Managing Training Examples for Fast Learning of Classifiers Ranks

Boris Omelaenko
Metaintelligence Lab.
Kharkov State Technical University of Radioelectronics
Kharkov, Ukraine
boris@milab.kharkov.ua

Vagan Terziyan
Metaintelligence Lab.
Kharkov State Technical University of Radioelectronics
Kharkov, Ukraine
vagan@milab.kharkov.ua

Seppo Puuronen
Department of Computer Science and Information Systems
University of Jyvaskyla
Jyvaskyla, Finland
sepi@jytko.jyu.fi

Abstract

Paper deals with the problem of learning ranks of classifiers in ensembles. The problem of ordering of objects to classify is discussed. Two marginal approaches for learning, batch and incremental, with corresponding ordering strategies are analyzed. Presented algorithm lays between marginal methods, and it orders training examples by the deviation of classifiers opinions to match restrictions on learning time, cost and quality. Few aspects of this algorithm are experimentally investigated: classifiers ranks after learning, learning quality, ensemble accuracy and dependence between rank recalculation budget and ensemble accuracy. It was found, that descending order of examples provides fast rank learning with the best learning quality.

1. Introduction

Common way for combining multiple opinions of classifiers from an ensemble is to form one common opinion from the set of individual opinions. This common opinion can be individual opinion of one classifier or integrated. The usual way to select it is weighted voting of classifiers, where classifiers vote for their opinions with their ranks. This arises the problem of learning ranks of classifiers during processing the training set.

Machine learning techniques can be broadly categorized as either batch or incremental. Batch systems learn by examining a large collection of instances en masse and forming a single concept. Incremental systems evolve and change a classification scheme as new observations are processed [15]. Both kinds have their advantages and disadvantages.

Five advantages of using incremental rank changing, dealing with domains with hidden changes in context, were presented in [5]. These benefits arise from the idea to partition the domain and to use context-dependent order of processing these parts. The main disadvantage of incremental learning is dependency of learning results on the order of training examples. In [5] a form of cross-validation over time is used for local concept validation.

Few methods for eliminating dependency on voting order are included in well-known classification algorithms. Bootstrap Aggregation (Bagging) technique [2] selects *m* training examples randomly with replacement. These training examples are used to create *T* bootstrap samples and to generate *T* classifiers. Final classifier is built from the most precise classifiers. Random order just hides the problem of selecting correct order of objects.

Well-known Boosting algorithm (ADABOOST), presented in [3, 4] manipulates training examples, like bagging. ADABOOST algorithm uses a probability distribution over the training examples. On the *i*-th iteration it draws a training set of size *m* by sampling with replacement according to the probability distribution. Then it uses this training set to produce the *i*-th classifier. The order of learning is presented with the probability distribution and few advances are made to improve this distribution formula for better performance.

An attempt to compare experimentally both bagging and boosting algorithms is made in [13]. Experiments showed, that boosting algorithm usually works better, than bagging. On the other hand, boosting also produces severe degradation on some datasets. Author found the course of these fails in the voting scheme. Modified scheme gives

Proceedings of the Workshop on Computer Science and Information Technologies CSIT'99
Moscow, Russia, 1999

error rate approximately 3% less, than original scheme.

From the other hand, [3] presents larger experiments with the same algorithms. It was found, that bagging is much more competitive with boosting. Two important differences between their experiments and [13] can explain this discrepancy. At first, in [3] there was used 10 times greater number of tunes in their experiments. Second, another method for resampling of training data was used. Note, that both differences only change an order of processing data.

Very important problem of comparing techniques arises in this area. Learning and classification results strongly depend the input data. This makes it very difficult to compare techniques theoretically. And most researchers use experimental comparison of classification techniques to hide dependency on the input data in huge number of datasets.

Several phenomena of experimental comparison of classification techniques are discussed in [14]. Most papers use artificial datasets for comparison, as well as tuning algorithms to map those datasets. In [14] it was found that unproper selection of datasets can result in statistically invalid conclusions. It was recommended to use partitioning of datasets and to run a cross-validation to avoid pitfalls, suffered by many experimental studies.

Present paper continues the research, started in [6]. This paper deals with incremental learning and presents ranking technique for learning Allen temporal relations. The problem of voting strategy's influence on voting results was experimentally investigated in [11]. Three strategies, consequent, called as real-time strategy, iterative consequent, called as batch, and iterative parallel were investigated. The dynamics of voting results, ranks and quality of voting results showed, that voting process greatly depends on the selection of voting strategy. It was found, that we must carefully select the strategy to obtain good voting results. But no method for context-dependent selection of the strategy was presented there.

In the present paper we investigate the problem of selecting proper order of training examples for fast learning of classifiers ranks. We consider two basic strategies for ordering: batch and incremental and propose a new combined strategy, which matches several restriction on learning process.

In the second chapter of this paper we define basic concepts, used throughout the paper. Third chapter presents two basic marginal voting strategies and two iterative kinds of them. Our approach for combining marginal strategies and the algorithm is presented and discussed in the fourth chapter. Next chapter describes experiments hold and their results. We finish with conclusions and appendix with some diagrams.

