Heterogeneous AdaBoost with Stochastic Algorithm Selection

Kuo-Wei Hsu
Department of Computer Science, National Chengchi University
No. 64, Sec. 2, Zhi Nan Rd., Wen Shan District
Taipei City 11605, Taiwan (R.O.C.)
+886-2-29393091 ext. 62297
kwhsu@nccu.edu.tw

ABSTRACT
We propose an ensemble learning algorithm based on AdaBoost and employing heterogeneous algorithms with a stochastic process for algorithm selection. Diversity is an important factor in ensemble learning and AdaBoost creates diversity by manipulating training data sets. However, we observe that AdaBoost generates training data sets of low diversity in later iterations. Some researchers suggest the employment of heterogeneous algorithms in ensemble learning to achieve better diversity. Following the idea, we extend AdaBoost and propose an algorithm that employs different base learning algorithms in different iterations. The most distinguishing feature of our algorithm is that it selects algorithms using a stochastic process where their earlier performance is considered. The results from experiments on several data sets show the utility of our algorithm: It could outperform AdaBoost on 22 to 33, depending on settings, out of 40 data sets considered in experiments.

CCS Concepts
• **Computing methodologies→Machine learning** • *Computing methodologies→Artificial intelligence* • *Information systems→Mobile information processing systems* • Information systems→Data mining • *Human-centered computing→Ubiquitous and mobile computing*

Keywords
Boosting; Classification; Ensemble

1. INTRODUCTION
Ensemble learning is to create multiple models and combine results from models, and it has attracted much research attention [4, 22, 26-29, 38]. If the center of attention is classification, ensemble learning could be understood as follows: It constitutes a group of classifiers first and then aggregates classification results or predictions from these member classifiers. For an unseen sample, it inputs the sample to member classifiers, collects predictions from member classifiers, and outputs the final prediction based on some fusion methods. Examples of fusion methods include unweighted or weighted majority voting.

IMCOM'17, January 05–07, 2017, Beppu, Japan.
© 2017 ACM. ISBN 978-1-4503-4888-1/17/01…$15.00
DOI: http://dx.doi.org/10.1145/3022227.3022266

The intuition behind ensemble learning is analogous to the process where we consider opinions from all possible sources and accordingly make a final decision. From this point of view, ensemble learning works like a committee where members provide opinions from their own perspectives. We expect that these opinions are different from and independent of each other. Therefore, classifier quality and diversity are among factors that contribute to the successful results of ensemble learning. The rationale behind the first factor (i.e. quality) is that a classifier with better prediction performance contributes more to the group. The ground for the second factor (diversity) is simply that one hundred identical classifiers are functionally equivalent to one.

Researchers have studied classification for decades and they have proposed various off-the-shelf classification algorithms, so classifier quality is less important than diversity in ensemble learning. Diversity, including diversity among classifiers in an ensemble and diversity creation for an ensemble, has gained considerable attention. Examples of methods to create diversity include those manipulating training sets, such as bagging [1], boosting [8, 37], and ideas presented in papers [23-24]. An example for feature set manipulation is random forests [2].

AdaBoost (Adaptive Boosting) is one of the most popular ensemble learning algorithms [8, 37]. It has been applied to various applications such as natural language processing [5-6], network security [12], healthcare [32], computer vision [7, 14, 19, 21], wearable computing [30], and intelligent transportation systems [31]. Ensemble learning requires diversity. AdaBoost creates diversity by manipulating training sets, and it operates on training sets composed of hard-to-classify samples as the process proceeds. However, we observe that diversity of training sets in later iterations is low, and this could potentially impair the prediction performance of AdaBoost. Below is the problem statement of this paper:

How to improve the prediction performance of AdaBoost?

If we are given a limited set of training samples, we need to run AdaBoost with a large number of iterations. Nevertheless, we are given limited time for most tasks in practice. Therefore, we put our focus on situations where training samples and our time are limited. In contrast, if the given training set is sufficiently large, we do not need to run AdaBoost with a larger number of iterations. Nevertheless, this is not always the case especially in situations where having more training samples are costly.

Usually a shrinkage coefficient is used to make AdaBoost run the exponential weight updating procedure, which is adopted to generate training sets, in a more gentle way. This is one of regularization techniques widely used in statistics. The impact could be seen when we perform a limited number of iterations for AdaBoost, which is generally the case. It could also be seen when there is a limited set of training samples.

We could employ heterogeneous algorithms in ensemble learning to achieve better diversity, as suggested in papers [10-11]. We extend AdaBoost and propose an algorithm that harnesses heterogeneity for encouraging diversity. We employ different algorithms and adopt a stochastic process for algorithm selection.

The rest of this paper is organized as follows. We will review diversity and AdaBoost in Section 2, and we will describe our algorithm in Section 3. There will be experiment results presented in Section 4. Finally, conclusion will be given in Section 5.

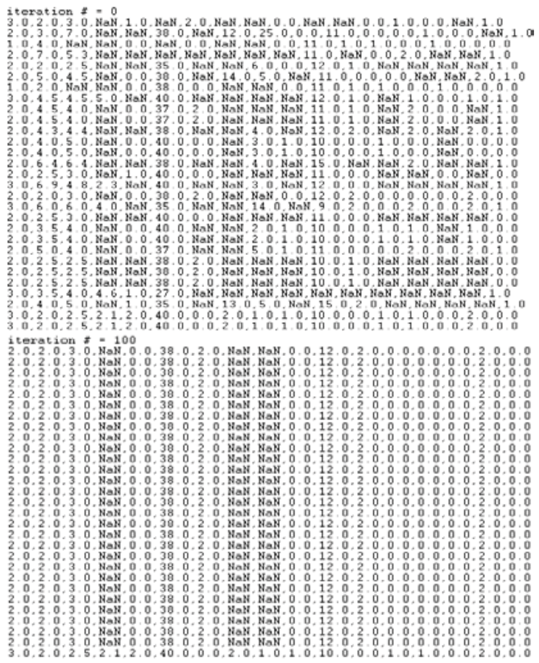
2. DIVERSITY AND ADABOOST

Diversity measures are proposed to assist in the selection of algorithms for member classifiers and/or the evaluation of fusion methodologies (or combination strategies) in ensemble learning [15, 18]. The study of diversity in ensemble learning has gained increasing attention, and examples of papers include [3, 15-17, 25, 29, 33-34]. Nevertheless, most papers studying diversity do not take heterogeneity into account. Here, heterogeneity is from using different or heterogeneous algorithms. Papers [10-11] investigate the relationship between heterogeneity and diversity. Experiment results in papers [10-11] have shown the impact of heterogeneity on diversity. Using two different algorithms in ensemble learning would achieve better diversity.

AdaBoost works as follows: Initially it assigns equal weights to samples and then draws samples with replacement to generate the first training set. Afterward, it employs the training set to create the first classifier. Next, it evaluates the performance of the classifier and weights of samples misclassified by the classifier will be increased exponentially. Then, it draws samples based on probabilities proportional to their weights in order to generate the second training set that will be used to create the second classifier. Next, it repeats doing weighted sampling, creating classifiers, evaluating classifiers, and updating weights of samples until it has reached a certain number of iterations. In other words, AdaBoost iteratively creates a group of classifiers and manipulates training sets by performing weighted sampling where the probability of a sample being drawn is exponentially proportional to the probability that it was misclassified.

