Rational Ordering and Clustering of Examples for Incremental Learning

Seppo Puuronen[1], Vagan Terziyan[2], and Borys Omelayenko[2]

[1] *University of Jyväskylä, P.O.Box 35,*
FIN-40351 Jyväskylä, Finland,
e-mail: sepi@jytko.jyu.fi

[2] *Kharkov State Technical University of Radioelectronics, 14 Lenin Avenue,*
310166 Kharkov, Ukraine,
e-mail: vagan@kture.cit-ua.net, vagan@ytko.jyu.fi, omelayenko@hotpop.com

Abstract

An incremental learning paradigm assumes that there is a sequence of examples, which is not under control of the learning system. However in some cases the learner can manage the order of some objects from the sequence by updating the weights with the same incremental algorithm. In this paper we focus on one possible approach to manage the order of such examples to achieve higher overall accuracy of the classification. We suggest a method that groups these examples into clusters interleaved by recalculations of weights of the classifiers from the ensemble. The method first orders the examples into the descending order according to the amount of differences in the classifiers' outputs. Then the examples are grouped into clusters for which the sum of differences in opinions are as equal as possible. Our experiments show that the proposed method allows to reduce the number of weight recalculations up to one fourth without any loss in classification accuracy.

1. Introduction

One of the most important directions in improvement of the Knowledge Discovery techniques is the integration of multiple classification methods with ensembles of classifiers. Weighted voting usually combines outputs of the classifiers in an ensemble. Wide range of applications requires incremental updating of classifiers' weights, where each new weight is derived from the previous one using additive or multiplicative formula.

Only a few methods to management the order of examples are included in the set of well-known on-line classification algorithms. The bagging technique [3] is one of these. It repeatedly selects m training examples from the original training set randomly with replacement and then generates T classifiers using these T bootstrap samples. In such technique the final classifier is built from the most precise classifiers. Another well-known technique, which manipulates the order of training examples, is the boosting algorithm [6]. The boosting algorithm uses a probability distribution over the training examples. On the i-th iteration it draws a training set of size m from the original training set by sampling with replacement according to the probability distribution. Then it uses this training set to produce the i-th classifier. The probability distribution represents the order of learning and there are few advances to improve this distribution formula in order to reach performance gain. Both the bagging and boosting algorithms seem to be competitive in batch tasks [10,6].

There are two kinds of classification systems: incremental and batch systems. The incremental classification systems change a classification scheme, as new instances are processed [12]. The batch systems proceed the whole collection of training instances to form a single concept and only then classify test examples. Most of the fastest classification algorithms are incremental and use simple internal scheme. The Weighted Majority algorithm [8,2] uses an ensemble of simple weighted classifiers, while Naive Bayes [5,4] uses two-level probabilistic network. Both do not modify the structure of the classifier, but update weights (or probabilities in the latter case).

Incremental learning paradigm assumes that there is a sequence of examples, which is not under control of the learning system. But in some cases the learner can manage the order of some objects from the sequence, but proceed them and update the weights with the same incremental algorithm. For example, in [2] the performance of batch based on C4.5 and incremental (Winnow) systems were compared on the calendar-scheduling domain. For better comparison incremental algorithm received the feedback with known classes for events of the day only at night. Here incremental system was generally controlled by the order of examples (day

after day) but it had the possibility to manage the order at nights.

One possible way to manage the examples in incremental classification is to divide the tasks into clusters so, that some parameters of the classification process (context or classifiers responsibility) can be considered to be stable within each cluster.

A clustering algorithm, called contextual clustering, was successfully applied in domains with hidden changes in the context of the concepts [7]. The whole domain was partitioned into clusters according to the apparent similarity of their contexts. The context was supposed to be stable within each cluster. The clustered algorithm based on this batch learner had accuracy up to 30% better, than a non-clustered batch learner did. This paper focuses on the problem of clustering classification tasks when the final classification is derived from the classifications of individual classifiers in an ensemble by weighted voting with dynamic weight recalculations.

We suggest a clustered voting sequence, which consists of clusters of classification tasks and weight recalculations between them. In our method classification tasks are ordered into the descending order according to the amount of differences in predictions of the classifiers and then the clusters are formed from the ordered list so that their sums of differences in predictions are as equal as possible. This clustering leads to the sequence that results more frequent changes of weights in the beginning and thus produces faster learning of the weights of the component classifiers.

