	87.64	87.35	91.05
kdd-ipums-97	77.11	65.53	68.52	61.55	65.85
kdd-ipums-98	76.26	64.34	67.91	59.93	64.53
kdd-ipums-99	86.02	80.16	82.30	76.12	80.16
kdd-Japanese	86.86	99.84	99.68	99.51	99.84
kr-vs-kp	99.44	90.49	94.83	95.92	96.98
letter	86.50	96.00	95.93	94.89	95.97

liver-disorders	64.60	61.88	61.23	62.45	63.12
monks-1	97.13	72.04	89.44	91.85	75.88
monks-2	67.13	57.28	57.65	47.37	59.54
monks-3	100.00	79.12	95.42	96.57	97.50
nursery	97.16	78.72	94.16	82.21	89.77
pendigits	96.56	99.36	99.39	99.17	99.34
poker-hand	57.94	45.32	45.84	44.88	46.13
segment	95.23	97.13	96.81	96.66	97.48
solar-flare-1	97.84	95.62	96.73	95.50	95.8
solar-flare-2	99.54	99.21	99.29	99.14	99.23
soybean	91.33	90.24	90.89	90.86	91.42
tic-tac-toe	84.97	81.06	95.45	94.57	85.32
vehicle	69.90	70.00	70.35	68.70	73.00
vote	96.55	92.09	92.73	92.50	95.31
vowel	80.59	99.07	98.61	98.77	99.07
wdbc	93.50	95.75	96.17	94.73	95.96
Mean	83.72	80.05	84.10	82.10	83.71
Win/Draw/Loss	22/0/18	2/5/33	16/0/24	3/0/37	

Table 6. Accuracy of LAZYDT, OneR, and Bagging, Boosting and PLAC with OneR as the Base Learner over the 40 Data Sets

Data	LAZYDT	OneR	Bagging	Boosting	PLAC
adult-census	84.74	80.91	74.71	74.96	84.23
anneal	98.40	83.63	83.63	84.95	98.84
anneal-ORIG	90.29	83.63	83.63	84.95	95.75
balance-scale	77.86	56.99	69.06	73.50	76.86
breast-w	94.96	92.11	92.91	95.34	95.05
car	92.34	70.02	70.02	73.13	80.85
cmc	51.12	47.68	46.15	43.91	54.57
colic	84.61	81.51	81.51	78.40	84.28
connect-4	80.91	66.14	66.14	66.14	70.18
credit-a	84.41	85.51	85.51	78.58	87.04
credit-g	71.42	66.20	68.04	63.80	72.76
cylinder-bands	57.89	50.11	47.96	51.59	80.93
dermatology	93.94	50.16	50.39	46.44	91.03
diabetes	74.69	71.51	71.80	69.16	74.77
ecoli	82.86	67.73	65.19	65.76	81.38
EEG Eye State	57.58	62.60	62.56	66.46	73.57
heart-disease	76.14	72.86	77.21	73.22	79.89
ionosphere	92.66	81.99	84.90	87.35	89.46
kdd-ipums-97	77.11	72.27	70.75	67.28	72.27
kdd-ipums-98	76.26	73.44	74.04	73.10	74.78
kdd-ipums-99	86.02	84.99	85.01	84.88	85.34
kdd-Japanese	86.86	31.18	34.53	27.88	75.67
kr-vs-kp	99.44	66.92	66.23	93.02	96.01
letter	86.50	17.24	17.56	16.64	62.22
liver-disorders	64.60	54.89	58.78	60.32	67.25
monks-1	97.13	75.01	75.01	75.01	90.70
monks-2	67.13	67.14	67.14	65.79	67.14
monks-3	100.00	79.94	80.54	98.80	100.00
nursery	97.16	70.97	70.97	83.16	89.39
pendigits	96.56	38.91	41.65	38.20	89.20
poker-hand	57.94	49.95	49.94	49.35	53.67
segment	95.23	64.16	67.70	79.52	94.90
solar-flare-1	97.84	97.65	97.84	97.28	97.65
solar-flare-2	99.54	99.44	99.50	99.27	99.53
soybean	91.33	39.91	40.41	40.44	87.70
tic-tac-toe	84.97	69.94	69.94	72.98	80.11
vehicle	69.90	52.60	51.85	50.38	70.38
vote	96.55	95.63	95.49	95.36	95.63

vowel	80.59	32.61	36.34	31.09	67.35
wdbc	93.50	88.44	91.21	91.99	94.06
Mean	83.72	67.36	68.10	69.23	82.06
Win/Draw/Loss	25/1/14	0/4/36	1/1/38	1/0/39	

As we know, mean is a central tendency statistical measure, and is often used to compare the general performance of different learning methods. Here, it distinguishes the 21 learners viewing all the 40 data sets as a whole. To compensate this, we further employ statistics Win/Draw/Loss record [26] to differentiate these learners based on one-tone comparison from another perspective. A Win/Draw/Loss record presents three values, the number of data sets for which one learner obtained better, equal, or worse performance than another learner in terms of a given measure.

The last rows of Tables 2 - 6 shows the Win/Draw/Loss records of the corresponding learners, where each record indicates how many data sets the learner in a column has accuracy higher than/equal to/lower than that of PLAC. The Win/Draw/Loss records reveal that the classification accuracy of PLAC is better than those of the five eager learners on 32, 28, 29, 33 and 36 out of 40 data sets, respectively. This confirmed the conclusion drawn from the classification accuracy.

