Computing and Informatics, Vol. 32, 2013, 661–677

PARALLEL CLASSIFICATION WITH TWO-STAGE BAGGING CLASSIFIERS

Verena Christina Horak*

Department of Computer Sciences, University of Salzburg
5020 Salzburg, Austria
e-mail: `vhorak@cosy.sbg.ac.at`

Tobias Berka*

Computer Laboratory, University of Cambridge
United Kingdom
&
Department of Computer Sciences, University of Salzburg
5020 Salzburg, Austria
e-mail: `tberka@cosy.sbg.ac.at`

Marian Vajteršic

Department of Computer Sciences, University of Salzburg
5020 Salzburg, Austria
&
Mathematical Institute, Slovak Academy of Sciences
840 00 Bratislava, Slovakia
e-mail: `marian@cosy.sbg.ac.at`

Abstract. Bootstrapped aggregation of classifiers, also referred to as bagging, is a classic meta-classification algorithm. We extend it to a two-stage architecture consisting of an initial voting amongst one-versus-all classifiers or single-class recognizers, and a second stage of one-versus-one classifiers or two-class discriminators used for disambiguation. Since our method constructs an ensemble of elementary classifiers, it lends itself very well to parallelization. We describe a static work-

* These co-authors contributed equally to this work.

load balancing strategy for embarrassingly parallel classifier construction as well as a parallelization of the classification process with the message passing interface. In our experiments, which are evaluated in terms of classification performance and speed-up, we obtained an up to three-fold increase in precision and significantly increased recall values.

Keywords: Classification methods, bagging classifiers, parallel algorithms

1 INTRODUCTION

In machine learning, 'classification' denotes the problem of deriving, applying and evaluating a function or more generally an algorithm – which will be called a *classifier* – that categorizes yet unlabeled data according to information provided by already categorized data. A typical example is blood typing, for which several medical parameters are chosen to determine the blood type of a not yet categorized blood sample. In general, classification consists of two steps: the construction of classifiers for which given labeled data is analyzed and the application of these classifiers on unlabeled data.

Construction of classifiers using labeled training examples is an established supervised learning task. Given a pattern, typically represented as a vector, we seek to predict the class of that pattern using a classifier.

There is a broad range of classification models, methods and algorithms to be used in a wide range of circumstances (cf. [1]). Bootstrapping as a population resampling technique applied to clustering has been introduced as "bagging", short for **b**ootstrap **agg**regation, in [2]. One highly popular extension is "boosting" [3], where the selection of individual items is not purely random but is directed to maximize the classification accuracy according to a gradient descent.

Both bagging and boosting have successfully been parallelized [4]. The authors report that bagging, quite naturally, achieves nearly linear speed-up in the number of processes with only a marginal overhead. Boosting has been adapted to MapReduce for cloud computing and other, more relaxed parallel environments in [5]. Resampling methods are also used for the construction of clustering ensembles to deal with the high computational complexity, e.g. in the CLARA and CLARANS algorithms [6].

In this paper, we present a meta-classification method, which in turn uses other elementary classification methods in order to alleviate this problem. It is motivated by two ideas from the fields of machine learning and statistics: one-versus-all classification [7] and the statistical method of bootstrapping [8] applied to the construction of classifiers in an approach called bagging. A classifier construction algorithm is executed on several sampled subsets of the training information to obtain a set of different classification criteria. The classification is performed separately on each of these criteria and combined by voting to form an overall result.

We describe its application on an adaption of a decision rule construction algorithm called SCALLOP [9, 10]. This classifier was originally developed for data streams, operating on large amounts of low-dimensional information. In our experimental evaluation, we demonstrate that we can use our meta-classifier algorithm to successfully apply SCALLOP to relatively small, high-dimensional data sets.

Furthermore, our method is very well suited for parallelization, and we introduce and evaluate two simple and intuitive parallelization strategies.

Hence, the paper is structured as follows: The main idea of our approach is introduced in the second section and parallelized in the third section. After the evaluation in the fourth section, we give a short summary and conclude with the future work.