From the process, we could see that AdaBoost assigns more weights to samples for which most classifiers (created so far) made incorrect predictions. AdaBoost utilizes oversampling to force the base learning algorithm to create classifiers focusing on those hard-to-classify samples. This feature distinguishes AdaBoost from other methods manipulating training sets. It has the potential to achieve better prediction performance because it considers all samples based on their difficulties. However, this feature brings the risk of being overfitting. Nevertheless, if the number of iterations is large enough or if the number of training samples is large enough, AdaBoost does not tend to be overfitting. However, the number of iterations is usually limited in practice.

Let us consider the following example, where we apply AdaBoost with C4.5 decision tree [28, 37] on a small data set (which is named *labor* coming with 57 samples and 17 features). We do 50-50 random split: One half of data is used as training samples, while the other half is used to test the created classification model (an ensemble, for example) built upon those training samples. These two are disjoint subsets of the data set. Examples of training sets are in Figure 1.


(a)

```
iteration # = 0
3.0,2.0,3.0,NaN,1.0,NaN,2.0,NaN,NaN,0.0,NaN,NaN,0.0,1.0,0.0,NaN,1.0
2.0,3.0,7.0,NaN,NaN,38.0,NaN,12.0,25.0,0.0,11.0,0.0,0.0,1.0,0.0,NaN,1.0
1.0,4.0,NaN,NaN,0.0,NaN,0.0,NaN,NaN,0.0,11.0,1.0,1.0,0.0,1.0,0.0,0.0
2.0,7.0,5.3,NaN,NaN,NaN,NaN,NaN,NaN,NaN,11.0,NaN,0.0,2.0,NaN,NaN,1.0
2.0,2.0,2.5,NaN,NaN,35.0,NaN,NaN,6.0,0.0,12.0,1.0,NaN,NaN,NaN,NaN,1.0
2.0,5.0,4.5,NaN,0.0,38.0,NaN,14.0,5.0,NaN,11.0,0.0,0.0,NaN,NaN,2.0,1.0
1.0,2.0,NaN,NaN,0.0,38.0,0.0,NaN,NaN,0.0,11.0,1.0,1.0,0.0,1.0,0.0,0.0
3.0,4.5,4.5,5.0,NaN,40.0,NaN,NaN,NaN,NaN,12.0,1.0,NaN,1.0,0.0,1.0,1.0
2.0,4.5,4.0,NaN,0.0,37.0,2.0,NaN,NaN,NaN,11.0,1.0,NaN,2.0,0.0,NaN,1.0
2.0,4.5,4.0,NaN,0.0,37.0,2.0,NaN,NaN,NaN,11.0,1.0,NaN,2.0,0.0,NaN,1.0
2.0,4.3,4.4,NaN,NaN,38.0,NaN,NaN,4.0,NaN,12.0,2.0,NaN,2.0,NaN,2.0,1.0
2.0,4.0,5.0,NaN,0.0,40.0,0.0,NaN,3.0,1.0,10.0,0.0,1.0,0.0,NaN,0.0,0.0
2.0,4.0,5.0,NaN,0.0,40.0,0.0,NaN,3.0,1.0,10.0,0.0,1.0,0.0,NaN,0.0,0.0
2.0,6.4,6.4,NaN,NaN,38.0,NaN,NaN,4.0,NaN,15.0,NaN,NaN,2.0,NaN,NaN,1.0
2.0,2.5,3.0,NaN,1.0,40.0,0.0,NaN,NaN,NaN,11.0,0.0,NaN,NaN,0.0,NaN,0.0
3.0,6.9,4.8,2.3,NaN,40.0,NaN,NaN,3.0,NaN,12.0,0.0,NaN,NaN,NaN,NaN,1.0
2.0,2.0,3.0,NaN,0.0,38.0,2.0,NaN,NaN,0.0,12.0,2.0,0.0,0.0,0.0,2.0,0.0
3.0,6.0,6.0,4.0,NaN,35.0,NaN,NaN,14.0,NaN,9.0,2.0,0.0,2.0,0.0,2.0,1.0
2.0,2.5,3.0,NaN,NaN,40.0,0.0,NaN,NaN,NaN,11.0,0.0,NaN,NaN,NaN,NaN,0.0
2.0,3.5,4.0,NaN,0.0,40.0,NaN,NaN,2.0,1.0,10.0,0.0,1.0,1.0,NaN,1.0,0.0
2.0,3.5,4.0,NaN,0.0,40.0,NaN,NaN,2.0,1.0,10.0,0.0,1.0,1.0,NaN,1.0,0.0
2.0,5.0,4.0,NaN,0.0,37.0,NaN,NaN,5.0,1.0,11.0,0.0,0.0,2.0,0.0,2.0,1.0
2.0,2.5,2.5,NaN,NaN,38.0,2.0,NaN,NaN,NaN,10.0,1.0,NaN,NaN,NaN,NaN,0.0
2.0,2.5,2.5,NaN,NaN,38.0,2.0,NaN,NaN,NaN,10.0,1.0,NaN,NaN,NaN,NaN,0.0
2.0,2.5,2.5,NaN,NaN,38.0,2.0,NaN,NaN,NaN,10.0,1.0,NaN,NaN,NaN,NaN,0.0
3.0,3.5,4.0,4.6,1.0,27.0,NaN,NaN,NaN,NaN,NaN,NaN,NaN,NaN,NaN,NaN,1.0
2.0,4.0,5.0,NaN,1.0,35.0,NaN,13.0,5.0,NaN,15.0,2.0,NaN,NaN,NaN,NaN,1.0
3.0,2.0,2.5,2.1,2.0,40.0,0.0,2.0,1.0,1.0,10.0,0.0,1.0,1.0,0.0,2.0,0.0
3.0,2.0,2.5,2.1,2.0,40.0,0.0,2.0,1.0,1.0,10.0,0.0,1.0,1.0,0.0,2.0,0.0
```