1) As for the lazy learning method L_{AZY}DT, its classification accuracy was improved by PLAC with NB, C4.5, and SVM as its base learners by 1.41%, 1.72%, and 3.08%, respectively. In contrast, L_{AZY}DT outperforms PLAC with IB1 and OneR as base learners by 0.01% and 1.66%. It indicates that PLAC performs better than L_{AZY}DT in most cases. This finding is also consistent with the fact that no method can perform better than others in all settings [27].
2) PLAC defeats ensemble learning method Bagging with NB, SVM, and OneR as the base learner in terms of classification accuracy, this is consistent with the Win/Draw/Loss records. PLAC also wins ensemble learning method Boosting with NB, SVM, IB1, and OneR as the base learner in terms of classification accuracy. Again, this is supported by the Win/Draw/Loss records. An exception occurs when C4.5 is chosen as the base learner, the classification accuracy of PLAC becomes slightly lower than those of Bagging and Boosting.
3) For very simple classification algorithms such as OneR, the classification accuracy was greatly improved by lazy learning method PLAC by 14.70%, while ensemble learning methods Bagging and Boosting were only able to improve it by at most 1.87%. This reveals that the first choice for enhancing the performance of very simple algorithms should be lazy learning methods, even if they are often used as the base learners in ensemble learning such as Bagging and Boosting [28].

5.4. Runtime Comparison

Tables 7, 8, 9, 10, and 11 show the classification time and the corresponding Win/Draw/Loss records of the single learning methods NB, C4.5, SVM, IB1, OneR, and LAZYDT, and our PLAC and ensemble learning Bagging and Boosting with these single methods excluding L_{AZY}DT as the base learner(s), respectively.

Table 7. Runtime of L_{AZY}DT, NB, and Bagging, Boosting and PLAC with NB as the Base Learner over the 40 Data Sets

Data	L_{AZY}DT	NB	Bagging	Boosting	PLAC
adult-census	3420.02	68.08	341.96	1458.96	291.28
anneal	38.54	4.88	31.05	468.94	5.86
anneal-ORIG	73.13	2.73	18.40	238.80	4.74
balance-scale	10.59	0.49	3.49	26.06	0.42
breast-w	17.83	0.84	6.69	36.27	1.91
car	4.32	0.69	5.71	64.43	1.07
cmc	25.74	0.97	8.41	35.50	1.33
colic	15.18	0.46	3.82	25.77	2.10

connect-4	3962.60	141.81	1045.45	11566.84	2039.42
credit-a	7.48	1.08	6.56	35.55	3.09
credit-g	18.49	1.16	10.97	89.36	1.24
cylinder-bands	9.25	1.34	12.65	101.31	4.98
dermatology	2.20	0.87	8.52	124.46	0.97
diabetes	23.57	0.77	7.27	38.31	2.27
ecoli	10.36	0.86	7.98	33.22	0.80
EEG Eye State	746.62	36.88	264.87	115.46	175.61
heart-disease	7.91	0.89	7.42	94.00	0.83
ionosphere	42.65	1.77	12.90	86.23	6.03
kdd-ipums-97	379.01	32.67	303.49	2457.76	37.59
kdd-ipums-98	172.58	36.86	370.29	3317.41	49.04
kdd-ipums-99	208.80	40.66	406.59	3708.29	54.54
kdd-Japanese	298.36	17.25	164.93	1596.79	115.59
kr-vs-kp	25.82	13.05	81.34	178.56	6.87
letter	7116.95	211.29	1680.94	15795.45	600.44
liver-disorders	14.75	0.28	2.59	12.63	0.49
monks-1	1.53	0.15	1.20	9.56	0.46
monks-2	1.89	0.13	1.14	3.10	0.15
monks-3	1.12	0.15	1.16	9.64	0.39
nursery	43.19	11.79	63.47	751.17	13.70
pendigits	1510.68	60.86	506.07	4301.14	181.91
poker-hand	1958.39	61.28	485.46	396.29	163.64
segment	140.57	12.02	110.24	356.93	47.12
solar-flare-1	1.51	0.17	1.39	13.31	0.37
solar-flare-2	2.30	0.57	5.10	63.90	0.80
soybean	12.44	4.17	38.37	628.79	4.11
tic-tac-toe	3.99	0.32	2.98	30.24	0.93
vehicle	80.09	2.39	22.10	12.03	9.71
vote	2.85	0.23	2.15	21.78	0.77
vowel	52.32	3.62	33.51	432.06	24.24
wdbc	43.31	2.20	20.38	123.48	11.10
Mean	512.72	19.47	152.73	1221.49	96.70
Win/Draw/Loss	6/0/34	35/0/5	1/0/39	1/0/39	

Note: A Win/Draw/Loss record indicates how many data sets the learner in a column has time less than/ equal to/greater than that of PLAC.

Table 8. Runtime of LAZYDT, C4.5, and Bagging, Boosting and PLAC with C4.5 as the Base Learner over the 40 Data Sets

Data	LAZYDT	C4.5	Bagging	Boosting	PLAC
adult-census	3420.02	1481.30	8331.87	17248.19	455.90
anneal	38.54	10.81	60.55	54.33	5.11
anneal-ORIG	73.13	11.45	90.04	71.45	5.80
balance-scale	10.59	1.97	17.20	26.52	1.50
breast-w	17.83	2.57	16.27	27.59	1.73
car	4.32	1.21	11.70	23.29	1.16
cmc	25.74	9.34	83.68	96.15	5.68
colic	15.18	2.76	23.22	42.26	2.73
connect-4	3962.60	3118.18	24451.09	33185.64	2405.84
credit-a	7.48	3.81	30.38	47.89	3.44
credit-g	18.49	6.96	57.18	77.15	5.95
cylinder-bands	9.25	1.55	17.55	5.28	6.74
dermatology	2.20	1.27	10.15	14.48	1.08
diabetes	23.57	4.39	53.36	65.49	3.89
ecoli	10.36	1.94	15.66	25.19	1.61
EEG Eye State	746.62	771.66	5311.33	9097.05	357.72
heart-disease	7.91	1.53	12.94	19.92	1.49
ionosphere	42.65	11.65	80.39	105.98	7.67
kdd-ipums-97	379.01	88.52	1682.68	1053.68	49.74
kdd-ipums-98	172.58	57.84	609.46	197.36	65.70