In chapter 2 the basic concepts used thorough the paper are introduced. In the third chapter we present our classification task clustering procedure. In chapter 4 the details of the steps 2 and 3 of the previous procedure are presented in an algorithmic form. Chapter 5 discusses our experiments and chapter 6 concludes with some further research suggestions.

2. Basic Concepts

In this chapter we define the basic concepts and notations used thorough the paper. Let the original set of the *d* classification tasks be $D = \{ D_1, D_2, \ldots, D_d \}$ and the ensemble of the *n* classifiers be $C = \{ C_1, C_2, \ldots, C_n \}$. The weight from the set of correspondent weights $r = \{ r_1, r_2, \ldots, r_n \}$ is associated to each classifier. During the classification process the weights of the classifiers are updated only after the end of each cluster and are not changed after any example within one cluster. Let $| O_{ij} |$ be the matrix of individual classifiers' predictions, where O_{ij} is the class that classifier C_j assigns to the example D_i. Let $S = \{S_1, S_2, \ldots, S_k\}$ be the set of k disjoint clusters S_i of original classification examples, such that $\bigcup_{i=1}^{k} S_i = D$.

We use the additive weight recalculation scheme for incremental learning. Information about the mistakes made by each classifier on the cluster is collected to do weight recalculation according the formula (1):

$$r_i^{\nu+1} = r_i^{\nu} + \Delta r_i^{\nu}, \tag{1}$$

where r_i^{ν} denotes the weight of the i-th classifier in the ν-th cluster S_ν and Δr_i^{ν} denotes the punishment or prize value earned by the i-th classifier while processing the classification tasks of the ν-th cluster. It is calculated using the formula (2):

$$\Delta r_i^{\nu} = \delta_i^{\nu} \cdot \frac{\mu^{\nu} \cdot (\mu^{\nu} - con_i^{\nu})}{m}, \tag{2}$$

$$\mu^{\nu} = \frac{1}{n} \cdot \sum_{j}^{n} con_j^{\nu},$$

where e_i^{ν} is the number of misclassification errors made by the *i*-th classifier during classifying examples from the ν-th cluster, $e_{\max}$ is the corresponding maximal possible amount of misclassification errors for this cluster and the parameter δ_i^{ν} depends on the selected weight recalculation strategy.

The formulas (1) and (2) are based on the following basic assumptions: (a) all the classifiers have the same initial weights; (b) a classifier raises its weight after cluster of classification tasks if it gives classifications, which has less conflicts with the final classification than the classifiers in average, otherwise it loses part of its weight.

In this paper we calculate parameter δ_i^{ν} as follows:

$$\delta_i^{\nu} = \frac{r_i^{\nu} \cdot (n - r_i^{\nu})}{n - 1} \tag{3}$$

The initial weights of the classifiers are set to 0.5. The updating rule (1) corresponds to the additive scheme of weight recalculation.

3. Clustering Classification Tasks

In this chapter we present the clustering procedure. The basic assumption for it is that the weights are kept unchanged during the voting about classification tasks inside a cluster and that the weights are recalculated after each cluster has been processed. The procedure includes three main steps:

Step 1: Selecting the number of weight recalculations

Estimate the amount of resources that are available to the weight recalculations. Calculate the maximum number of weight recalculations under this resource restriction. If the maximum number of recalculations is equal to or higher than the amount of the classification tasks then set the number of weight recalculations k same as the amount of the classification examples. Otherwise set the number of weight recalculations k to the maximum number under resource restriction.

Step 2: Ordering the examples

For each example we first calculate the average difference between opinions of the classifiers on this example. Then we order the examples in the descending order according to the amount of classification differences between different classifiers. The objective of this step is to bring the hardest classification tasks in the beginning of the classification task sequence. The classifiers start to vote with the same weights and this ordering supports faster weight learning in the beginning of the voting process.

Step 3: Assigning classification tasks into clusters

3.1. If the number of weight recalculations k was not set under resource restrictions then assign one example to each of the first k-1 clusters in the order produced by the step 2, and assign the remaining examples to the k-th cluster.