2 A TWO-STAGE BOOTSTRAPPING APPROACH USING TWO-CLASS CLASSIFIERS

Bootstrapping is a resampling method, whereby only a small part of a sample is selected at random to generate a new, smaller population of samples. The naive application of bootstrapping to classification is to construct an independent classifier for every resampled population, which can "independently" predict the class label for any new object. The final prediction can then be obtained by voting. We have combined this statistical technique with the "one-versus-all" approach to classifier construction. Instead of building a single classifier capable of predicting every class, we construct a set of classifiers. Every one of these classifiers is a two-class discriminator, and it distinguishes between the instances of a single class and *all other* classes (hence the name one-versus-all classifier). In order to break ties in the voting process, we also observe the "all-versus-all" approach to classifier construction. Here, we consider all pairs of classes and ignore the patterns of all other classes.

For our purposes, every pattern can be written as a vector in $\mathbb{R}^n$. Let us assume that all vectors are normalized and shifted to fit into $[0,1]^n$, for simplicity. We have C classes which are identified as labels. These class labels are elements of the set $\{0,\dots,C-1\}$.

To construct classifiers, we require labeled training data. Let X be a given set of N' normalized patterns, meaning that $X \subset [0,1]^n$ and $|X| = N'$. These N' labeled "training vectors" can (but need not) be contained within the set of N patterns to be classified.

The elements of X are labeled with a function $l : X \to \{0,\dots,C-1\}$, which maps every individual training pattern x from the set X to the known class membership $l(x)$.

As a convenient shorthand notation, let us denote by $X^{(c)}$ selected training instances for an individual class $c \in \{0,1,\dots,C-1\}$, formally $X^{(c)} = \{x \in X \mid l(x) = c\}$.

For statistical reasons, we require that each class is represented approximately equally often, which means that the cardinality of $X^{(c)}$ corresponds to approximately

N'/C for each $c \in \{0, \dots, C-1\}$.

A classifier π for predicting the class membership is a function $\pi : [0,1]^n \to \{0, \dots, C-1\}$. Based on the labeled training instances, we can construct a classifier with a learning (or training) algorithm $\mathcal{C}$, which maps the set of training patterns X and the labeling function l to a classifier $\pi = \mathcal{C}(X, l)$.

The first step is to build one-versus-all training sets. For each class c, we relabel the patterns in X by assigning 0 if the label corresponds to c or 1 otherwise. Formally, we construct a new labeling function $l_c : X \to \{0,1\}$ with the assignment rule $l_c(x) = 0$ if $x \in X^{(c)}$ or 1 otherwise.

In the second step, we consider all-versus-all classification [7]. As before, new data sets are generated by relabeling elements of X. This time, we build new subsets for every pair of classes (i,j), where $0 \leq i < j \leq C-1$, containing all patterns with labels i and j, formally $X_{(i,j)} = X^{(i)} \cup X^{(j)}$. These sets are relabeled to 0 or 1 with labeling functions $l_{(i,j)} : X_{(i,j)} \to \{0,1\}$, where $l_{(i,j)}(x) = 0$ if $x \in X^{(i)}$ and 1 if $x \in X^{(j)}$. Hence we receive $C(C-1)/2$ new training sets with approximately $2N'/C$ elements each due to the requirement that $|X^{(c)}| \approx N'/C$ for each class c. Although the name "all-versus-all" classification is common in the literature, we prefer to speak of "one-versus-one" sets, since we will only inspect individual sets during the voting process.

The third step of our algorithm is called "selection for iteration". For large data sets and a small number of classes it would be fine to work with the training sets X_i and $X_{(i,j)}$ described above. However, for small data sets this is not the case. Therefore, we introduce an iterative process which generates additional subsets by bootstrapping. We thereby receive iteration subsets X_i^t or $X_{(i,j)}^t$ of size $2s$, where s is smaller than or equals N'/C, i.e. the approximate cardinality of each $X^{(c)}$, and $0 \leq t \leq I-1$ for I iterations. However, since this is a bootstrapping method, s should be somewhat smaller than the total number of patterns of one class. Formally, we define an operator $\mathcal{S}$ which randomly selects s elements of a set X and returns them as a resampled set $\mathcal{S}(X, s)$. In special cases where further information on the vectors to be classified is given it may be useful to choose different numbers of iterations, i.e. $I_1 \neq I_2$, for the generation and application of one-versus-all and one-versus-one classifiers. Although the underlying (given) classifier is exactly the same for both one-versus-all and one-versus-one classifiers, it is common to speak of recognizers and discriminators, which will be denoted by φ and ψ, respectively. We have summarized the key notation, and our global variables, in Table 1.