kdd-ipums-99	208.80	80.03	767.21	1224.49	92.35
kdd-Japanese	298.36	197.53	1503.60	2129.97	158.45
kr-vs-kp	25.82	13.05	81.34	178.56	6.87
letter	7116.95	1267.86	6722.09	8810.25	381.74
liver-disorders	14.75	1.69	17.53	15.31	1.59
monks-1	1.53	0.32	3.62	1.97	0.38
monks-2	1.89	0.51	5.49	6.85	0.53
monks-3	1.12	0.15	1.48	0.37	0.31
nursery	43.19	26.27	133.06	267.04	15.70
pendigits	1510.68	362.92	1767.21	2268.89	164.31
poker-hand	1958.39	1916.52	7563.29	9849.47	283.44
segment	140.57	40.80	314.91	461.13	31.84
solar-flare-1	1.51	0.24	1.71	3.43	0.24
solar-flare-2	2.30	0.45	3.36	7.67	0.48
soybean	12.44	4.18	35.17	54.77	3.69
tic-tac-toe	3.99	1.21	11.73	15.67	1.18
vehicle	80.09	14.22	133.51	166.46	12.28
vote	2.85	0.71	5.17	9.60	0.93
vowel	52.32	29.29	215.95	309.70	24.37
wdbc	43.31	11.62	85.60	128.56	9.27
Mean	512.72	239.01	1508.49	2184.88	114.51
Win/Draw/Loss	1/0/39	8/1/32	0/0/40	1/0/39	

Table 9. Runtime of LAZYDT, SVM, and Bagging, Boosting and PLAC with SVM as the Base Learner over the 40 Data Sets

Data	LAZYDT	SVM	Bagging	Boosting	PLAC
adult-census	3420.02	513229.69	964630.97	3714043.61	4223.91
anneal	38.54	70.27	447.75	418.53	27.01
anneal-ORIG	73.13	62.02	409.49	494.40	49.65
balance-scale	10.59	6.81	68.00	55.33	9.24
breast-w	17.83	3.03	28.62	63.26	3.87
car	4.32	94.43	933.74	1694.70	38.07
cmc	25.74	208.58	2077.69	548.63	96.27
colic	15.18	27.02	267.18	268.38	18.66
connect-4	3962.60	NA	NA	NA	6690.74
credit-a	7.48	144.98	724.78	560.55	18.10
credit-g	18.49	198.50	2211.58	1912.95	100.92
cylinder-bands	9.25	280.01	2040.18	1233.07	103.84
dermatology	2.20	30.23	293.34	44.36	29.82
diabetes	23.57	4.99	53.19	31.03	5.13
ecoli	10.36	27.77	253.01	154.10	43.05
EEG Eye State	746.62	1107.81	3038.84	1460.33	1904.19
heart-disease	7.91	6.06	68.18	69.07	6.19
ionosphere	42.65	6.90	72.78	74.04	13.50
kdd-ipums-97	379.01	51787.50	90337.50	409065.63	3523.02
kdd-ipums-98	172.58	250957.81	434540.63	1965918.75	14348.35
kdd-ipums-99	208.80	271725.00	449712.50	2139904.69	18935.24
kdd-Japanese	298.36	400.00	2000.00	2809.38	693.85
kr-vs-kp	25.82	528.28	5493.94	9734.90	31.13
letter	7116.95	5823.44	34660.09	14493.94	10932.67
liver-disorders	14.75	1.94	19.97	22.20	12.29
monks-1	1.53	7.67	87.62	101.18	14.23
monks-2	1.89	6.81	77.99	64.22	5.82
monks-3	1.12	6.18	55.33	6.56	3.39
nursery	43.19	14967.19	33113.67	200901.65	273.59
pendigits	1510.68	946.88	6681.26	10471.40	963.95
poker-hand	1958.39	210950.00	391813.47	96273.90	8710.99
segment	140.57	61.10	612.54	495.73	54.79
solar-flare-1	1.51	3.34	25.75	38.76	3.38
solar-flare-2	2.30	8.80	64.53	83.84	6.23

soybean	12.44	256.08	2445.59	2933.19	140.14
tic-tac-toe	3.99	78.96	769.96	832.41	83.17
vehicle	80.09	26.25	273.67	144.70	29.09
vote	2.85	5.39	54.65	70.94	5.92
vowel	52.32	183.80	1924.57	1734.22	169.37
wdbc	43.31	2.92	31.04	46.89	3.10
Mean	512.72	33954.98	62369.63	219981.42	1808.15
Win/Draw/Loss	25/0/15	17/0/23	0/0/40	1/0/39	

From Tables 7 - 11 we achieve the following findings.