(b)

```
iteration # = 100
2.0,2.0,3.0,NaN,0.0,38.0,2.0,NaN,NaN,0.0,12.0,2.0,0.0,0.0,0.0,2.0,0.0
2.0,2.0,3.0,NaN,0.0,38.0,2.0,NaN,NaN,0.0,12.0,2.0,0.0,0.0,0.0,2.0,0.0
2.0,2.0,3.0,NaN,0.0,38.0,2.0,NaN,NaN,0.0,12.0,2.0,0.0,0.0,0.0,2.0,0.0
2.0,2.0,3.0,NaN,0.0,38.0,2.0,NaN,NaN,0.0,12.0,2.0,0.0,0.0,0.0,2.0,0.0
2.0,2.0,3.0,NaN,0.0,38.0,2.0,NaN,NaN,0.0,12.0,2.0,0.0,0.0,0.0,2.0,0.0
2.0,2.0,3.0,NaN,0.0,38.0,2.0,NaN,NaN,0.0,12.0,2.0,0.0,0.0,0.0,2.0,0.0
2.0,2.0,3.0,NaN,0.0,38.0,2.0,NaN,NaN,0.0,12.0,2.0,0.0,0.0,0.0,2.0,0.0
2.0,2.0,3.0,NaN,0.0,38.0,2.0,NaN,NaN,0.0,12.0,2.0,0.0,0.0,0.0,2.0,0.0
2.0,2.0,3.0,NaN,0.0,38.0,2.0,NaN,NaN,0.0,12.0,2.0,0.0,0.0,0.0,2.0,0.0
2.0,2.0,3.0,NaN,0.0,38.0,2.0,NaN,NaN,0.0,12.0,2.0,0.0,0.0,0.0,2.0,0.0
2.0,2.0,3.0,NaN,0.0,38.0,2.0,NaN,NaN,0.0,12.0,2.0,0.0,0.0,0.0,2.0,0.0
2.0,2.0,3.0,NaN,0.0,38.0,2.0,NaN,NaN,0.0,12.0,2.0,0.0,0.0,0.0,2.0,0.0
2.0,2.0,3.0,NaN,0.0,38.0,2.0,NaN,NaN,0.0,12.0,2.0,0.0,0.0,0.0,2.0,0.0
2.0,2.0,3.0,NaN,0.0,38.0,2.0,NaN,NaN,0.0,12.0,2.0,0.0,0.0,0.0,2.0,0.0
2.0,2.0,3.0,NaN,0.0,38.0,2.0,NaN,NaN,0.0,12.0,2.0,0.0,0.0,0.0,2.0,0.0
2.0,2.0,3.0,NaN,0.0,38.0,2.0,NaN,NaN,0.0,12.0,2.0,0.0,0.0,0.0,2.0,0.0
2.0,2.0,3.0,NaN,0.0,38.0,2.0,NaN,NaN,0.0,12.0,2.0,0.0,0.0,0.0,2.0,0.0
2.0,2.0,3.0,NaN,0.0,38.0,2.0,NaN,NaN,0.0,12.0,2.0,0.0,0.0,0.0,2.0,0.0
2.0,2.0,3.0,NaN,0.0,38.0,2.0,NaN,NaN,0.0,12.0,2.0,0.0,0.0,0.0,2.0,0.0
2.0,2.0,3.0,NaN,0.0,38.0,2.0,NaN,NaN,0.0,12.0,2.0,0.0,0.0,0.0,2.0,0.0
2.0,2.0,3.0,NaN,0.0,38.0,2.0,NaN,NaN,0.0,12.0,2.0,0.0,0.0,0.0,2.0,0.0
2.0,2.0,3.0,NaN,0.0,38.0,2.0,NaN,NaN,0.0,12.0,2.0,0.0,0.0,0.0,2.0,0.0
2.0,2.0,3.0,NaN,0.0,38.0,2.0,NaN,NaN,0.0,12.0,2.0,0.0,0.0,0.0,2.0,0.0
2.0,2.0,3.0,NaN,0.0,38.0,2.0,NaN,NaN,0.0,12.0,2.0,0.0,0.0,0.0,2.0,0.0
2.0,2.0,3.0,NaN,0.0,38.0,2.0,NaN,NaN,0.0,12.0,2.0,0.0,0.0,0.0,2.0,0.0
2.0,2.0,3.0,NaN,0.0,38.0,2.0,NaN,NaN,0.0,12.0,2.0,0.0,0.0,0.0,2.0,0.0
2.0,2.0,3.0,NaN,0.0,38.0,2.0,NaN,NaN,0.0,12.0,2.0,0.0,0.0,0.0,2.0,0.0
2.0,2.0,3.0,NaN,0.0,38.0,2.0,NaN,NaN,0.0,12.0,2.0,0.0,0.0,0.0,2.0,0.0
3.0,2.0,2.5,2.1,2.0,40.0,0.0,2.0,1.0,1.0,10.0,0.0,1.0,1.0,0.0,2.0,0.0
```

Figure 1. Example: Training samples in the initial iteration (a) and later one (b)

Results reported here are averaged over 50 runs, while training and test samples are generated randomly in every run. In Figure 1, there are data matrices, in each of which a row represents a sample and a column represents a feature. Figure 1 (a) and (b) present training samples used in the initial iteration and a later one, respectively. It is clear from Figure 1 that variety of training sets in the initial iteration is higher than that in a later iteration. Samples in a later iteration are duplicates of hard-to-classify ones.

Figure 1 visualizes data matrices of training sets while Figure 2 provides a numerical measure of their complexity. Figure 2 presents the number of iterations versus the average number of principal components in training sets. At first, there are about 13 principal components in the initial training set. This means that we need a 13-dimensional space to cover 95% variance among training samples that originally exist in a 17-dimensional space. Later, there are about 4 principal components in a training set. This means that we only need a 4-dimensional space to cover 95% variance among training samples used in a later iteration.

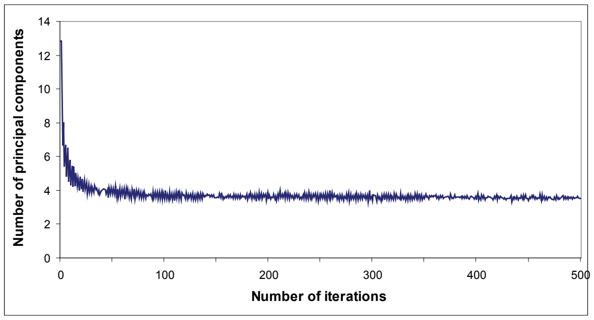


Figure 2. Number of principal components vs. iterations

Figure 2 presents complexity of training sets while Figure 3 presents complexity of created classification models. Figure 3 shows the number of iterations versus the average number of nodes in created decision trees. As a training set becomes *monotonic* in variety, the classification model (a decision tree in this example) also becomes *monotonic*.

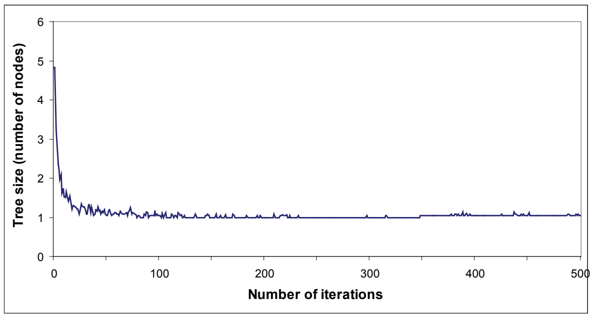


Figure 3. Tree size vs. iterations

Figure 4 presents the average training and test error rates versus the number of iterations. Here the training error rate is a cumulative one and is aggregated with respect to all classifiers created so far. The test error rate is with respect to an isolated test set and it is the one given by the combination of classifiers created so far. AdaBoost significantly lowers both training and test error rates in less than 50 iterations. This demonstrates an advantage of AdaBoost and ensemble learning as well.

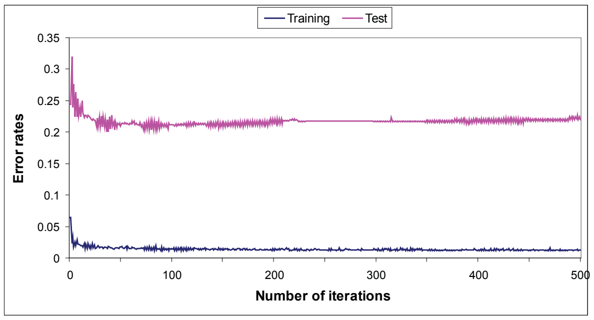


Figure 4. Training and test error rates vs. iterations

AdaBoost generates training sets composed of hard-to-classify samples as it iterates, such that the data distribution of a training set in a later iteration could be fairly different from the one in the original data set. This feature contributes to the generalization capability of AdaBoost because it motivates AdaBoost to examine different aspects of the underlying distribution. Nevertheless, this happens when the number of iterations is large enough, and usually it is not practical to perform a large number of iterations.