3.2. Otherwise use a formula (4) to calculate the threshold value Θ. Then assign the examples to the first cluster in the order produced in the step 2 until the sum of classification differences of the assigned examples is higher than the threshold value. Proceed in the similar way with the next k-1 clusters and after that assign the remaining tasks to the k-th cluster.

$$\Theta = \frac{1}{k+1}\sum_{i=1}^{k}\sqrt{\tfrac{1}{n}\sum_{j=1}^{n}(\overline{O}-O_y)^2}\ , \qquad (4)$$

The key idea of clustering in the step 3 is to distribute the examples into clusters so that the total amount of differences is divided between clusters as equally as possible when the hardest examples are taken first. The goal is to have about the same amount of weight changes during each weight recalculation. Of course, in extreme cases the above process can lead to a very biased distribution on weight changes where only the first few changes spend the most of the differences. Anyway it guarantees that the classification tasks with biggest differences are taken into account in the beginning of the voting process.

4. The Clustering Algorithm

This chapter describes step 2 and step 3.2 of the clustering process presented in chapter 4 as an algorithm presented in Figure 1. The result of step 1 of the algorithm is supposed to be given as input and the special cases of step 3.1 are not considered. Thus the algorithm takes the examples D, the matrix of the opinions of classifiers on the examples $|O_{ij}|$, and the number k of the weight recalculations as an input. The algorithm produces the clustering of the examples into the subsets $S_1, S_2, \ldots, S_k, \bigcup_{i=1}^{k} S_i = D$.

There are three possible kinds of classes considered in the classification area. Continuous classes are real values, and classification into such classes is regression. Linear discrete (or integer) classes have only discrete values, but the relations of strict order are defined on class space. And nominal (or symbolic) classes have discrete values without any order relations between classes. Formulas (4) and (5) uses Euclidean's distance between classes, which has sense only for linear discrete classes. Such classes are widely used to represent discretized real values. Also formulas (4) and (5) can be easily updated to implement another distance metric.

While it is supposed that classes are linear discrete, it is possible to calculate their average and standard deviation. The standard deviation of opinions is used as a classification difference measure. This standard deviation D_i^C of the opinions about one example D_i is as follows:

$$D_i^C = \sqrt{\tfrac{1}{n}\sum_{j=1}^{n}(\overline{O}-O_{ij})^2}\ , \qquad (5)$$

where O_{ij} is the class of the example D_i given by the classifier C_j, and $\overline{O}$ is the average class of the classifications O_{ij} made for the example D_i.

The deviation of the opinions on the examples from the cluster S is defined as follows:

$$D(S) = \sum_{\forall D_i \in S} D_i^C\ . \qquad (6)$$

Input: D, k, $|O_{ij}|$.

Output: the clusters $S_1, S_2, \ldots, S_k$

0. Initialize the clusters: $S_i = \varnothing,\ i = \overline{1,k}$

1. Calculate D_i^C for every vote $i = \overline{1,d}$.

2. Sort the examples into descending order by D_i^C.

3. Calculate $D_{\max} = \frac{1}{k} \cdot \sum_{i=1}^{d} D_i^C$.

4. **Let** *I:=0, J:=1*

5. *I:=I+1*

6. **If** *i>d* **Then Goto** step 9.

7. **If** $D(S_j) < D_{\max}$ **Then** J:=J+1

Else $S_j := S_j \cup D_i$.

8. **Goto** step 5

9. **End**

Figure 1. The algorithm for clustering

The example

As an example, let us consider a case where four classifiers vote on 15 classification tasks. Classifications of the classifiers are presented in Table 1.

Table 1. Classifications of the classifiers

	C_1	C_2	C_3	C_4
D_1	6	12	2	8
D_2	2	3	8	1
D_3	5	7	9	12
D_4	12	13	7	4
D_5	7	3	12	13
D_6	6	7	12	3
D_7	13	9	5	10
D_8	5	2	8	6
D_9	5	3	9	11
D_{10}	7	2	6	10
D_{11}	13	4	2	1
D_{12}	8	7	2	8
D_{13}	5	5	5	5
D_{14}	4	5	5	7
D_{15}	3	8	13	2

The columns of this table correspond to the classification tasks and the rows of this table correspond to the classifiers. Classifications are integers from 1 to 13. Let us assume that the distance between classes in this example is measured as absolute difference between appropriate integer numbers. Table 2 includes classification tasks ordered in the descending order according to standard deviations of the classifications.