1) The runtime of PLAC differs when different single learning methods are used as its base learner. With OneR and SVM as the base learner, PLAC achieves its shortest runtime of 78.90 and longest runtime of 1808.15 milliseconds, respectively, with the difference of up to 1729.25 milliseconds, which is about 22 times of the shortest one. The reason behind this huge difference is that of PLAC's time complexity $O(n\times log(n)) + O(classificationAlg(nearPart))$, the latter $O(classificationAlg(nearPart))$ is contributed by the base learner, and the time complexity of OneR and SVM are $O(m)$ and $O(n^3)$, respectively. Thus, the time complexity of PLAC with OneR and SVM as the base learner are $O(n\times log(n)) + O(m)$ and $O(n\times log(n)) + O(n3)$, respectively. Since $m \leq n$, $O(m) \leq O(n3)$, further $O(n\times log(n)) + O(m) \leq O(n\times log(n)) + O(n3)$.
2) Compared to eager learning methods SVM, IB1, and C4.5, the runtime of PLAC with them as the base learner is only 5.33%, 10.79%, and 47.91% of those that are used as a single learning method, respectively. This means that PLAC can be faster than a single learning method, which could be interpreted as follows. Because the time complexity of PLAC consists of two parts, the first part $O(n\times log(n))$ is intrinsic to PLAC itself while the second part $O(classificationAlg(nearPart))$ depends on the base learner *classificationAlg*. Considering that the data partitioning of PLAC is applied in the entire training data and base learner *classificationAlg* utilizes only a very small part to build a classifier, the main part of the complexity is $O(n\times log(n))$. This means if *classificationAlg*'s complexity $O(classificationAlg)$ is greater than $O(n\times log(n))$, then PLAC with *classificationAlg* as the base learner is faster than *classificationAlg* as a single method. For instance, SVM's time complexity $O(n^3)$ is greater than $O(n\times log(n))$, and as PLAC's base learner, SVM is only used to build a classifier with the nearest partition of a new instance, thus PLAC is more efficient than SVM.
   With the above explanation we can easily understand why PLAC with NB and OneR as the base learner is slower than NB and OneR used as a single learner, respectively. Fortunately, with NB and OneR as the base learner of PLAC, the longest runtime is only 96.7 milliseconds, which is still very efficient
3) Both the mean runtime and the Win/Draw/Loss records demonstrate that PLAC with three base learners, namely NB, C4.5, and OneR, are much more efficient than $L_{AZY}DT$.
4) Both PLAC and ensemble learning methods employ a single method as the base learner, but the former is much more efficient than the latter and this conclusion is confirmed by the Win/Draw/Loss records. For example, if C4.5 is used as the base learner, PLAC's runtime is only 7.59% of Bagging and 5.24% of Boosting, separately. Therefore, the efficiencies of Bagging and Boosting are greatly improved. The reason is that compared with ensemble learning that builds several classifiers, PLAC only uses one of the multiple partitions to construct a single classifier, thus significantly reducing the induction and deduction time.

5.5. Performance Validation and Summary

5.5.1. Validation method

The nonparametric Friedman test [29] is often used to compare k algorithms over *N* data sets by ranking each algorithm on each data set separately. The algorithm achieving the best performance gets the rank of 1, the second best ranks 2, and so on. In case of ties, average ranks are assigned. Then the average ranks of all algorithms on all data sets are calculated and compared. If the null hypothesis, which is that all algorithms are performing equivalently, is rejected under the Friedman test statistic, post-hoc tests such as the Nemenyi test [30] can be used to determine which algorithms perform statistically different.

Table 10. Runtime of L_{AZY}DT, IB1 and Bagging, Boosting and PLAC with IB1 as the Base Learner over the 40 Data Sets

Data	L_{AZY}DT	IB1	Bagging	Boosting	PLAC
adult-census	3420.02	19327.50	149871.55	2047935.76	1120.71
anneal	38.54	16.96	166.08	2898.60	0.99
anneal-ORIG	73.13	11.09	119.72	2065.59	0.94
balance-scale	10.59	1.32	13.24	236.02	0.85
breast-w	17.83	3.34	35.11	303.69	0.95
car	4.32	16.09	153.26	2794.47	0.84
cmc	25.74	14.87	142.82	2626.31	0.60
colic	15.18	2.38	20.94	221.22	0.81
connect-4	3962.60	157766.84	1416972.46	NA	10789.69
credit-a	7.48	5.27	50.59	757.98	0.82
credit-g	18.49	14.72	146.30	2763.12	0.66
cylinder-bands	9.25	7.64	74.07	1372.94	0.77
dermatology	2.20	3.10	29.94	492.18	0.97
diabetes	23.57	3.88	40.09	752.31	0.69
ecoli	10.36	0.94	7.49	132.02	0.88
EEG Eye State	746.62	4270.69	29608.60	846608.68	556.74
heart-disease	7.91	1.00	9.65	175.40	0.78
ionosphere	42.65	2.77	27.11	504.09	0.89
kdd-ipums-97	379.01	542.75	5250.70	54228.09	109.44
kdd-ipums-98	172.58	805.48	6800.84	75491.21	30.78
kdd-ipums-99	208.80	1210.78	10423.03	117102.15	1309.40
kdd-Japanese	298.36	218.00	2096.20	37637.42	201.45
kr-vs-kp	25.82	247.91	2328.36	41415.79	0.97
letter	7116.95	8390.73	66542.06	2131676.24	6220.10
liver-disorders	14.75	0.68	6.65	120.09	0.62
monks-1	1.53	1.07	10.32	187.60	0.76
monks-2	1.89	1.05	10.09	189.68	0.54
monks-3	1.12	1.07	10.28	186.35	0.97
nursery	43.19	1604.18	12158.34	971552.86	152.39
pendigits	1510.68	2147.57	18528.94	903302.84	1309.40
poker-hand	1958.39	9000.32	64560.96	1885927.93	386.24
segment	140.57	85.46	825.37	14942.21	0.99
solar-flare-1	1.51	0.87	8.01	116.31	0.34
solar-flare-2	2.30	8.56	82.87	1221.24	0.27
soybean	12.44	9.41	90.52	1451.89	0.95
tic-tac-toe	3.99	7.18	66.73	1235.43	0.83
vehicle	80.09	10.75	106.61	1967.18	0.82
vote	2.85	2.43	20.75	316.56	0.95
vowel	52.32	11.42	108.04	1905.44	0.99
wdbc	43.31	8.47	76.06	1430.21	0.96
Mean	512.72	5144.66	44690.02	234775.52	555.24
Win/Draw/Loss	5/0/35	1/0/39	0/0/40	0/0/40	

Table 11. Runtime of LAZYDT, OneR and Bagging, Boosting and PLAC with OneR as the Base Learner over the 40 Data Sets