The introduction of a shrinkage coefficient, v, is a widely-used technique that allows AdaBoost to generate training sets in a more gentle way. Using v is to postpone the decline of variety of training sets (e.g., the sharp drops in Figures 2 and 3) such that diversity from difference among training sets could be preserved. It helps AdaBoost avoid overfitting because v discourages AdaBoost to over-emphasize hard-to-classify samples (especially in a limited number of iterations). Usually it is a small positive constant ($0 < v < 1$). When weights of samples are updated in smaller steps (or changes of weights are smaller) and when the number of updates is limited, the weighted sampling procedure works as an unweighted sampling procedure. In other words, v allows AdaBoost to gently generate training sets and accordingly maintains diversity from training sets.

We introduce the use of different or heterogeneous algorithms in AdaBoost. Our algorithm establishes another source of diversity, diversity from heterogeneity between algorithms. The purpose of using heterogeneous algorithms is two-fold: First, it adds another source of diversity when diversity from difference among training sets is still sufficient to make a major contribution (in earlier iterations). Second, it serves as the primary source of diversity when variety of training sets declines and they could only make a minor contribution to diversity (in later iterations).

3. THE PROPOSED ALGORITHM

Our algorithm follows the basic procedure of AdaBoost: Sampling with weights, training, evaluating classifiers, updating weights according to earlier prediction performance, and adopting weighted majority voting to reach final predictions. Our algorithm is different from AdaBoost in that, it creates classifiers with heterogeneous algorithms in a stochastic way. In some iteration, our algorithm might not only focus on samples that are predicted incorrectly in the last iteration but it might also use an algorithm different from the one used in the last iteration. It uses a stochastic process to select algorithms from *a bag of algorithms*, or *BA*.

The process behaves in a non-deterministic way and the algorithm selected for the next iteration is determined by both the earlier prediction performance of the algorithm and a random number. The design of the random element in this stochastic process is presented below. Figure 5 illustrates an example, where there are two base learning algorithms, A_1 and A_2, and P_1 represents the transition probability for A_1. A_1 is the one used in the last iteration, and P_1 determines which will be used in the current iteration. The transition probability is the probability that the process transits from A_1 to A_2. It is the probability that the process does not select A_1 but the A_2. In other words, 1-P_1 is the probability that the process selects A_1 again. In our algorithm, 1-P_1 is proportional to the weight of A_1. Therefore, there is not only a set of weights of samples but also a set of weights of algorithms in our algorithm.

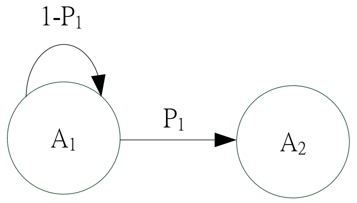


Figure 5. Stochastic algorithm selection for two algorithms

Our algorithm is presented in Figure 6 and described as follows. One of its input parameters is a data set S of size n. Another input parameter is BA, a set of algorithms that will be considered as base learning algorithms. The size of BA (denoted by $|BA|$) is m. In our experiments, BA contains decision stump algorithm (which is stable but less powerful) and decision tree algorithm generating unpruned trees (which is more powerful but unstable). Such a combination of algorithms is interesting because decision stump algorithm tends to be underfitting while the decision tree algorithm (generating unpruned trees) tends to be overfitting. Furthermore, the other input parameter of our algorithm is the total number of iterations, T. The first two steps are to initialize variables. The third step is basically a loop. In the t-th iteration, it does the following: It uses weighted sampling to have a training set, S_t. It uses weighted sampling to have an algorithm, A_t. It uses A_t and S_t to create a classifier C_t. Then, it evaluates the prediction performance of C_t. It updates weights of samples and weights of algorithms according to the prediction performance of C_t.. In Figure 6, I is the indicator function. In the end, the algorithm will create a set of T classifiers and use a weighted majority voting to generate final predictions. The fourth step is to use the prediction performance of each classifier in the ensemble as a weight and calculate the probability that a sample belongs to a class, and it is to select one among all classes with the largest probability.

```
1   Initialize weights of samples, $ws_i = 1/n$, $WS = \{ws_i\}$,
    where $n$ is the number of samples ($n = |S| = |WS|$)
2   Initialize weights of algorithms, $wa_j = 1/m$, $WA = \{wa_j\}$, where $m$ is the number of algorithms ($m = |WA| = |BA|$)
3   For iteration $t = 1$ to $T$, do
    3.1  Create a training set of size $n$:
         $S_t = ResampleWithWeights(S, WS, n)$
    3.2  Select an algorithm:
         $A_t = ResampleWithWeights(BA, WA, 1)$
    3.3  Create a classifier, $C_t = \mathbf{C}(A_t, S_t)$, based on the
         selected algorithm $A_t$ and $S_t$, where $\mathbf{C}$ could be
         a function call to create an instance of classifier
    3.4  Calculate the training error:
         $\varepsilon_t = \frac{1}{n}\sum_{i=1}^{n} ws_i \cdot \mathrm{I}\left[y_i \neq \hat{y}_i^{C_t} = \hat{y}_i^{(A_t, S_t)}\right]$
    3.5  Calculate $\alpha_t$ (presenting quality of $C_t$):
         $\alpha_t = \frac{1}{2}\log\left(\frac{1-\varepsilon_t}{\varepsilon_t}\right)$
    3.6  Update weights of samples:
         $ws_i^{new} = ws_i \cdot e^{v \cdot \alpha_t \cdot \mathrm{I}\left[y_i \neq \hat{y}_i^{C_t}\right]}$
    3.7  Normalize weights of samples and set the
         updated weights for the next iteration:
         $ws_i = ws_i^{new} \cdot \left(\sum_{i=1}^{n} ws_i^{new}\right)^{-1}$
    3.8  Update weights of algorithms:
         $wa_j^{new} = wa_j \cdot e^{v_2 \cdot \alpha_t}$, if $A_t$ equals to the $j$-th
         element in $BA$; $wa_j^{new} = wa_j$, otherwise.
    3.9  Normalize weights of algorithms and set the
         updated weights for the next iteration:
         $wa_j = wa_j^{new} \cdot \left(\sum_{j=1}^{m} wa_j^{new}\right)^{-1}$
4   Create an ensemble for prediction:
    $\hat{y}_i = \arg\max_y\left(\sum_{t=1}^{T} \alpha_t \cdot \mathrm{I}\left[y = \hat{y}_i^{C_t}\right]\right)$
```

Figure 6. AdaBoost with stochastic algorithm selection

We design a function to perform weighted sampling with replacement for samples and algorithms. The function, *ResampleWithWeights*(X, W, k), takes three input parameters. The first one, X, is a set of objects (such as samples or algorithms). The second one, W, is a set of numbers, each of which exclusively corresponds to an object in X, and also $|X| = |W|$. The third one, k, is used to control the size of the output set. When it terminates normally, this function returns a new set X_{new} such that $X_{new} \bigcup X = X$ and also $|X_{new}| = k \leq |X|$. Moreover, the probability that an object in X would be selected in X_{new} (or the expected number of times that an object in X would be duplicated in X_{new}) is propositional to the value of its corresponding element in W, as given in (1).

$$P_r\left(x \in X_{new} \mid x \in X\right) \propto W_{\text{index of } x} \qquad (1)$$

Similar to the case that AdaBoost updates weights of samples according to weighted training errors, our algorithm updates the weight of a base learning algorithm by considering the quality of corresponding classifiers. From one iteration to another, the weight of a base learning algorithm is increased as the quality of the classifier based on it increases or as the weighted error rate given by the corresponding classifier decreases. Following the concept of AdaBoost in which an exponential function is employed to update weights of samples, we use the same exponential function to update weights of algorithms. different functions for this purpose are certainly worth further investigation.