Algorithm 1 gives an accurate description of the general case with the same number of iterations for recognizers and discriminators.

To predict the class membership of new patterns, a voting algorithm is used to combine the outputs of all of the individual classifiers.

First, only the recognizers are taken into account, which is illustrated in Figure 1. If the result is unambiguous, the estimated label is assigned to the pattern straight away. Otherwise, the second part of the voting algorithm, which is visualized in Figure 2, is applied.

Algorithm 1: Bootstrapped Generation of Classifiers

```
Input: The set of labeled training patterns $X$, and the number of classes $C$
       and iterations $I$.
Result: $IC$ recognizers $\varphi_0^t, \dots, \varphi_{C-1}^t$ and $IC(C-1)/2$ discriminators $\psi_{(i,j)}^t$
        for $t \in \{0, \dots, I-1\}$ and $0 \leq i < j \leq C-1$.
for $t \leftarrow 0$ to $I-1$ do
    for $i \leftarrow 0$ to $C-1$ do
        $T \leftarrow \mathcal{S}(X^{(i)}, s)$; $\hat{T} \leftarrow \mathcal{S}(X \setminus X^{(i)}, s)$;
        let $l(u) = \begin{cases} 0, & \text{if } u \in T \\ 1, & \text{if } u \in \hat{T} \end{cases}$ ;
        $\varphi_i^t \leftarrow \mathcal{C}(T \cup \hat{T}, l)$;
        for $j \leftarrow i+1$ to $C-1$ do
            $T \leftarrow \mathcal{S}(X^{(i)}, s)$; $\hat{T} \leftarrow \mathcal{S}(X^{(j)}, s)$;
            let $l(u) = \begin{cases} 0, & \text{if } u \in T \\ 1, & \text{if } u \in \hat{T} \end{cases}$ ;
            $\psi_{(i,j)}^t \leftarrow \mathcal{C}(T \cup \hat{T}, l)$;
```