The sum of standard deviations is 43.57. Assume that three weight recalculations (*k*=3) are wanted. Thus $D_{\max} = \frac{1}{4} \cdot 43{,}57 = 10{,}89$. The algorithm will produce four subsets (clusters), as presented in Table 2, as follows:

$$S_1 = \{D_{11}, D_{15}\}, \quad S_2 = \{D_5, D_4, D_1\},$$
$$S_3 = \{D_6, D_9, D_7, D_{10}\}, \quad S_4 = \{D_2, D_3, D_{12}, D_8, D_{14}, D_{13}\}.$$

Table 2. Ordered classification tasks

	D^C	*Sum*	*Clusters*
D_{11}	4,74		
D_{15}	4,38	9,12	S_1
D_5	4,02		
D_4	3,67		
D_1	3,61	20,42	S_2
D_6	3,24		
D_9	3,16		
D_7	2,86		
D_{10}	2,86	32,64	S_3
D_2	2,69		
D_3	2,59		
D_{12}	2,49		
D_8	2,17		
D_{14}	1,09		
D_{13}	0,00	43,57	S_4

The graphical illustration for the ordering and partitioning in the example is presented in Figure 2.

5. Experimental Results

We use the Heart Cleveland data set from the UCI Machine Learning Repository of Databases [9] for experiments. This data set contains about 270 examples with thirteen numerical attributes and five classes. We create an ensemble of five 1-Nearest Neighbor classifiers, each of them keeping one prototype per class.

The accuracy of the ensemble was investigated in different clustering modes. First, we apply the above algorithm punctually with descending order of classification differences. We name this mode as the descending order mode. Second, we change the word «descending» in step 2 of the algorithm to «ascending». Such a modified algorithm starts with less conflicting examples with similar classifications and then continues with more conflicting examples. We name this mode as the ascending order mode. These two modes allow us to investigate accuracy gain provided by the ordering of examples. Finally, we remove step 2 of the algorithm to perform no sorting. This gives the third mode, the original order mode. This mode shows how well the clustering improves the accuracy without any ordering.

The results of experimental investigation of the algorithm in three modes are presented in Figure 3. Three lines in Figure 3 show the accuracy of the algorithm demonstrated with different number of clusters under the descending, ascending, and normal orders. The vertical

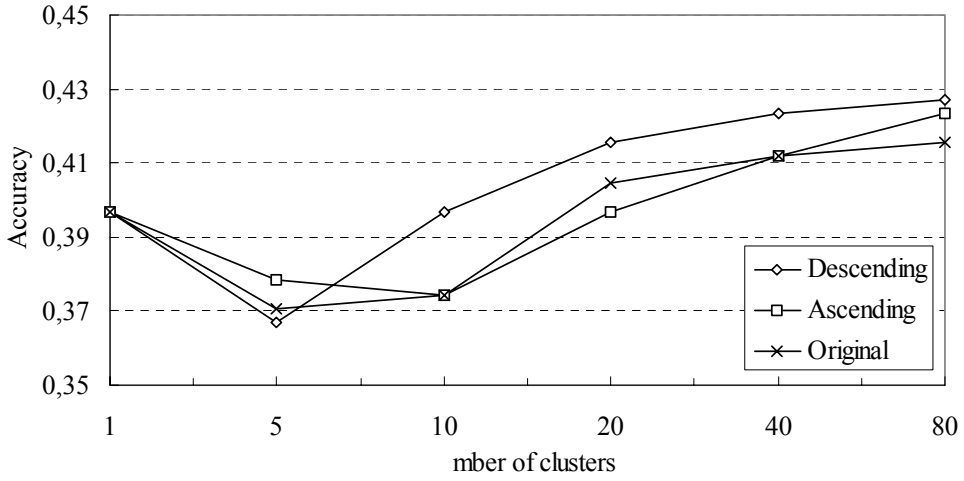

Figure 3. Comparison of different modes of the algorithm

axis shows the accuracy while the horizontal axis represents the number of clusters.

At first, we compare the proposed algorithm in the descending order mode with the original order to understand how does clustering improve classification results. The case with only one cluster and one weight recalculation represents simple voting with equal weights of the classifiers. Simple voting gives equal accuracy under all three clustering orders. Low number of clusters decreases the accuracy, but the accuracy begins to grow when five or more clusters are allowable. Ten clusters for the descending order give the same accuracy as simple voting, twenty clusters are needed for ascending or original order to outperform the descending order. The case of big number of clusters is the opposite of simple voting, it represents frequent weight recalculations. The proposed algorithm with 20 clusters has better accuracy than non-ordered clustering with 80 clusters. This is the first experimental result: the proposed algorithm allows reducing the number of weight recalculations few times without accuracy lost.