Following the idea of regularization, there is a shrinkage coefficient v_2 used to work with the aforementioned stochastic process and to update weights of algorithms in a more gentle way. Its function is similar to that of the shrinkage coefficient v described earlier. It is used to smooth the stochastic process and further the selection process or the sampling process for algorithms. If we do not introduce v_2 in our algorithm, after a relatively small number of iterations, the (normalized) weight of some algorithm would be 1 (or very close to but still smaller than 1). If this happens, the transition probability of the algorithm would converge to 0 (or very close to but still larger than 0) and the selection process would be trapped in some algorithm. This means that the algorithm would always be selected afterward. Obviously, such a convergence happens with or without the use of the shrinkage coefficient v_2. However, we would like to slow down the convergence because we perform a relatively small number of iterations. This is also part of the reason that v is used. Usually, v_2 is much smaller than v because the number of algorithms is much smaller than the number of samples.

This paper is different from the paper [14] in three ways: First, we describe the intuition of the design of our algorithm in a way more detailed than the way taken by the authors of the paper [14]. Second, our algorithm selects base learning algorithms, while that proposed in the paper [14] selects classifiers. Third, this paper presents results from experiments on data sets from various application domains, while the paper [14] presents results only for a computer vision application.

4. EXPERIMENTS AND RESULTS

In experiments we consider synthetic data sets and also benchmark data sets obtained from references [20, 35]. There are 40 binary data sets considered here: 30 are real data sets and 10 are synthetic ones. These data sets are from different application domains and with a variety of characteristics. We use *one-versus-the-rest* to transform non-binary data sets into binary ones.

Table 1 presents characteristics of data sets used in experiments. Columns are the name, the number of samples, the number of features, and the number of principal components of a data set. The last column is the number of principal components suggested by the PCA [13] built in WEKA [9]. If the number of principal components is larger than the number of features, there would probably be hidden features and/or interactions among features.

We implement our algorithm using WEKA. Three algorithms are used as base algorithms in the classic AdaBoost. Decision stump algorithm (DS) generates a tree with a single decision node, while decision tree algorithm is employed to generate pruned decision trees (DT) and unpruned ones (UDT). DS and UDT are used as base learning algorithms in our algorithm (DS+UDT). This combination is interesting because these two base learning algorithms are different. DS is a high-variance and low-bias learning algorithm, and the generated classification models tend to be underfitting; UDT is a low-variance and high-bias learning algorithm, and it has a tendency to generate overfitting models.

For a classification model, its generalization capability is the capability to use samples it has seen to predict unseen samples. If its prediction performance for the training set is as good as that for the test set, a model demonstrates good generalization capability. In order to evaluate the prediction performance of our algorithm, we adopt the following procedure in experiments. Initially, we perform 10-90 random split for a data set. In other words, we randomly select 10% of samples without replacement to generate a training set. Then we use the rest 90% of samples to generate an independent test set. Such a pair of training and test sets is employed for a trial. We repeat these two steps 10 times and accordingly obtain 10 such pairs for 10 trails in an experiment. Afterward, we run experiments with 100 and 1000 iterations.

Table 1. Characteristics of data sets

Name		Size	Dim	PCs
German Credit Data		1000	20	42
Pima Indians Diabetes		768	8	8
Ionosphere		351	34	23
Chess End-Game		3196	36	31
Connectionist Bench		208	60	30
Heart Disease		303	13	16
Heart		294	13	15
Hypothyroid Negative		3772	29	25
Image Segmentation	brickface	2310	19	10
	cement	2310	19	10
	foliage	2310	19	10
	grass	2310	19	10
	path	2310	19	10
	sky	2310	19	10
	window	2310	19	10
Splice-junction Gene Sequences		3190	60	170
Vehicle Silhouettes	bus	846	18	7
	opel	846	18	7
	saab	846	18	7
	van	846	18	7
Zoo	amphibian	101	16	10
	bird	101	16	10
	fish	101	16	10
	insect	101	16	10
	invertebrate	101	16	10
	mammal	101	16	10
	reptile	101	16	10
Spambase		4601	57	48
Runshoes		60	10	9
Schizophrenic Eye-Tracking		340	13	12
Synthetic A		1000	20	299
Synthetic B		1000	20	62
Synthetic C		1000	20	90
Synthetic D		1000	20	59
Synthetic E		1000	16	490
Synthetic F		1000	5	5
Synthetic G		1000	11	10
Synthetic H		1000	6	20
Synthetic I		1000	6	365
Synthetic J*		1000	55	N/A

On one hand, for the classic AdaBoost, variables changing throughout trials in an experiment include the base algorithm (DS, DT, or UDT) and the shrinkage coefficient v (1 or 10^{-1}). Accordingly, 6 combinations are for the classic AdaBoost. For example, we use AdaBoost(DS, v=1) to denote Adaboost with DS and v set to 1. On the other hand, for our algorithm that adopts DS and UDT as base learning algorithms, variables include the shrinkage coefficients v (1 or 10^{-1}) and v_2 (1, 10^{-3}, or 10^{-6}). Therefore, 6 combinations are for our algorithm. Likewise, DS+UDT(v=1, v_2=1) is used to denote our algorithm with v set to1 and v_2 set to 1.

We average training and test error rates for every data set considered in experiments, and the means and standard deviations are calculated over 40 data sets. Training errors with 100 and 1000 iterations, are reported in Tables 2 and 3, respectively. Similarly, test errors are reported in Tables 4 and 5.