2. Basic Concepts

2.1 Notation

We define the model for learning ranks of classifiers as the sixth-tuple <*D,C,O,L,L*,T*>, where:

$D = \{D_1, D_2, \ldots, D_d\}$ is the set of *d* training examples.

$C = \{C_1, C_2, \ldots, C_n\}$ is the set of *n* classifiers, which form the ensemble C. A numeral rank is assigned to each classifier. These ranks form the set $r = \{r_1, r_2, \ldots, r_n\}$ where each r_i is the rank of corresponding classifier C_i, $i = \overline{1,n}$.

$L = \{L_1, L_2, \ldots, L_l\}$ is the set of *l* possible labels, or classes, which can be assigned to training examples.

$L^* = \{L^*_1, L^*_2, \ldots, L^*_d\}$ is the set of real labels, already known for training examples.

The matrix $|O_{ij}|$ of classifiers opinions defines classes, assigned by individual classifiers. It is defined as follows:

$$O = \begin{vmatrix} O_{11} & O_{21} & \ldots & O_{d1} \\ O_{12} & O_{22} & \ldots & O_{d2} \\ \ldots & \ldots & & \ldots \\ O_{1n} & O_{2n} & \ldots & O_{dn} \end{vmatrix},$$

where $O_{ij}, O_{ij} \in L, i = \overline{1,d}, j = \overline{1,n}$ represents the class or label, assigned to the training example D_i by the classifier C_j.

To change classifier rank we need to know how «far» is it's opinion from the real class. Usually we can not measure the distance between classes, except some narrow cases. But we know exactly, whether classifier opinion equals to the real class or not. We use this information to create the matrix $|E|$ of classifiers errors, as follows:

$$|E| = \begin{vmatrix} e_{11} & e_{21} & \ldots & e_{d1} \\ e_{12} & e_{22} & \ldots & e_{d2} \\ \ldots & \ldots & & \ldots \\ e_{1n} & e_{2n} & \ldots & e_{dn} \end{vmatrix},$$

$$e_{ij} = \begin{cases} 1, \ if \ O_{ij} \neq L^*_i \\ 0, \ if \ O_{ij} = L^*_i \end{cases}, \quad i = \overline{1,d}, \quad j = \overline{1,n}$$

2.2 Evaluation of Learning Quality

The aim of classification process is to predict the class precisely. Quality of classification is usually measured with error rate of an ensemble. The aim of learning ranks is to assign ranks to classifiers, that will minimize bad

classifier's influence on ensemble opinion. This possibility will change during processing training examples, because processing of new examples will assign another ranks to classifiers. We define quality of learning ranks Q_i after processing the *i*-th training example as follows:

$$Q_i = \frac{1}{n}\sum_{j=1}^{n} r_j \cdot e_{ji}, \quad i = \overline{1,d}$$

Quality after processing the whole training set Q_d is the quality of learning, because these final ranks will then be used in classification. The aim of learning is to minimize parameter Q_i: $Q_i \rightarrow \min$.

2.3 Ranking

We use incremental learning, and consequent rank changing after processing a group of training examples. *Rank refinement strategy RS* defines the process of rank recalculation. The main formula used to refine the rank of each classifier after the ν-th group is the following:

$$r_i^{v+1} = r_i^v + \Delta r_i^v,$$

where the value of Δr_i^v (punishment or prize value), is equal to

$$\Delta r_i^v = \delta_i^v \cdot (\mu^v - e_i^v), \quad \mu^v = \frac{1}{n} \cdot \sum_j e_j^v,$$

The value δ_i^v depends the rank refinement strategy selected. We use the strategy «Leaders meet greater requirements than outsiders». The formula for δ_i^v under this strategy is as follows:

$$\delta_i^v = \begin{cases} (1 - r_i^v)^2, & if\ (\mu^v - e_i^v) \geq 0; \\ (r_i^v)^2, & if\ (\mu^v - e_i^v) < 0. \end{cases}$$

This formula forces rank leaders, who make an error, to be punished in rank more, than outsiders, who made the same error. This corresponds to the principle of greater responsibility for leaders.

3. Basic Voting Strategies

Voting strategy VS is the order, in which we must process training data. Voting strategy also defines the moments in which we must recalculate ranks of classifiers. Formally, we define voting strategy as the order:

$$VS = \langle D_i \ or \ R \mid D_i \in D \rangle$$

Symbol *R* in the above formula denotes the process of rank recalculation, conducted by the rank refinement strategy *RS*. Symbol D_i shows current training example. Appearance of this symbol forces the strategy to proceed classifiers opinions on this example and to calculate error rate for each classifier. Voting process is illustrated in Figure 1. In Figure 1 symbol *E* represents error rate of each classifier, other symbols are defined above. Voting strategy reorders classifiers opinions from the upper table to manage learning process, presented in the lower table.

Two basic voting strategies correspond to incremental and batch learning systems. *Consequent voting strategy* demands passing training examples one-by-one with rank recalculation just after every issue, as follows:

$$VS_{consequent} = \langle D_1, R, D_2, R, \ldots, D_d, R \rangle$$

This strategy will require *d* rank recalculations to proceed *d* examples. Each rank refinement uses classifiers error rates $|e|$, obtained on the previous vote.