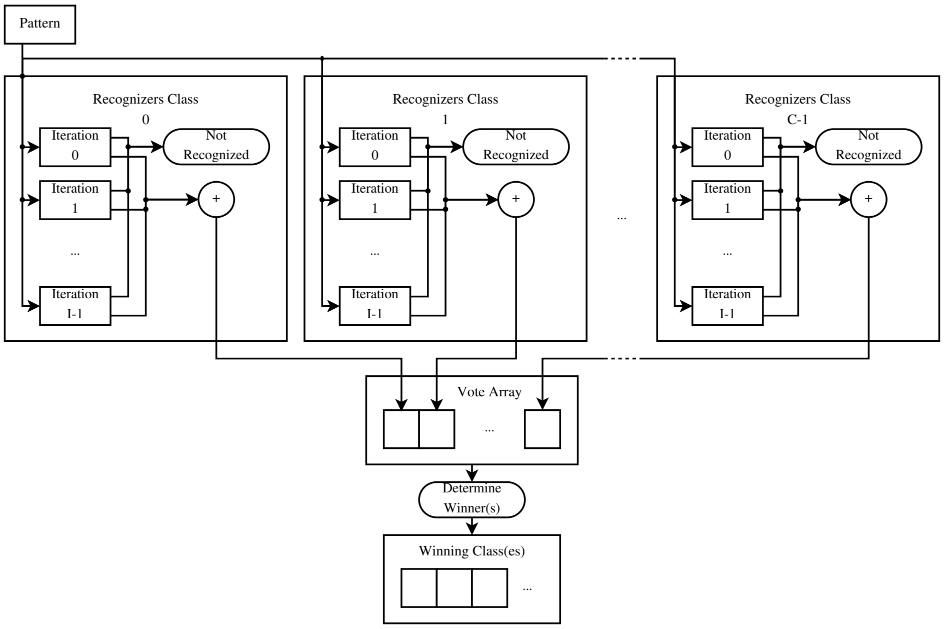


Figure 1. Voting Stage 1

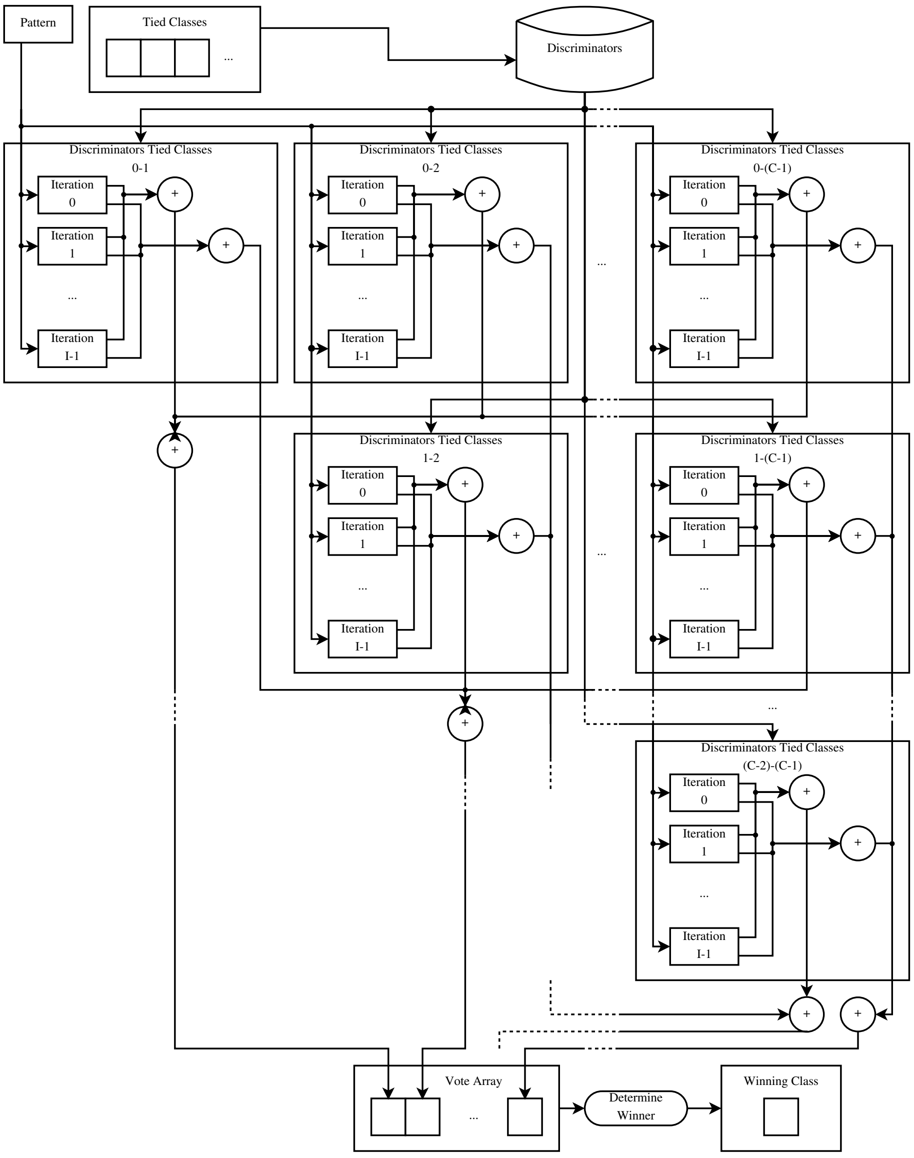

Pattern
Tied Classes
...
Discriminators
Discriminators Tied Classes 0-1
Discriminators Tied Classes 0-2
Discriminators Tied Classes 0-(C-1)
Discriminators Tied Classes 1-2
Discriminators Tied Classes 1-(C-1)
Discriminators Tied Classes (C-2)-(C-1)
Iteration 0
Iteration 1
Iteration I-1
+
Vote Array
Determine Winner
Winning Class