Table 2. Means and standard deviations (in parentheses) for training errors for 100 iterations

v	v_2	AdaBoost with			DS+UDT
		DS	DT	UDT	
1	1	0.11 (0.099)	0.053 (0.055)	0.049 (0.048)	0.069 (0.064)
	10^{-3}				0.069 (0.064)
	10^{-6}				0.068 (0.068)
10^{-1}	1	0.108 (0.101)	0.058 (0.056)	0.055 (0.055)	0.078 (0.068)
	10^{-3}				0.063 (0.066)
	10^{-6}				0.064 (0.064)

Table 3. Means and standard deviations (in parentheses) for training errors for 1000 iterations

v	v_2	AdaBoost wtih			DS+UDT
		DS	DT	UDT	
1	1	0.113 (0.1)	0.057 (0.056)	0.051 (0.056)	0.076 (0.07)
	10^{-3}				0.054 (0.053)
	10^{-6}				0.072 (0.063)
10^{-1}	1	0.111 (0.104)	0.057 (0.06)	0.056 (0.055)	0.074 (0.062)
	10^{-3}				0.054 (0.05)
	10^{-6}				0.062 (0.064)

Table 4. Means and standard deviations (in parentheses) for test errors for 100 iterations

v	v_2	AdaBoost with			DS+UDT
		DS	DT	UDT	
1	1	0.174 (0.141)	0.14 (0.138)	0.143 (0.136)	0.121 (0.114)
	10^{-3}				0.119 (0.119)
	10^{-6}				0.12 (0.123)
10^{-1}	1	0.17 (0.136)	0.148 (0.145)	0.147 (0.139)	0.122 (0.119)
	10^{-3}				0.117 (0.119)
	10^{-6}				0.118 (0.119)

Table 5. Means and standard deviations (in parentheses) for test errors for 1000 iterations

v	v_2	AdaBoost with			DS+UDT
		DS	DT	UDT	
1	1	0.173 (0.137)	0.143 (0.138)	0.148 (0.138)	0.12 (0.112)
	10^{-3}				0.118 (0.122)
	10^{-6}				0.122 (0.12)
10^{-1}	1	0.171 (0.142)	0.143 (0.143)	0.153 (0.144)	0.121 (0.113)
	10^{-3}				0.116 (0.121)
	10^{-6}				0.117 (0.121)

The best results for algorithms in Tables 2-5 are illustrated in Figures 7-10.

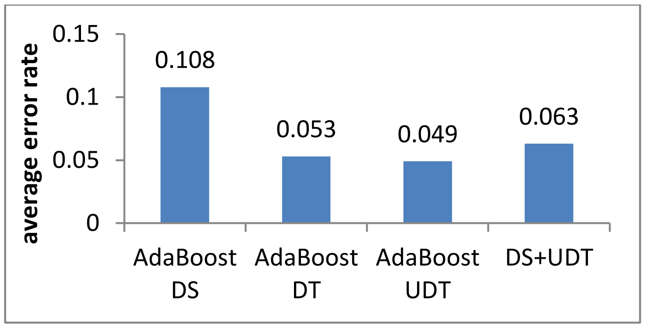


Figure 7. Training errors for 100 iterations

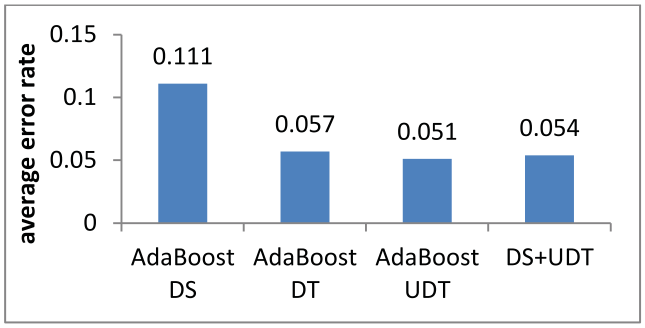


Figure 8. Training errors for 1000 iterations

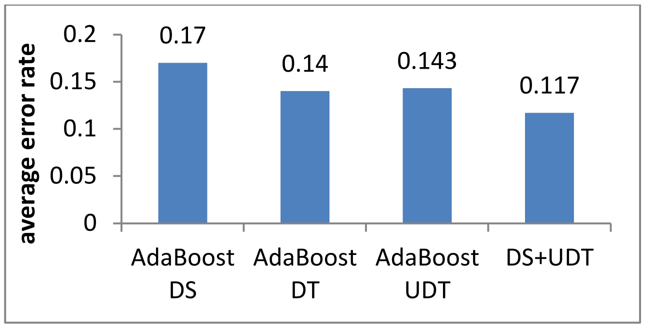


Figure 9. Test errors for 100 iterations

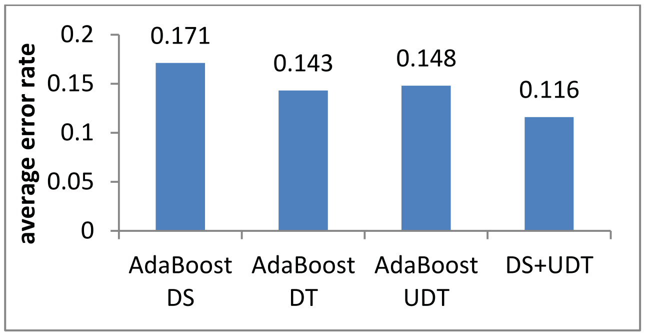


Figure 10. Test errors for 1000 iterations

For 100-iteration experiments, the lowest training error is given by AdaBoost(UDT, v=1), and error rates given by DS+UDT are only lower than those given by AdaBoost with DS. However, the lowest test error rate is given by DS+UDT($v=10^{-1}$, $v_2=10^{-3}$). In fact, all combinations for DS+UDT give lower test error rates if we compare them to all combinations for the classic AdaBoost. Similar observations could be made for 1000-iteration experiments. Additionally, both DS+UDT(v=1, $v_2=10^{-3}$) and DS+UDT($v=10^{-1}$, $v_2=10^{-3}$) give lower error rates in training and test if we compare them to all combinations for DS+UDT. If we compare them to all combinations for AdaBoost, only AdaBoost(UDT, v=1) gives a lower error in training. Test error rates given by DS+UDT are all lower than those given by AdaBoost with DS, DT, and UDT.

It is clear from above observations that DS+UDT outperforms AdaBoost with DS, DT, and UDT. First of all, DS+UDT outperforms AdaBoost with DS and UDT because they are two extremes: Models created by DS are too simple while those created by UDT are too complicated. Simple models often give high error rates, and we observe this from results given by AdaBoost with DS; complicated models tend to overfit training samples but perform poorly on unseen samples in a test set, and we observe this from results given by Adaboost with UDT. Not considering these two extremes but AdaBoost with DT, which is a popular setting in practice, we still obverse that DS+UDT provides better performance in terms of test error rates. Moreover, above observations also demonstrate that our algorithm provides better generalization capability. Our algorithm has a smaller performance *gap* between performance associated with seen samples and that associated with unseen samples.

As for the effect of the number of iterations, AdaBoost(DT, $v=10^{-1}$) decreases its training and test error rates as the number of iterations increases from 100 to 1000, while AdaBoost(DS, v=1) decreases its test error rate as we perform more iterations. Given small training sets (and given the fact that 10-90 random split makes them even smaller), we need to perform more iterations to avoid overfitting. However, DS+UDT shows more positive effects. Both DS+UDT(v=1, $v_2=10^{-3}$) and DS+UDT($v=10^{-1}$, $v_2=10^{-3}$) decrease training error rates as we perform more iterations, while all except DS+UDT(v=1, $v_2=10^{-6}$) do so. These observations, again, suggest that DS+UDT has a lower risk of being overfitting.