Parallel voting strategy demands passing training examples one-by-one, as under the consequent strategy, but it requires only one rank recalculation after proceeding all examples:

$$VS_{parallel} = \langle D_1, D_2, \ldots, D_d, R \rangle$$

Parallel strategy summarizes error rates on all *d* votes, and uses this sum in rank recalculation:

$$|e| = \sum_{i=1}^{d} |e_i|$$

Both strategies can be iterative. *Iterative voting strategy* $VS*$ runs corresponding strategy few times to obtain better rank convergence. The formulas are the following:

$$VS^*_{consequent} = \langle VS^1_{consequent}, VS^2_{consequent}, \ldots, VS^k_{consequent} \rangle$$

$$VS*_{parallel} = \langle VS^1_{parallel}, VS^2_{parallel}, \ldots, VS^k_{parallel} \rangle$$

In the above formulas symbols *VS* represent processing training examples and running rank recalculations under the corresponding strategy.

The number of iterations *k* must be selected to correspond the context of classification process. The context may contain restrictions on classifiers ranks, number of rank recalculations, time of classification process, etc. One possible context, number of allowable rank recalculations, is investigated below.

4. Combined Voting Strategy

4.1 Few Ways to Combine Basic Strategies

There are few ways for combining marginal voting strategies. Let parameter *m* to denote the number of training examples to be processed without rank changing.

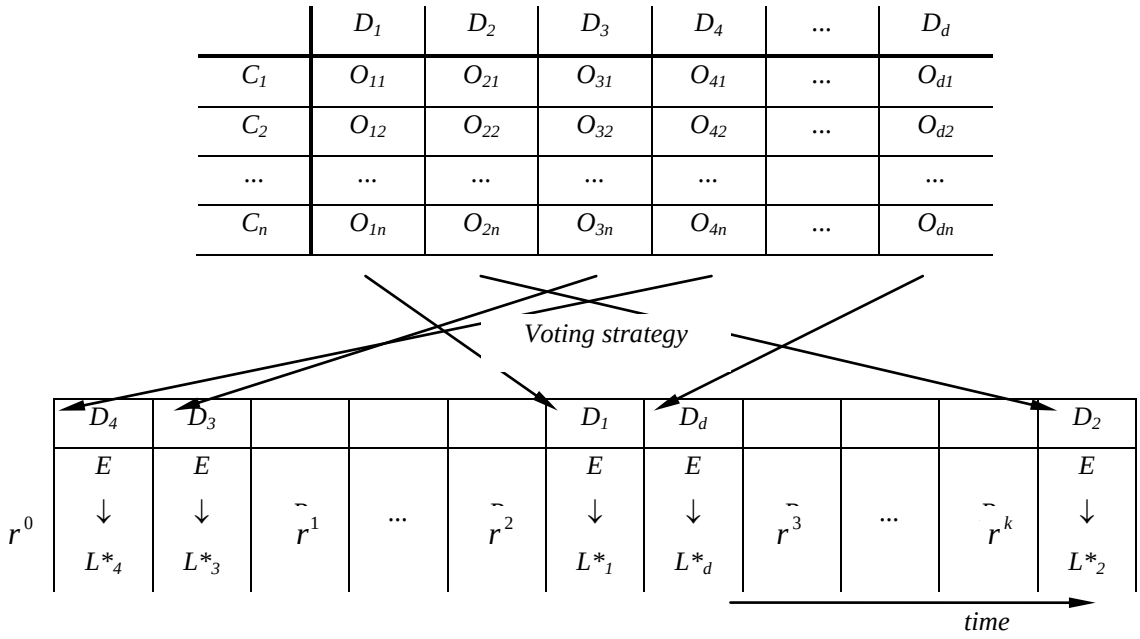

Figure 1. Voting process

Manipulating this parameter allows creating compromise strategies from consequent (m=1) to parallel (m=d), where d is the size of the training set. At least two factors influence selection of m. First factor presents restrictions on allowable number b of rank recalculations: $\frac{d}{m} \le b \;\Rightarrow\; m \ge \frac{d}{b}$, where b is rank recalculation budget. Second factor deals with dynamics of the domain and the training set. If time L is required to process one classifier opinion and the situation in the domain changes after time τ, then the following restriction on m arises: $\frac{d}{m} \cdot t \le \tau \;\Rightarrow\; m \ge \frac{dt}{\tau}$.

Another dimension of combining is in selecting parameter k for iterative strategies. Main restrictions for k are produced by rank recalculation budget and required learning quality.

4.2 Voting Strategy For Fast Rank Learning

We developed a combined strategy, that uses information about the diversity of classifiers opinions and recalculation budget to maximize speed of learning ranks. Classifiers error influence their ranks only after rank recalculation. For speed rank changing we must do rank recalculations as frequent as possible. We suppose that each rank recalculation takes computer time and has its cost. Hence, we must do rank recalculations as rear as possible to save money.

We order training examples according to diversity of classifiers opinions on them: ascending or descending. Ascending order lets classifiers change their ranks softly, starting with the less conflicting examples, that are easy to classify. Then they continue with more conflicting and more difficult examples. Descending order lets classifiers to start with the most difficult problem, without spending computer time of solving easy examples.