The comparison of the descending order and the ascending order shows the gain made by the ordering. The descending order provides the accuracy up to 2% better than the ascending order. This gain reduces with increasing the number of clusters and, hence, weights recalculations. This gives the second experimental result: the descending ordering gains 2% of accuracy, that is statistically not significant, and this accuracy gain falls while number of clusters grows. The detailed dynamics of classifiers' weights helps to understand why the proposed clustering works. Figure 4, Figure 5 and Figure 6 presents weights of each classifier while processing classification tasks ordered into ten clusters in the descending, ascending, and original order respectively.

Figure 4 shows that the expected fact that cluster size grows with cluster number. Each classifier has initial weight equal to 0.5 before proceeding the first cluster, so classifiers' ranks are close to each other after the first cluster (marked with «1» on the horizontal axis). Classifiers weights change while processing other clusters and classifier C_3 obtains the highest weight, it forms the ensemble opinion and the ensemble accuracy. The algorithm learned the weights fast under the descending order.

Figure 5 shows that the ascending order mode needs more time to learn that C_3 is the best classifier of the ensemble. For a long time other ensemble classifiers influence the opinion and reduce the ensemble accuracy. Also it is possible that C_3 is not confident in classification tasks that were less conflicting and this is the course of its fast weight rising under the descending order and low weight rising under the ascending order. Further experiments are needed to understand this aspect better.

Figure 6 shows that unordered classification tasks provide worse classifiers weight change than ordered tasks. The proportion of the classifiers' weights keeps the same from the first cluster to the last one. We can notice that all learning occurs after the first weight recalculation, when the order of classifiers by their weights is done. This explains why great number of clusters and weight recalculations does not improve the accuracy of the original order comparing with the descending order.

Note that clustering or ordering alone does not improve classification results significantly, only the combination of both allows to improve the accuracy and to reduce the number of weight recalculations.