Tables 6 and 7 present comparisons in test error rates. The comparison between AdaBoost with DS and DS+UDT in presented in Table 6, and that between AdaBoost with UDT and DS+UDT is presented in Table 7. Here we consider the difference between the test error rates (with respect to a data set and trails on it). We use the one-tailed two-sample t-test. DS-(DS+UDT), as an example, is related to the difference between the test error rate from AdaBoost with DS and that from DS+UDT. In both tables, a number in an entry represents the number of data sets corresponding to significant or insignificant results from applying the test to the differences. The statistical significance test is conducted under 95% confidence level. For both DS-(DS+UDT) and UDT-(DS+UDT), we consider 4 cases for the differences: 1) Positive and significant. 2) Positive but insignificant. 3) Nonpositive but insignificant. 4) Nonpositive and insignificant. Here significant means that the corresponding p-value is less than 0.05. The first case implies the effectiveness of DS+UDT; the second and third cases tell us nothing but that DS+UDT performs as well as the classic AdaBoost with DS or UDT; the fourth case, however, presents a situation where DS+UDT does not demonstrate comparable prediction performance. As we can see from the Tables 6 and 7, DS+UDT outperforms AdaBoost with DS or UDT on most data sets, even though it is not always the winner: Depending on combinations of parameters, DS+UDT achieves significantly lower test error rates on 22 to 33 data sets, provides comparable performance on 2 to 13 data sets, and gives significantly higher test error rates on at most 4 data sets (and on 0 data sets for most of the time).

Setting v_2 to 1 actually destroys the nature of the stochastic process because some algorithm will (almost) always be selected. If the algorithm is a good fit for the data set then we would have a good classification model; otherwise, we miss an opportunity to use another heterogeneous algorithm to compensate for the weakness of the algorithm. If we set v_2 to a very small value (close to 0), the process becomes (almost) fully random and we miss an opportunity to utilize the strength of the algorithm that is potentially a good fit for the data set.

Table 6. Statistical test results for 100 iterations

	v	v_2	Positive diff.; DS+UDT is better		Nonpositive diff.; it is not better	
			sig.	insig.	insig.	sig.
DS-(DS+UDT)	1	1	29	9	2	0
		10^{-3}	33	5	2	0
		10^{-6}	30	6	4	0
	10^{-1}	1	30	6	4	0
		10^{-3}	30	7	3	0
		10^{-6}	27	9	4	0
UDT-(DS+UDT)	1	1	26	6	5	3
		10^{-3}	22	8	8	2
		10^{-6}	22	11	3	4
	10^{-1}	1	24	8	5	3
		10^{-3}	23	9	6	2
		10^{-6}	24	6	9	1

Table 7.Statistical test results for 1000 iterations

	v	v_2	Positive diff.; DS+UDT is better		Nonpositive diff.; it is not better	
			sig.	insig.	insig.	sig.
DS-(DS+UDT)	1	1	31	4	4	1
		10^{-3}	29	7	3	1
		10^{-6}	31	4	5	0
	10^{-1}	1	27	10	3	0
		10^{-3}	26	11	3	0
		10^{-6}	28	9	2	1
UDT-(DS+UDT)	1	1	23	8	7	2
		10^{-3}	26	8	3	3
		10^{-6}	23	8	6	3
	10^{-1}	1	23	10	3	4
		10^{-3}	30	3	6	1
		10^{-6}	22	13	5	0

5. CONCLUSION

AdaBoost creates diversity by manipulating training sets, and it demonstrates superior performance when the number of data samples is sufficiently large or the number of iterations is sufficiently large. Nevertheless, we observe that it over-emphasizes hard-to-classify samples and generates training sets of low variety if we perform a limited number of iterations. Moreover, we could achieve better diversity by employing heterogeneous algorithms. Following the idea of employing heterogeneous algorithms in ensemble learning, we propose a variant of AdaBoost and our algorithm adopts different base learning algorithms (such as decision stump algorithm and decision tree algorithm generating unpruned trees) in different iterations. The other feature that distinguishes our algorithm from other variants of AdaBoost is that it selects the base learning algorithms using a stochastic process where their earlier performance is considered. Experiment results show that, our algorithm does not necessarily lower average training error rates but it achieves lower average test error rates, which means that our algorithm is less prone to overfitting. Our algorithm employing two different base learning algorithms performs better than does the classic AdaBoost with either of these two algorithms.

6. ACKNOWLEDGMENTS

The author would like to thank anonymous reviewers for their valuable time.

7. REFERENCES

[1] Leo Breiman. 1996. Bagging predictors. *Mach. Learn.* 24, 2 (August 1996), 123-140. DOI=http://dx.doi.org/10.1023/A:1018054314350

[2] Leo Breiman. 2001. Random forests. *Mach. Learn.* 45, 1 (October 2001), 5-32. DOI=http://dx.doi.org/10.1023/A:1010933404324

[3] Gavin Brown, Jeremy Wyatt, Rachel Harris, and Xin Yao. 2005. Diversity creation methods: a survey and categorization. *Inf. Fusion* 6, 1 (March 2005), 5-20. DOI=http://dx.doi.org/10.1016/j.inffus.2004.04.004

[4] Gavin Brown. 2010. Ensemble Learning. *Encyclopedia of Machine Learning*. Springer US, 312-320. DOI=http://dx.doi.org/10.1007/978-0-387-30164-8_252

[5] Xavier Carreras, Lluís Màrquez, and Lluís Padró. 2002. Named Entity Extraction using AdaBoost. In P*roceedings of the 6th Conference on Natural Language Learning* (COLING-02). Association for Computational Linguistics, 1-4. DOI=http://dx.doi.org/10.3115/1118853.1118857

[6] Xavier Carreras, Lluís Màrquez, and Lluís Padró. 2003. A simple named entity extractor using AdaBoost. In *Proceedings of the 7th Conference on Natural Language Learning at HLT-NAACL 2003* (CONLL '03). Association for Computational Linguistics, 152-155. DOI=http://dx.doi.org/10.3115/1119176.1119197

[7] Jinhui Chen, Yasuo Ariki, and Tetsuya Takiguchi. 2013. Robust facial expressions recognition using 3D average face and ameliorated adaboost. In *Proceedings of the 21st ACM International Conference on Multimedia* (MM '13). ACM, 661-664. DOI=http://dx.doi.org/10.1145/2502081.2502173

[8] Yoav Freund and Robert E. Schapire. 1996. Experiments with a new boosting algorithm. In *Proceedings of the 13th International Conference on Machine Learning* (ICML '96). Lorenza Saitta (Ed.). Morgan Kaufmann, 148-156.

[9] Mark Hall, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, and Ian H. Witten. 2009. The WEKA data mining software: an update. *SIGKDD Explor. Newsl.* 11, 1 (November 2009), 10-18. DOI=http://dx.doi.org/10.1145/1656274.1656278

[10] Kuo-Wei Hsu and Jaideep Srivastava. 2009. Diversity in combinations of heterogeneous classifiers. In *Proceedings of the 13th Pacific-Asia Conference on Advances in Knowledge Discovery and Data Mining* (PAKDD '09). Springer-Verlag, 923-932. DOI=http://dx.doi.org/10.1007/978-3-642-01307-2_97

[11] Kuo-Wei Hsu and Jaideep Srivastava. 2009. An empirical study of applying ensembles of heterogeneous classifiers on imperfect data. *New Frontiers in Applied Data Mining, Lecture Notes in Computer Science*, Vol. 5669/2010, pp. 28-39. DOI=http://dx.doi.org/10.1007/978-3-642-14640-4_3

[12] Wei Hu and Weiming Hu. 2005. Network-Based Intrusion Detection Using Adaboost Algorithm. In *Proceedings of the 2005 IEEE/WIC/ACM International Conference on Web*

Intelligence (WI '05). IEEE Computer Society, 712-717. DOI=http://dx.doi.org/10.1109/WI.2005.107

[13] Ian Jolliffe. 2005. Principal component analysis. *Encyclopedia of Statistics in Behavioral Science.* John Wiley & Sons, Ltd. DOI= http://dx.doi.org/10.1002/0470013192.bsa501

[14] Loıc Jourdheuil, Nicolas Allezard, Thierry Chateau, and Thierry Chesnais. 2012. Heterogeneous Adaboost with Real Time Constraints: Application to the Detection of Pedestrians by Stereovision. In *Visapp, International Conference on Vision Theory and Applications*.