4.3 Partitioning method

The key idea for partitioning is to make summary error rates $|e|$ for every rank recalculation, as equal as possible. That forces each recalculation to do the same amount of rank changing and to judge the same amount of errors. This is the most efficient way of using limited number of recalculations. For this we partition the domain into parts with the same summary error of classifiers opinions.

The partitioning algorithm takes the training set D, matrix of classifiers errors $|E|$ and the number k of possible rank recalculations as input. Algorithm produces the partition $S_1, S_2, \ldots, S_k$, $\bigcup_{i=1}^{k} S_i = D$ of training examples. The expression $E(S)$ denotes mean error rate for classifier opinions on the examples from S.

The algorithm is presented in Figure 2. Mean error rate E^C of classifiers on every vote, used in the algorithm is defined as follows:

$$E_i^C = \frac{1}{n}\sum_{j=1}^{n} e_{ij},\ i = \overline{1,d}\ .$$

Maximal error rate of the part $E_{\max}$ is defined as follows:

Input: D, k, $|E|$.

Output: $S_1, S_2, ..., S_k, \bigcup_{i=1}^{k} S_i = D, S_i = \{D_k | D_k \in D\}$

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

1. Calculate the mean error rate of classifiers on every vote E_i^C.

2. Put domain objects in descending or ascending order of mean error rates, taken at step 1. This produces the order of domain objects $\langle o_1, o_2, ..., o_d \rangle$, where $o_i \in D, \bigcup_{i=1}^{d} o_i = D$ and $E_i^C > E_{i+1}^C, \quad i = \overline{1,d-1}$.

3. Calculate maximal error rate of the part $E_{\max}$.

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

5. I=I+1

6. **If** i>d **then goto** step 9.

7. **If** $E(S_j) < E_{\max}$ **then** J=J+1 **Else** Add object o_i to the part S_j: $S_j = S_j \cup o_i$.

8. **Goto** step 5

9. End

Figure 2. The algorithm for partitioning

$$E_{\max} = \frac{1}{k} \cdot \sum_{i=1}^{d} E_i^C .$$

This method orders training examples by deviation of classifiers opinions. If we will consider lower deviation as a sign of higher ensemble competence, then we will find similar ideas in [1]. In this work classifiers in the ensemble are grouped into an ordered list by classifier's competence, presented with corresponding threshold. This order can be presented with the order of classifier's ranks after learning. Unfortunately, in [1] a special domain of recognizing images of Venus is used, and we can not compare their algorithm with the present one.

The background of ascending order of training examples is to give easy tasks to experts (classifiers) first, and then continue with hard problems. The examples with minimal deviation of classifiers opinions seems to be the easiest. The background of descending order is the opposite: to give experts hard task first, and then continue with easy problems. We suppose, that tasks with diverse classifiers opinions are more difficult.

5. Experimental comparison

We used IRIS data set from the UCI Machine learning repository of databases [10] for experiments. This data set has 150 examples with four numeral attributes and three classes. The ensemble was constructed with five well-known classifiers: ID3, MC4, Decision Table with majority votes (Table majority), Decision table with simple votes (Table no majority) and Naive-Bayes. ID3 and MC4 are well described in [12], both decision table methods are presented in [7], and Naive-Bayes is investigated in [9].

Well-known MLC++ machine learning library [8] was used to produce opinions of individual classifiers. 12 examples from the data set were used to train individual classifiers, and the rest — to train the ensemble.

Rank dynamics for classifiers under the strategy with descending order is presented in Figure 3 and Figure 4 for rank recalculation budget of 5 and 20 recalculation respectively. Rank dynamics for ascending order strategy is presented in Figure 5 and Figure 6 for budget of 5 and 20. In these figures X-axis represent training examples from 1 to 140, Y-axis represent rank of each classifier on every example. Quality evaluations and mean classification error are marked on this axis, too.

Experimental results are presented in Table 1. Column Quality contains the final learning quality, Budget column contains corresponding number of rank recalculations, column Rank leaders contains classifiers, whose final ranks are more than 0.6. Column Rank outsiders contains

classifiers with final ranks less than 0.3. Mean ranks classifiers have final ranks between 0.3 and 0.6.

Table 1 shows that increasing of rank recalculation budget generally changes the final order of classifiers by ranks. Five rank recalculations are not sufficient for this data set, but 10 or more recalculations produce very similar results. This forms the first experimental result: both strategies provide similar rank ordering of classifiers on the end of learning process.

Quality of learning process is few times better under descending order of examples, than under ascending or random order. Descending order provides quality about 0.05, ascending order – 0.35, which is 7 times worse. Even random order is better, than ascending, with the quality of 0.24. The second experimental result is: descending order provides 7 times better quality, than ascending order, and 5 times better quality, than random order.

Final ensemble will include all these classifiers, combined with voting. We investigated the accuracy of this ensemble with ranks assigned during learning process. Note, that learning quality deals only with rank and errors of each classifier, while accuracy deals with error rate of the whole ensemble. We expect, that accuracy will be higher, than quality, because poor classifiers will influence quality, while their opinions will be ignored due to selection made by voting.