Data	LAZYDT	OneR	Bagging	Boosting	PLAC
adult-census	3420.02	121.32	416.64	595.90	279.50
anneal	38.54	1.61	5.15	8.06	4.22
anneal-ORIG	73.13	1.93	5.19	7.85	2.72
balance-scale	10.59	0.28	1.77	2.20	1.04
breast-w	17.83	0.56	3.66	5.49	1.25
car	4.32	0.53	1.75	4.48	0.58
cmc	25.74	0.85	3.57	0.98	1.53
colic	15.18	0.68	2.45	3.53	0.62
connect-4	3962.60	664.19	3531.38	2537.77	1962.21
credit-a	7.48	1.01	4.32	6.07	0.91
credit-g	18.49	1.15	5.82	8.84	2.09
cylinder-bands	9.25	1.17	7.69	8.69	4.02
dermatology	2.20	0.20	1.48	0.50	1.04
diabetes	23.57	0.82	5.33	7.50	3.45
ecoli	10.36	0.35	2.04	2.60	1.06
EEG Eye State	746.62	53.35	191.40	275.11	171.81
heart-disease	7.91	0.34	1.84	2.73	0.74
ionosphere	42.65	2.29	11.17	10.90	7.90
kdd-ipums-97	379.01	7.75	22.44	21.03	2.87
kdd-ipums-98	172.58	5.64	33.00	37.28	15.76
kdd-ipums-99	208.80	7.84	35.87	47.71	30.72
kdd-Japanese	298.36	12.08	87.48	11.29	104.09
kr-vs-kp	25.82	2.41	13.90	19.63	6.42
letter	7116.95	77.05	232.32	69.11	240.25
liver-disorders	14.75	0.29	1.78	2.74	1.08
monks-1	1.53	0.06	0.51	1.60	0.35
monks-2	1.89	0.07	0.52	1.60	0.14
monks-3	1.12	0.06	0.50	1.68	0.33
nursery	43.19	6.46	31.11	87.33	13.77
pendigits	1510.68	32.84	140.09	29.30	115.92
poker-hand	1958.39	57.09	128.13	58.85	111.55
segment	140.57	6.07	44.70	54.04	24.24
solar-flare-1	1.51	0.06	0.53	1.40	0.15
solar-flare-2	2.30	0.19	1.44	4.07	0.44
soybean	12.44	0.33	2.22	0.78	2.23
tic-tac-toe	3.99	0.12	1.18	3.70	0.92
vehicle	80.09	1.24	11.43	8.19	8.11
vote	2.85	0.09	0.82	1.92	0.19
vowel	52.32	1.88	15.47	2.15	18.25
wdbc	43.31	2.20	19.02	17.35	11.45
Mean	512.72	26.86	125.68	99.30	78.90
Win/Draw/Loss	0/0/40	37/0/3	4/0/36	8/0/32	

The Nemenyi post-hoc test compares algorithms in a pairwise manner. According to this test, the performances of two algorithms are significantly different if the distance of the average ranks exceeds the critical distance $CD_\alpha = q_\alpha\sqrt{\frac{k(k+1)}{6N}}$, where q_α is based on the Studentized range statistic [31] divided by $\sqrt{2}$.

To validate the statistical significance of PLAC performance improvement, we performed a nonparametric Friedman test followed by Nemenyi post-hoc test, as advised by Demsar [32] and Garcia and Herrerato [33] to statistically compare algorithms on multiple data sets. Both tests were applied at the significance level of $\alpha = 0.05$ on all the evaluated data sets.

The comparison results are depicted using Demsar's significance diagrams [32]. The diagram plots classifiers against average performance ranks, where all classifiers are sorted according to their ranks, the '∗' denotes the respective average rank of each classifier and the line segment to the right of each classifier represents its critical difference, which means the classifiers whose '∗' on the right end of the line segment are outperformed significantly. The critical difference is highlighted with a vertical dotted line. If the vertical line locates at the leftmost end of a line segment representing a classifier, then all classifiers right to this line perform significantly worse than the classifier. Otherwise. if the line locates at the rightmost end of a line segment, all classifiers left to this line perform significantly better than this classifier.

5.5.2. Accuracy validation

The first experiment employed Friedman test to validate the statistical significance of accuracy improvements by PLAC. The null hypotheses are that, LAZYDT, each of the five single learning methods (NB, C4.5, SVM, IB1 and OneR), and the methods (Bagging, Boosting and PLAC) with these single methods as the base learner are equivalent in terms of classification accuracy. The test results are all p = 0. This means at α = 0.05, there is evidence to reject the null hypotheses and the classification methods are different in terms of classification accuracy.

In order to further explore classification methods whose accuracy have statistically significant differences, we performed a Nemenyi test. Fig. 4 shows the results.

From Fig. 4 we see that

1) Each of the five single learning methods is on the right side of the line associated with PLAC. This reveals that PLAC performs significantly better than the five single learners NB, C4.5, SVM, IB1, and OneR in terms of classification accuracy.
2) Ensemble learning methods Bagging and Boosting are on the right side of the lines associated with PLAC when base learners are NB, SVM, and OneR, respectively. This means the accuracy improvements of PLAC have statistical significance. However, when IB1 and C4.5 are used as the base learners, the accuracy differences among Bagging, Boosting, and PLAC are not significantly important except for PLAC with IB1 as the base learner which performs significantly better than Boosting.
3) PLAC is better-ranked than lazy learning method $L_{AZY}DT$ in four out of five cases, and performs significantly better than $L_{AZY}DT$ when C4.5 or SVM is chosen as the base learner. Since PLAC is not limited to any particular single learner, C4.5 and SVM are recommended to be used as PLAC's base learner if classification accuracy is being pursued.

5.5.3. Runtime validation

The second experiment used Friedman test to validate the statistical significance of runtime improvement by PLAC. The null hypotheses are that, $L_{AZY}DT$, each of the five single learning methods (NB, C4.5, SVM, IB1 and OneR), and the methods (Bagging, Boosting and PLAC) with these single methods as the base learner are equivalent in terms of classification time. The test results are all $p = 0$. This means at $\alpha = 0.05$, there is evidence to reject the null hypotheses and the classification methods are different in terms of runtime.

In order to further explore classification methods whose runtime have statistically significant differences, we performed a Nemenyi test. See Fig.5 for the results.