[15] Ludmila I. Kuncheva and Christopher J. Whitaker. 2001. Ten measures of diversity in classifier ensembles: limits for two classifiers. In *A DERA/IEE Workshop on Intelligent Sensor Processing*. IEEE. DOI=http://dx.doi.org/10.1049/ic:20010105

[16] Ludmila I. Kuncheva, M. Skurichina and R. P. W. Duin. 2002. An experimental study on diversity for bagging and boosting with linear classifiers. *Inf. Fusion* 3, 4 (December 2002), 245-258. DOI=http://dx.doi.org/10.1016/S1566-2535(02)00093-3

[17] Ludmila I. Kuncheva and Christopher J. Whitaker. 2003. Measures of diversity in classifier ensembles and their relationship with the ensemble accuracy. *Mach. Learn.* 51, 2 (May 2003), 181-207. DOI=http://dx.doi.org/10.1023/A:1022859003006

[18] Ludmila I. Kuncheva. 2003. That elusive diversity in classifier ensembles. In *Proceedings of the 1st Iberian Conference on Pattern Recognition and Image Analysis* (IbPRIA '03). Springer, 1126-1138. DOI=http://dx.doi.org/10.1007/978-3-540-44871-6_130

[19] Avisek Lahiri and Prabir Biswas. 2014. Knowledge Sharing and Cooperation Based Adaptive Boosting for Robust Eye Detection. In *Proceedings of the 2014 Indian Conference on Computer Vision Graphics and Image Processing* (ICVGIP '14). ACM, Article 13. DOI=http://dx.doi.org/10.1145/2683483.2683496

[20] M. Lichman. 2013. UCI Machine Learning Repository [http://archive.ics.uci.edu/ml]. Irvine, CA: University of California, School of Information and Computer Science.

[21] Jiebo Luo, Jie Yu, Dhiraj Joshi, and Wei Hao. 2008. Event recognition: viewing the world with a third eye. In *Proceedings of the 16th ACM International Conference on Multimedia* (MM '08). ACM, 1071-1080. DOI=http://dx.doi.org/10.1145/1459359.1459574

[22] Yunqian Ma. 2012. *Ensemble machine learning: Methods and applications*. Springer.

[23] Prem Melville and Raymond J. Mooney. 2003. Constructing diverse classifier ensembles using artificial training examples. In *Proceedings of the 18th international joint conference on Artificial intelligence* (IJCAI '03). Morgan Kaufmann Publishers Inc., 505-510.

[24] Prem Melville and Raymond J. Mooney. 2005. Creating diversity in ensembles using artificial data. *Inf. Fusion* 6, 1 (March 2005), 99-111. DOI= http://dx.doi.org/10.1016/j.inffus.2004.04.001

[25] David Opitz and Richard Maclin. 1999. Popular ensemble methods: An empirical study. *Journal of Artificial Intelligence Research* 11 (August 1999), 169-198. DOI= http://dx.doi.org/10.1613/jair.614

[26] Robi Polikar. 2006. Ensemble based systems in decision making. *IEEE Circuits Syst. Mag.* 6, 3 (September 2006), 21-45. DOI=http://dx/doi.org/10.1109/MCAS.2006.1688199

[27] Robi Polikar. 2007. Bootstrap-inspired techniques in computational intelligence. *IEEE Signal Process. Mag.* 24, 4 (August 2007), 56-72. DOI=http://dx.doi.org/10.1109/MSP.2007.4286565

[28] J. Ross Quinlan. 1993. *C4.5: Programs for Machine Learning*. Morgan Kaufmann Publishers, Inc.

[29] Romesh Ranawana and Vasile Palade. 2006. Multi-Classifier systems: Review and a roadmap for developers. *Int. J. Hybrid Intell. Syst.* 3, 1 (January 2006), 35-61.

[30] Attila Reiss, Gustaf Hendeby, and Didier Stricker. 2013. Confidence-based multiclass AdaBoost for physical activity monitoring. In *Proceedings of the 2013 International Symposium on Wearable Computers* (ISWC '13). ACM, 13-20. DOI=http://dx.doi.org/10.1145/2493988.2494325

[31] Xiaona Song, Ting Rui, Zhengjun Zha, Xinqing Wang, and Husheng Fang. 2015. The AdaBoost algorithm for vehicle detection based on CNN features. In *Proceedings of the 7th International Conference on Internet Multimedia Computing and Service* (ICIMCS '15). ACM, Article 5. DOI=http://dx.doi.org/10.1145/2808492.2808497

[32] Jaree Thongkam, Guandong Xu, Yanchun Zhang, and Fuchun Huang. 2008. Breast cancer survivability via AdaBoost algorithms. In *Proceedings of the 2nd Australasian Workshop on Health Data and Knowledge Management* (HDKM '08). Australian Computer Society, Inc., 55-64.

[33] R. M. Valdovinos, J. S. Sánchez, and E. Gasca. 2007. Influence of resampling and weighting on diversity and accuracy of classifier ensembles. In *Proceedings of the 3rd Iberian conference on Pattern Recognition and Image Analysis, Part II* (IbPRIA '07). Springer-Verlag, 250-257. DOI=http://dx.doi.org/10.1007/978-3-540-72849-8_32

[34] Giorgio Valentini and Francesco Masulli. 2002. Ensembles of Learning Machines. In *Proceedings of the 13th Italian Workshop on Neural Nets-Revised Papers* (WIRN VIETRI 2002). Springer-Verlag, 3-22.

[35] Pantelis Vlachos. 2005. StatLib datasets archive. URL http://lib.stat.cmu.edu/datasets.

[36] Michał Woniak, Manuel Graña, and Emilio Corchado. 2014. A survey of multiple classifier systems as hybrid systems. *Inf. Fusion* 16 (March 2014), 3-17. DOI= http://dx.doi.org/10.1016/j.inffus.2013.04.006

[37] Xindong Wu, Vipin Kumar, J. Ross Quinlan, Joydeep Ghosh, Qiang Yang, Hiroshi Motoda, Geoffrey J. McLachlan, Angus Ng, Bing Liu, Philip S. Yu, Zhi-Hua Zhou, Michael Steinbach, David J. Hand, and Dan Steinberg. 2007. Top 10 algorithms in data mining. *Knowl. Inf. Syst.* 14, 1 (December 2007), 1-37. DOI=http://dx.doi.org/10.1007/s10115-007-0114-2

[38] Zhi-Hua Zhou. 2012. *Ensemble methods: Foundations and algorithms*. CRC press.