Accuracy of individual classifiers on IRIS data set is presented in Table 3. Decision Table classifiers have extremely poor accuracy, that's why they were punished under all orders and budgets.

Accuracy of the ensemble with simple non-weighted

Table 1. Experimental results: learning ranks

Quality	Budget	Rank leaders	Mean ranks	Rank outsiders
Descending order				
0.0688	5	MC4, ID3, NB	TM, TNM	
0.0533	10	ID3, NB, MC4	TM	TNM
0.0449	15	ID3, NB, MC4	TM	TNM
0.0401	20	ID3, NB	MC4, TM	TNM
Ascending order				
0.3871	5	MC4, ID3	NB, TM, TNM	
0.3677	10	MC4, ID3	NB	TM, TNM
0.3427	15	ID3, MC4	NB	TM, TNM
0.3361	20	MC4, ID3	NB	TM, TNM
Random order				
0.2415	20	ID3, MC4, NB	TM	TNM

Table 2. Experimental results: ensemble accuracy

Number of rank recalculations	Ensemble accuracy under the order:				
	Final, under the order			Change, under the order	
	ascending	descending	random	ascending	descending
5	0.8913	0.8043	0.8768	—	—
10	0.8913	0.8913	0.8406	0.0000	0.0870
20	0.9420	0.9203	0.8261	0.0507	0.0290
40	0.9203	0.9493	0.8913	-0.0217	0.0290
80	0.9638	0.9710	0.9058	0.0435	0.0217

voting of classifiers, on the data set is equal to 0.7174. The ensemble, constructed with non-weighted voting produces poor results, than most it's classifiers. Table 2 presents accuracy of the ensemble with ranked majority voting. This ensemble accuracy is greatly better, than any individual accuracy due to rank refinement. This is the third, expected experimental result: developed algorithm forms ranks, which increase accuracy of the ensemble.

Table 3. Classifier's accuracy

Classifier	Accuracy
ID3	0.8841
MC4	0.8551
Table Majority	0.3188
Table no Majority	0.0072
Naive Bayes	0.8406

It is shown, that doubling recalculation budget is required to keep linear accuracy growth under descending order. Ascending and random orders provide not evident accuracy changing while doubling budget. This makes the fourth experimental result: only descending order provides simple accuracy dependence on recalculation budget, double budget will linearly (mostly constantly) increase the accuracy.

6. Conclusions

We conclude with the following results:

- The problem of managing the order of training examples is formulated. Existent well-known classification algorithms manipulates this order, and authors use to change ordering to improve performance of their algorithms, without considering ordering problem separately. Two basic marginal voting strategies: consequent and parallel have limited applications. But we can combine them to create flexible strategies.

- Few directions for combining were given on the basis of possible restrictions on learning process: required learning quality, allowable cost, etc. A method for creating combined strategy to match demands on learning cost and learning speed was developed and experimentally investigated.

Experiments forced us to the following experimental results.

- Both strategies provide similar rank ordering of classifiers on the end of learning process.

- Descending order provides 7 times better quality, than ascending order, and 5 times better quality, than random order.

- Developed algorithm forms ranks, which increase accuracy of the ensemble.

- Only descending order provides simple accuracy dependence on recalculation budget, double budget will linearly (mostly constantly) increase the accuracy. We must prefer descending order of training examples to get fast and high-quality rank learning.

The order of training examples is not managed inside the intervals with the same mean ensemble error. But such a higher-level management is needed. Also recommendations for enhancing cross-validation and other standard ways of ordering must be developed and experimentally investigated in the future.

7. References

1. Asker L. and Maclin R. "Ensembles as a Sequence of Classifiers". In: *Proceedings of the International Joint Conference on Artificial Intelligence IJCAI-97*, Nagoya, Aichi, Japan, August 23-29, 1997.

2. Breiman L. "Bagging Predictors". *Machine Learning* 24, 1996, pp. 123-140.

3. Freund Y. and Schapire R. E. "Experiments with a new boosting algorithm". In: *Proceedings of the Thirteenth International Conference on Machine Learning*, Bari, Italy, 1996, July 3-6.

4. Freund Y., Iyer R., Schapire R. E. and Singer Y. "En Efficient Algorithm for Combining Preferences". In: *Proceedings of the Fifteenth International Conference on Machine Learning*, Madison, Wisconsin USA, 1998, July 24-27.

5. Harries M. and Horn K. "Learning stable concepts in domains with hidden changes in context". In: *Proceedings of the Thirteenth International Conference on Machine Learning, Workshop on Learning in Context Sensitive Domains*, Bari, Italy, 1996, July 3-6.

6. Kaikova H. and Terziyan V. "Temporal Knowledge Acquisition From Multiple Experts". In: Shoval P. & Silberschatz A. (Eds.), *Proceedings of NGITS'97 - The Third International Workshop on Next Generation Information Technologies and Systems*, Neve Ilan, Israel, 1997, June 30 - July 3, pp. 44 - 55.

7. Kohavi R. "The power of decision tables". In: N. Lavrac & S. Wrobel, eds, *Proceedings of the European Conference on Machine Learning*, Lecture Notes in Artificial Intelligence 914, Springer Verlag, Berlin, Heidelberg, New York, 1995, pp. 174-189.