From Fig. 5 we observe that

1) PLAC is better-ranked than three out of five single learning methods, and performs significantly faster than IB1. However, PLAC is on the right side of the line associated with OneR, which means it is significantly outperformed by OneR in terms of runtime. The reason is that OneR is a very simple

method, its time complexity is *O(m)*, thus is much efficient. Unfortunately, its accuracy is just 82.09% of PLAC with OneR as the base learner.

2) Ensemble learning methods Bagging and Boosting are on the right side of the line associated with PLAC when each of the five single learning method is used as the base learner. This discloses that the runtime improvements of PLAC for these two ensemble learning methods have statistical significance.
3) Lazy learning method L_{AZY}DT is on the right side of the line associated with PLAC when four out of five single learning methods are employed as the base learner. This means PLAC performs significantly faster than L_{AZY}DT except when SVM is chosen as the base learner. The underlying reason is that although SVM is only used to build the classifier on a very small part of the training data, it is still time consuming due to its high time complexity ($O(n3)$). Even so, its average runtime is decreased by PLAC by 94.67%.

5.5.4. Performance summary

In order to provide a holistic view on the performance of classification methods used in this paper, for each single learning method, we ranked it with L_{AZY}DT, and Bagging, Boosting and PLAC with this single method as the base learner according to classification accuracy and runtime, respectively. Then the ranks of accuracy and runtime were summarized, and the final ranks were computed in accordance with the summation. See Table 12 for the results. From it we observe that, PLAC ranks 1 in each case, thus should be an undisputed first choice.

Table 12. Performance Ranks of Classification Methods over 40 Data Sets

			NB+		
	LDT	NB	Bag	Bst	PLAC
Accuracy	2	5	4	3	1
Runtime	4	1	3	5	2
Sum	6	6	7	8	3
Rank	2	2	4	5	1
			C4.5+		
	LDT	C4.5	Bag	Bst	PLAC
Accuracy	5	4	1	3	2
Runtime	3	2	4	5	2
Sum	8	6	5	8	3
Rank	4	3	2	4	1
			SVM+		
	LDT	SVM	Bag	Bst	PLAC
Accuracy	2	5	3	4	1
Runtime	1	3	4	5	2
Sum	3	8	7	9	3
Rank	1	4	3	5	1
			IB1+		
	LDT	IB1	Bag	Bst	PLAC
Accuracy	3	4	2	5	1
Runtime	3	2	4	5	1
Sum	6	6	6	10	2
Rank	2	2	2	5	1
			OneR+		
	LDT	OneR	Bag	Bst	PLAC
Accuracy	1	4	3	5	2
Runtime	5	1	3	4	2
Sum	6	5	6	9	4
Rank	3	2	3	5	1

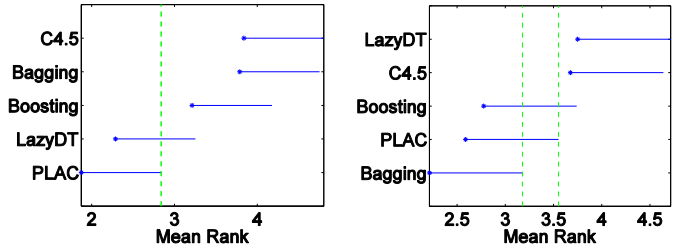
Note: LDT, Bag, and Bst denote LAZYDT, Bagging, and Boosting, respectively.

5.6. Sensitivity Analysis

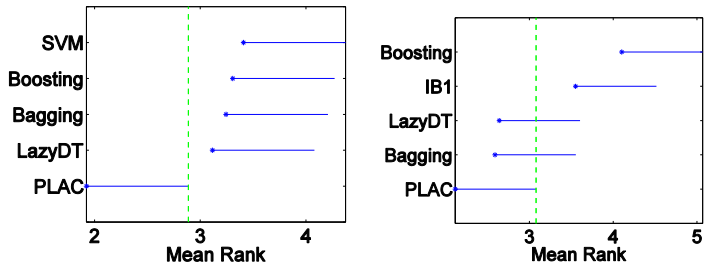
The minimum size threshold *partitionSize* of a partition is the only parameter of PLAC. It should be predefined and its value definitely affects the classification performance of PLAC. Thus, we analyzed its effect by enforcing a wide range of different size values, i.e., 1/2, 1/3, 1/4, ..., 1/30 of a data set size, on all the 40 evaluated data sets. Fig. 6 shows the accuracy variation of PLAC with five different types of classification algorithms NB, OneR, C4.5, IB1, and SVM as the base learner with respect to *partitionSize* for the 40 real world data sets.

From Fig. 6 we observe that

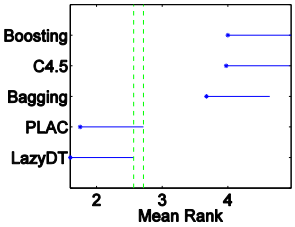
1) For PLAC with SVM, NB, and IB1 as base leaner, each of the accuracy curves starts from a bigger point, and climbs to the peak with fluctuation, then decreases much slowly. Moreover, when *partitionSize* is 0.054 times the data set size, all the three curves achieve their peak points. This means if classification algorithms SVM, NB, and IB1 are chosen as the base learner, *partitionSize* should be predefined as 0.054 times of the number of instances for a classification problem.
2) For PLAC with C4.5 as its base learner, the accuracy curve gradually increases from a small value, then quickly decreases after reaching the peak point when *partitionSize* is 0.44 times the data set size. Clearly, 0.44 times of the number of instances should be predefined as the value of *partitionSize*.
3) For PLAC with OneR as the base learner, the accuracy curve starts from the peak point, then falls sharply to a much smaller point at the end. This reveals that *partitionSize* should be predefined as 0.03 times of the number of instances for the problems to be solved.