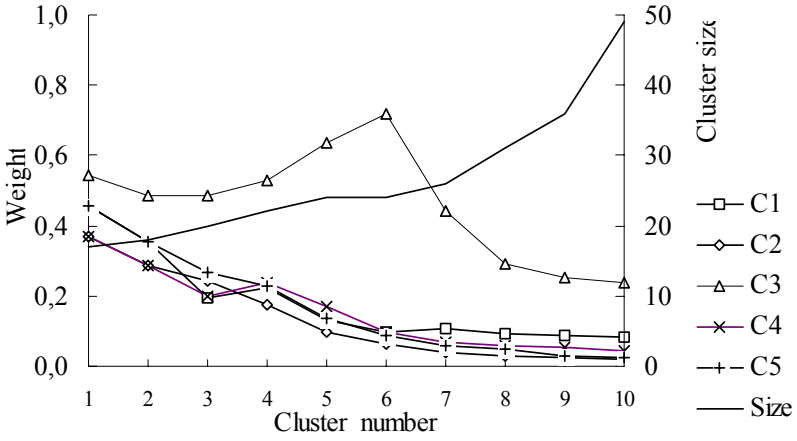

Figure 4. Descending order, ten clusters

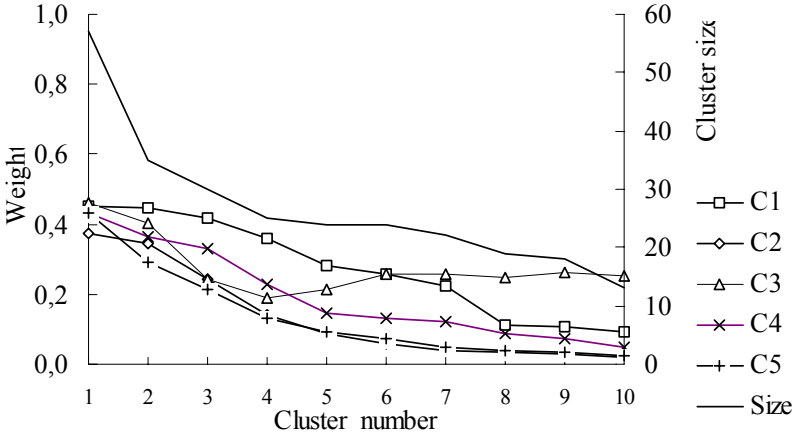

Figure 5. Ascending order, ten clusters

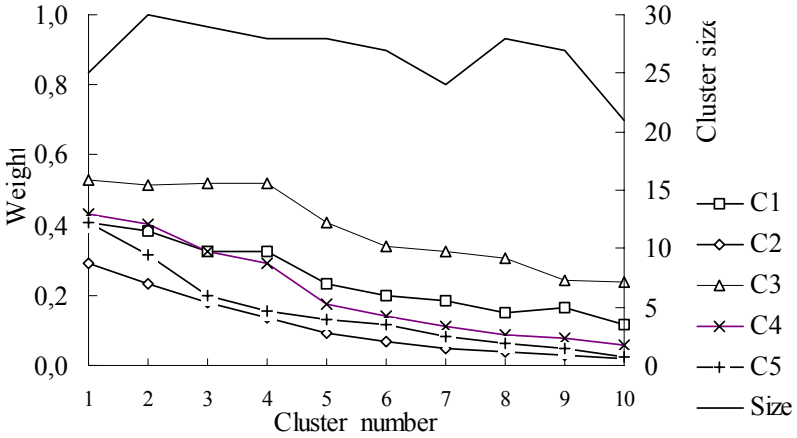

Figure 6. Original order, ten clusters

6. Conclusions

In this article we have discussed the problem of ordering of examples and rational weight recalculation in weighted voting. Our focus has been on the problem of clustering examples to be classified to derive a voting sequence when weight recalculation occurs between clusters. In this paper we suggest and investigate special clustering procedure that produces a sequence of examples. The beginning of the sequence is formed with examples, on which opinions of the classifiers had the biggest differences. The sequence ends with less conflicting examples. Then the sequence is divided into clusters with about the same amount of differences between classifiers' predictions. We place weight recalculation points between clusters. This leads to more frequent weight changes in the beginning of the classification and because of this to faster learning.

Experiments have shown that the proposed algorithm allows reducing the number of weight recalculations four times without accuracy lost in the investigated domain due to clustering. Clustering or ordering alone does not improve classification results significantly, only the combination of both allows to improve the accuracy and to reduce the number of weight recalculations.

More comprehensive experiments are planned as a future research to find the areas where present clustering algorithm will show better results and to understand its behavior deeply.

Acknowledgment. This research has been partly supported by the grant from the Academy of Finland.

7. References

1. L. Asker and R. Maclin, "Ensembles as a Sequence of Classifiers", in *Proceedings of the International Joint Conference on Artificial Intelligence IJCAI-97*, (Nagoya, Aichi, Japan), August 1997.
2. A. Blum, "Empirical Support for WINNOW and Weighted-Majority Algorithms: Results on a Calendar Scheduling Domain", *Machine Learning*, vol. 26, no. 1, pp. 5-24, 1997.
3. L. Breiman, "Bagging Predictors", *Machine Learning*, vol. 24, pp. 123-140, 1996.
4. T. Dietterich, "Machine-Learning Research: Four Current Directions", *AI Magazine*, vol. 18, no. 4, pp. 97-136, 1997.
5. R. O. Duda and P. E. Hart, *Pattern Classification and Scene Analysis*, New York, Wiley, 1973.
6. Y. Freund and R. Schapire, "Experiments with a new boosting algorithm", in *Proceedings of the Thirteenth International Conference on Machine Learning*, (Bari, Italy), pp. 148-156, July 1996.
7. M. Harries, C. Sammut, and K. Horn, "Extracting Hidden Context", *Machine Learning*, vol. 32, no. 2, pp. 101-126, 1998.
8. N. Littlestone and M. Warmuth, "The Weighted Majority Algorithm", *Information and computation* vol. 108, no. 2, pp. 212-261, 1994.
9. C. Merz and P. Murphy, *UCI Repository of Machine Learning Databases*, Available on WWW: www.ics.uci.edu/~mlearn/MLRepository.html, 1998.
10. J. Quinlan, "Bagging, Boosting, and C4.5", in: *Proceedings of the Thirteenth National Conference on Artificial Intelligence AAAI'96*, (Portland, Oregon), AAAI Press/The MIT Press, August 1996.
11. R. Schapire, P. Bartlett, Y. Freund, and W. Lee, "Boosting the Margin: A New Explanation for the Effectiveness of Voting Methods", *The Annals of Statistics*, vol. 25, no. 5, pp. 1651-1686, 1998.
12. J. Schlimmer and Jr. Granger, "Incremental Learning From Noisy Data", *Machine Learning*, vol. 1, pp. 317-354, 1986.