8. Kohavi R., Sommerfield D. and Dougherty J. "Data Mining using MLC++: A machine learning library in C++". *International Journal on Artificial Intelligence Tools* 6(4), 1997, pp. 537-566.

9. Langley P., Iba W. and Thompson K. "An Analysis of Bayesian Classifiers". In: *Proceedings of the Tenth National Conference on Artificial Intelligence*, AAAI Press/MIT Press, 1992, pp. 223-228.

10. Merz C. and Murphy P. "UCI Repository of Machine Learning Databases". http://www.ics.uci.edu/~mlearn/MLRepository.html, 1998.

11. Omelaenko B., Terziyan V. and Puuronen S. "Multiple Expert Knowledge Acquisition: Experimental Investigation of Three Voting Strategies". In: *STeP'98 Human and Artificial Information Processing*, Finnish Conference on Artificial Intelligence, 7-9 September, Jyvaskyla, Finland, Publ. of the Finnish AI Society, 1998.

12. Quinlan J. R. "C4.5: Programs for Machine Learning". Morgan Kaufmann Publishers, Los Altos, California, 1993.

13. Quinlan J. R. "Bagging, Boosting, and C4.5". In: *The Thirteenth National Conference on Artificial Intelligence AAAI'96, August 4-8, Portland, Oregon*, AAAI Press/The MIT Press, 1996.

14. Salzberg S. "On Comparing Classifiers: Pitfalls to Avoid and a Recommended Approach". *Data Mining and Knowledge Discovery* 1, 1997, p. 317-328.