Figure 2. Voting Stage 2

Symbol	**Usage**
n	Number of variables or features
N	Number of instances or patterns to be classified
N'	Number of labeled training instances or patterns
X	Set of all training vectors
C	Number of classes
$X^{(c)}$	Set of pattern vectors with label c
l	Labeling function
s	Sample size
I	Number of iterations
π	Classifier
$\mathcal{C}$	Classifier construction operator
$\mathcal{S}$	Resampling operator
φ_i^t	Recognizer for class i from iteration t
$\psi_{(i,j)}^t$	Discriminator for classes i and j from iteration t

Table 1. Notation and global environment

The second stage of the voting uses the discriminators $\psi_{(i,j)}^t$ to distinguish between the two most promising classes. In the case that this comparison does not deliver a unique result, we randomly assign either label i or j to the actual pattern – which has never turned out to be necessary in our tests. The combination of these two voting stages is shown in Algorithm 2.

3 PARALLELIZATION

We have used the message passing interface (MPI) as middleware for our parallel implementation [11]. It is an appealing choice for our algorithm because it offers high-performance implementations for broadcasts and reductions such as parallel vector summation. Each of the P individual parallel processes is identified by a number $r \in \mathbb{N}, 0 \leq r \leq P - 1$. Table 2 summarizes the additional notation.

The computationally expensive part of our algorithm is the construction of the individual two-class discriminators. Fortunately, our approach allows us to construct all classifiers individually and in an embarrassingly parallel manner. We simply resample the data set and conduct a number of independent executions of the classifier construction algorithm.

However, since the structure of the classifier architecture is highly regular, we can devise a systematic way to construct all the individual classifiers for C classes and P processes.

For the one-versus-all classifiers, we have to distribute a total of $I\,C$ classifier construction tasks amongst P processes. This means that every individual process has to build approximately $I\,C/P$ recognizers which results in an uneven distribution of tasks if $(I\,C)\%P \neq 0$ (% denotes the modulo operator). To balance the load evenly, we split our processes into two groups: the processes of the first group

Algorithm 2: Two-Stage Voting

```
Input: A pattern vector $x$.
Result: An estimated label for $x$.
/* First stage (one-versus-all classification)                    */
for $t \leftarrow 0$ to $I-1$ do
    for $i \leftarrow 0$ to $C-1$ do
        if $\varphi_i^t(x) = 0$ then vote[i] ← vote[i] + 1;
        ;
$M \leftarrow \operatorname{argmax}_{0 \le i < C} \text{vote}[i]$;
if $|M| = 1$ then
    return $m \in M$;
/* Second stage (one-versus-one classification)                   */
else
    for $(i,j) \in M \times M$, $i < j$ do
        for $t \leftarrow 0$ to $I-1$ do
            if $\psi_{(i,j)}^t(x) = 0$ then
                vote[i] ← vote[i] + 1;
            else if $\psi_{(i,j)}^t(x) = 1$ then
                vote[j] ← vote[j] + 1;
    $M \leftarrow \operatorname{argmax}_{0 \le i < C} \text{vote}[i]$;
    if $|M| = 1$ then return $m \in M$;
    ;
    else
        return random element of $M$;
```

construct $\lfloor (I\,C)/P \rfloor + 1$ recognizers and those from the second group build one less. Using these workload sizes, we can compute the first and last classifier construction task for every process. The procedure *Responsibilities* computes the local classifier tasks, and derives the concise classes and corresponding numbers of iterations.

For the one-versus-all classifiers, we have to extend the approach to distribute all $I\,C\,(C-1)/2$ individual classifiers amongst our P processes. We can again use the *Responsibilities* procedure to allocate a total of $C\,(C-1)/2$ artificial classes, but we need to decode these into pairs of classes for the discriminators. The procedure *Decode* implements this conversion.

Algorithm 3 uses the procedures *Responsibilities* and *Decode* to construct the two sets Φ_r and Ψ_r that contain all one-versus-all and one-versus-one classifiers for the process identified by r. The sampling operator $\mathcal{S}$ and classifier construction

Symbol	Usage
P	Number of processes
r	Numerical identifier for a process r
Φ_r	All recognizer for a process r
Ψ_r	All discriminators for a process r