(a)LAZYDT, NB/NB as the base learner (b)LAZYDT, C4.5/C4.5 as the base learner

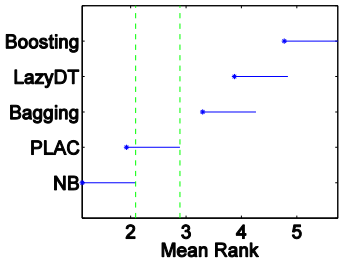


(c) LAZYDT, C4.5/C4.5 as the base learner (d) LAZYDT, IB1/IB1 as the base learner

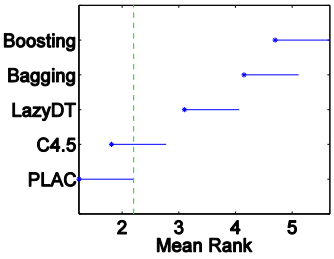


(e) LAZYDT, OneR/OneR as the base learner

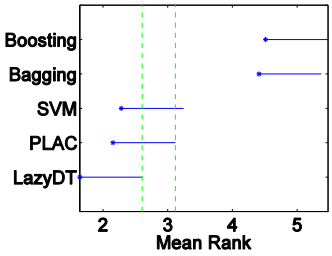
Fig. 4. Pairwise accuracy comparisons of the classification methods using Nemenyi's post hoc test with α = *0.05.*



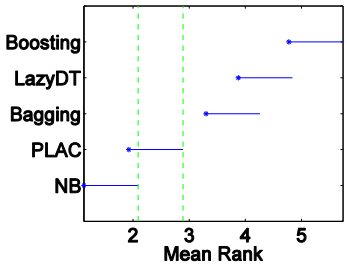
(a) LAZYDT, NB/NB as the base learner



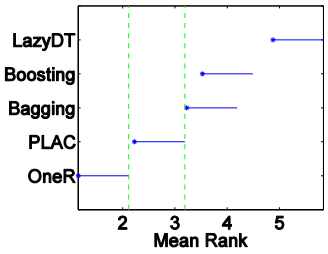
(b)LAZYDT, C4.5/C4.5 as the base learner



(b) LAZYDT, C4.5/C4.5 as the base learner



(d) LAZYDT, IB1/IB1 as the base learner



(e) LAZYDT, OneR/OneR as the base learner

Fig. 5. Pairwise accuracy comparisons of the classification methods using Nemenyi's post hoc test with $\alpha = 0.05$.

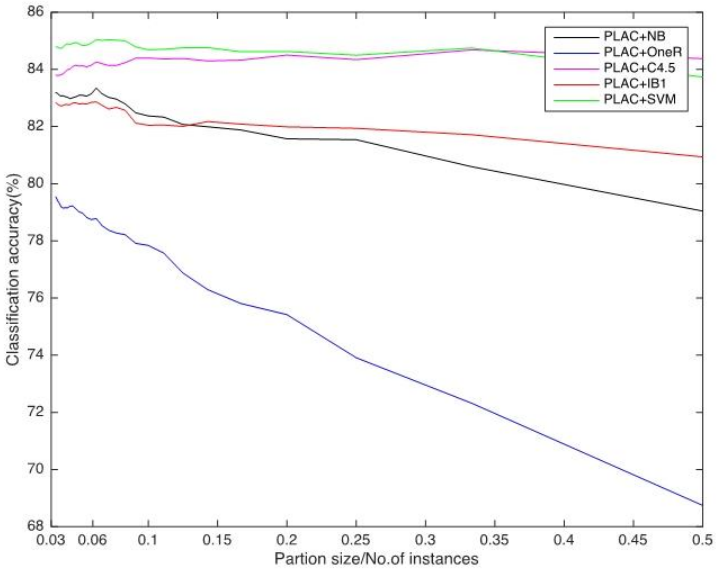


Fig. 6. Accuracy when varying the minimum size of a partition.

6. Conclusion

In this paper, we have presented a novel lazy learning method named PLAC. PLAC transforms a complicated problem into multiple smaller and easier-to-learn partitions with an effective data partitioning approach, then upon receiving a new instance, its nearest partition is located and employed to build a classifier that is finally used to classify the new instance. In PLAC, the built classifier is customized with the particular characteristics of the new instance, and only one of the multiple partitions is used, thus classification performance is increased. Moreover, since the caching mechanism is designed to avoid work replication during classification, efficiency is further improved.

We have compared the performance of PLAC with those of three different types of well-known classification methods including eager learning methods SVM, OneR, NB, and C4.5, lazy learning methods IB1 and LAZYDT, and ensemble learning methods Bagging and Boosting on 40 publicly available real world data from two different aspects of the classification accuracy and runtime. Generally, PLAC performs better than the other methods.

We have also conducted the performance validation through the nonparametric Friedman test followed by the Nemenyi post-hoc test. By combining the results of accuracy validation and runtime validation, we found that PLAC ranks best out of all the comparative methods.

Acknowledgment

This study is partially supported by National Natural Science Foundation of China under Grant 61370144.

References

[1] Quinlan, J. R. (2014). *C4.5*: *Programs for Machine Learning*. Elsevier.

[2] Breiman, L., Friedman, J., Olshen, R., Stone, C., Steinberg, D., & Colla, P. (1983). *Cart: Classification and Regression Trees*. Wadsworth: Belmont, C.

[3] Friedman, J. H., Kohavi, R., & Yun, Y. (1996). Lazy decision trees. *Proceedings of the National Conference on Artificial Intelligence*.

[4] Cohen, W. W. (1995). Fast effective rule induction. *Proceedings of the 12th International Conference on Machine Learning*.

[5] Frank, E., & Witten, I. H. (1998). Generating accurate rule sets without global optimization. *Proceedings of the 15th International Conference on Machine Learning*.

[6] Holte, R. C. (1993). Very simple classification rules perform well on most commonly used datasets. *Machine learning*, *11(1)*, 63–90.

[7] Cover, T., & Hart, P. (1967). Nearest neighbor pattern classification. *IEEE Transactions on Information Theory*, *13*, 21–27.