15. Schlimmer J. and Granger Jr. R. "Incremental Learning From Noisy Data". *Machine Learning* 1, 1986, pp. 317-354.

Appendix

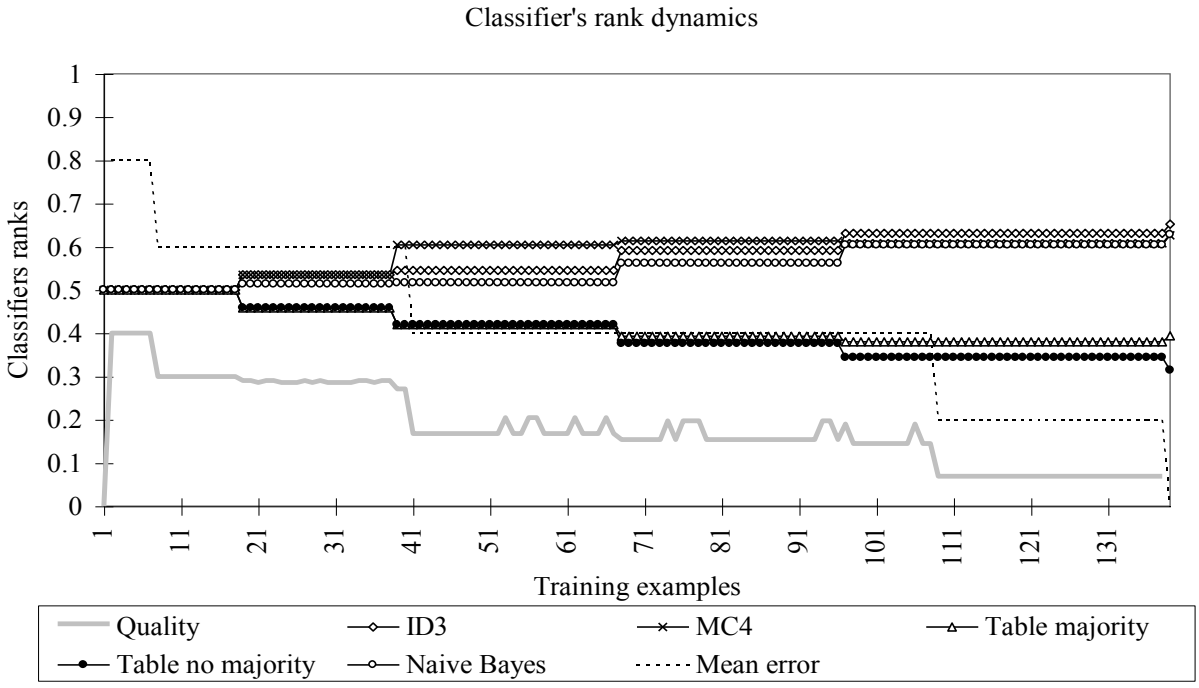

Figure 3. Rank dynamics under descending order, 5 recalculations

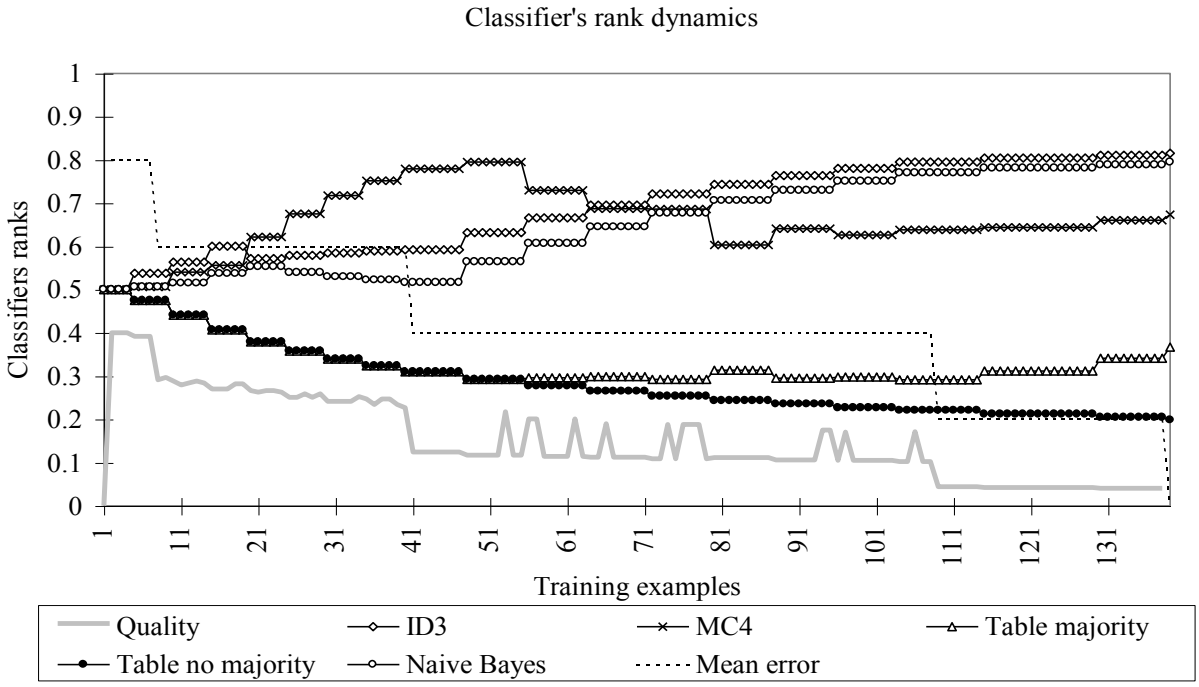

Figure 4. Rank dynamics under descending order, 20 recalculations

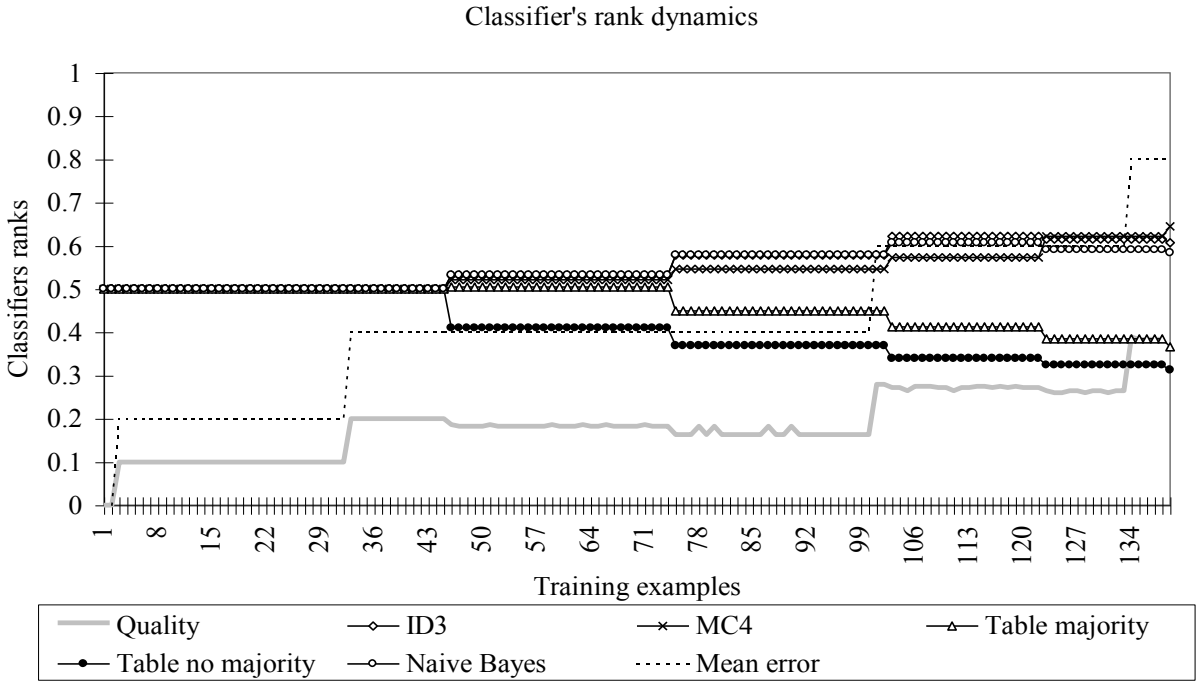

Figure 5. Rank dynamics under ascending order, 5 recalculations

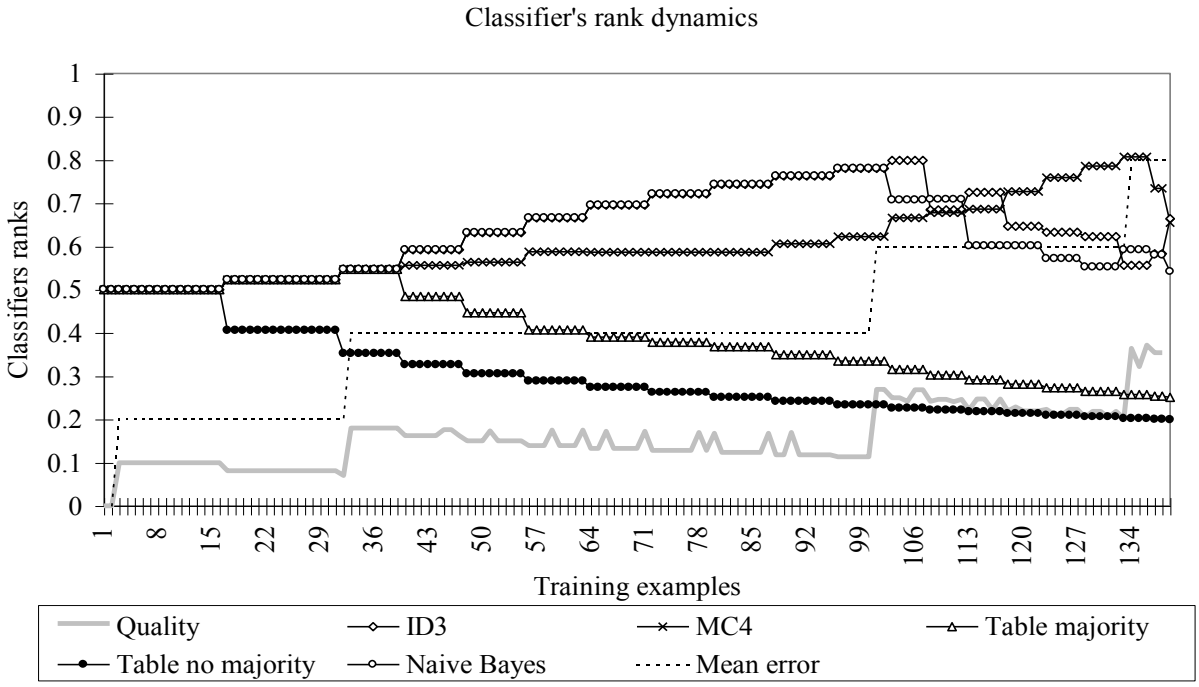

Figure 6. Rank dynamics under ascending order, 20 recalculations

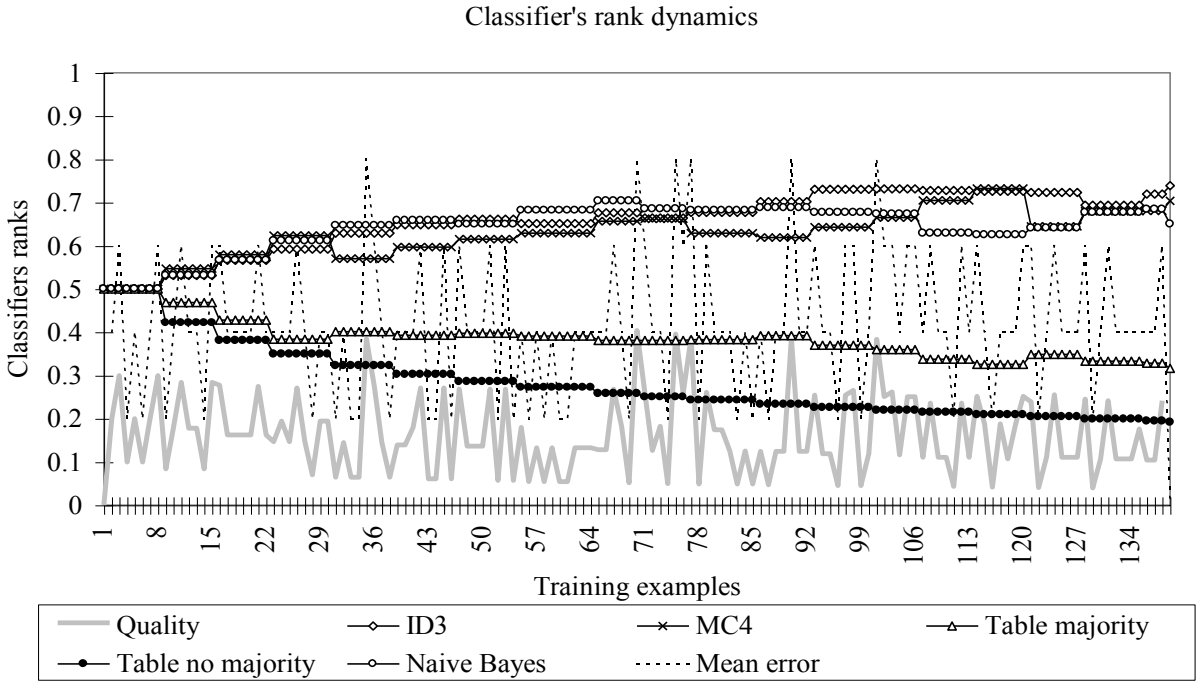

Figure 7. Rank dynamics under random order, 20 recalculations