[8] Aha, D. W., Kibler, D., & Albert, M. K. (1991). Instance-based learning algorithms. *Machine Learning*, *6(1)*, 37–66.

[9] John, G. H., & Langley, P. (1995). Estimating continuous distributions in bayesian classifiers. *Proceedings of the Eleventh Conference on Uncertainty in Artificial Intelligence*.

[10] Platt, J. (1999). Using analytic qp and sparseness to speed training of support vector machines. *Proceedings of the Advances in Neural Information Processing Systems*.

[11] Atkeson, C., Moore, A., & Schaal, S. (1997). Locally weighted learning for control. *Lazy Learning*.

[12] Garcia, E. K., Feldman, S., Gupta, M. R., & Srivastava, S. (2010). Completely lazy learning. *Knowledge and Data Engineering*.

[13] Veloso, A., Meira,W., & Zaki, M. J. (2006). Lazy associative classification. *Data Mining*.

[14] Zhu, X., & Yang, Y. (2008). A lazy bagging approach to classification. *Pattern Recognition*, *41(10)*, 2980–

2992.
[15] Sharkey, A. J., & Sharkey, N. E. (1997). Combining diverse neural nets. *The Knowledge Engineering Review*, *12(3)*, 231–247.
[16] Dietterich, T. G. (2002). Ensemble learning. *The Handbook of Brain Theory and Neural Networks*.
[17] Breiman, L. (1996). Bagging predictors. *Machine Learning*, *24(2)*.
[18] Freund, Y. (1995). Boosting a weak learning algorithm by majority. *Information and Computation*.
[19] Ho, T. K. (1998). The random subspace method for constructing decision forests. *Pattern Analysis and Machine Intelligence*.
[20] Bryll, R., Gutierrez-Osuna, R., & Quek, F. (2003). Attribute bagging: Improving accuracy of classifier ensembles by using random feature subsets. *Pattern Recognition*.
[21] Zheng, X., Shen, J., Cox, C., Wakefield, J. C., Ehm, M. G., Nelson, M. R., & Weir, B. S. (2014). Hibagâ ăťhla genotype imputation with attribute bagging. *The Pharmacogenomics Journal*, *14(2)*, 192–200.
[22] Cherkauer, K. J. (1996). Human expert-level performance on a scientific image analysis task by a system using combined artificial neural networks. *Working Notes of the AAAI Workshop on Integrating Multiple Learned Models*.
[23] Breiman, L. (2001). Random forests. *Machine learning*, *45(1)*, 5–32.
[24] Zhou, Z.-H. (2012). *Ensemble Methods: Foundations and Algorithms*.
[25] Lichman, M. (2013). *UCI Machine Learning Repository*.
[26] Webb, G. I. (2000). Multiboosting: A technique for combining boosting and wagging. *Machine Learning*.
[27] S. C. (1994). A conservation law for generalization performance. *Proceedings of the 11th International Conference on Machine Learning*.
[28] Reyzin, L., & Schapire, R. E. (2006). How boosting the margin can also boost classifier complexity. *Proceedings of the 23rd International Conference on Machine Learning*.
[29] Friedman, M. (1940). A comparison of alternative tests of significance for the problem of m rankings. *The Annals of Mathematical Statistics*, *11(1),* 86–92.
[30] Nemenyi, P. (1962). Distribution-free multiple comparisons.
[31] Newman, D. (1939). The distribution of range in samples from a normal population, expressed in terms of an independent estimate of standard deviation. *Biometrika*, 20–30.
[32] Demˇsar, J. (2006). Statistical comparisons of classifiers over multiple data sets. *The Journal of Machine Learning Research*.
[33] GarcÃŋa, S., Herrera, F., & Shawe-taylor, J. (2008). An extension on statistical comparisons of classifiers over multiple data sets" for all pairwise comparisons. *Journal of Machine Learning Research*.

Wei Song received her B.A. in computer science from UC Berkeley in Dec 2016. She is currently a software engineer at Microsoft working on the Azure Cloud. Before Microsoft, she was a software development engineer at Amazon working on the Amazon Marketplace Seller Platform. Aside from her work in the software engineering industry, her research is intended towards machine learning and cloud security.

He Jiang is an awardee of the NSFC Excellent Young Scholars Program in 2017. He is currently a professor with Dalian University of Technology and an adjunct professor with Beijing Institute of Technology. His current research interests include search-based software engineering and mining software repositories. He has published over 60 referred papers on journals and international conferences, including IEEE Trans.

Software Engineering, IEEE Trans. Knowledge and Data Engineering, ICSE, SANER, etc., supported by the Program for New Century Excellent Talents in University and the National Science Fund for Excellent Young Scholars. In addition, he serves as the guest editors of some journals and magazines, including IEEE Computational Intelligence, Journal of Computer Science and Technology, Frontiers of Computer Science, etc.

Fan Ma is a first-year Phd candidate at Centre for Artificial Intelligence, University of Technology Sydney (UTS). He received the BE and ME degree from Xi'an Jiaotong University in 2014 and 2017 respectively. His research interests include machine learning and computer vision, especially on semi-supervised learning, self-paced learning and video analysis.

Qinbao Song received his PhD degree in computer science from Xi'an Jiaotong University in Xi'an, China, in 2001. He is a professor in software technology at the Department of Computer Science and Technology, Xi'an Jiaotong University. He is also an adjunct professor in the State Key Laboratory of Software Engineering, Wuhan University. He has authored and coauthored more than 100 referred papers in the areas of machine learning and software engineering. In addition, he serves as a board member of the Open Software Engineering Journal. His research interests include data mining/machine learning, empirical software engineering, and trustworthy software.

Guangtao Wang received the PhD degree in computer science from Xi'an Jiaotong University, Xi'an, China, in 2013. He is currently an assistant professor in the Department of Computer Science and Technology, Xian Jiaotong University. His research focuses on data mining and machine learning.