Table 2. Notation and global environment

```
Algorithm 3: Parallel Classifier Construction
Input: The set of all training patterns $X$ and the sample size $s$.
Result: Returns the sets of recognizers $\Phi_r$ and discriminators $\Psi_r$ for each
        local process $r$.
parallel for $r \in \{0, \dots, P-1\}$ do
    $\Phi_r \leftarrow \emptyset$; $\Psi_r \leftarrow \emptyset$;
    $R = \text{Responsibilities}(C, I, P, r)$;
    for $(c, i) \in R$ do
        do $i$ times
            $T \leftarrow \mathcal{S}(X^{(c)}, s)$; $\hat{T} \leftarrow \mathcal{S}(X \setminus X^{(c)}, s)$;
            let $l(u) = \begin{cases} 0, & \text{if } u \in T \\ 1, & \text{if } u \in \hat{T} \end{cases}$;
            $\Phi_r \leftarrow \Phi_r \cup \{\mathcal{C}(T \cup \hat{T}, l)\}$;
    $R = \text{Decode}(\text{Responsibilities}(\frac{C(C-1)}{2}, I, P, r), C)$;
    for $(c, c', i) \in R$ do
        do $i$ times
            $T \leftarrow \mathcal{S}(X^{(c)}, s)$; $\hat{T} \leftarrow \mathcal{S}(X^{(c')}, s)$;
            let $l(u) = \begin{cases} 0, & \text{if } u \in T \\ 1, & \text{if } u \in \hat{T} \end{cases}$;
            $\Psi_r \leftarrow \Psi_r \cup \{\mathcal{C}(T \cup \hat{T}, l)\}$;
    return $\Phi_r$, $\Psi_r$;
```

algorithm $\mathcal{C}$ are an abstract notation for any suitable implementation of these two processes.

The construction of the classifiers can be parallelized with little communication effort. Only a single broadcast at the beginning is required – or even without any explicit communication if the training set can be simultaneously read from a parallel file system.

Unfortunately, the voting part cannot be parallelized so easily. Assume that we have set up the system and all classifiers are available according to the responsibilities outlined above. The voting can be parallelized as described in Algorithm 4 in which the iteration index t of all recognizers and discriminators is omitted for readability

```
Procedure Responsibilities
  Input: The number of (pairs of) classes $C$, of iterations $I$, of processes $P$,
         and the local process identified by $r$.
  Result: A set $R$ with pairs $(c_j, i_j)$ specifying the classes and number of
          iterations for the process identified by $r$.
  $U = C\,I$; $R \leftarrow \emptyset$;
  if $r > U\%P$ then
      $p_0 \leftarrow (\frac{U}{P}+1)(U\%P) + \frac{U}{P}(r - U\%P)$;
  else
      $p_0 \leftarrow (\frac{U}{P}+1)r$;
  if $(r+1) > U\%P$ then
      $p_1 \leftarrow (\frac{U}{P}+1)(U\%P) + \frac{U}{P}(r+1-U\%P) - 1$;
  else
      $p_1 \leftarrow (\frac{U}{P}+1)(r+1) - 1$;
  $c_0 \leftarrow \lfloor\frac{p_0}{I}\rfloor$; $c_1 \leftarrow \lfloor\frac{p_1}{I}\rfloor$;
  if $c_0 = c_1$ then
      $R \leftarrow R \cup \{(c_0, p_1 - p_0 + 1)\}$;
  else
      $R \leftarrow R \cup \{(c_0, I - p_0\%I)\}$;
      for $k \leftarrow c_0 + 1$ to $c_1 - 1$ do
          $R \leftarrow R \cup \{(k, I)\}$;
      $R \leftarrow R \cup \{(c_1, p_1\%I + 1)\}$;
  return $R$;
```

```
Procedure Decode
  Input: A set $R$ of encoded pairs $(c, i)$ of encoded classes and iterations, and
         the total number of classes $C$.
  Result: A set $R'$ containing decoded triples $(c, c', i)$ of pairs of classes $(c, c')$
          and iterations.
  $R' \leftarrow \emptyset$;
  for $(c, i) \in R$ do
      $j \leftarrow 0$;
      for $k \leftarrow C - 1$ downto 0 do
          if $c < k$ then
              $R' \leftarrow R' \cup \{(j, c+j+1, i)\}$;
              break;
          $c \leftarrow c - k$;
          $j \leftarrow j + 1$;
  return $R'$;
```

purposes. As we can see, MPI's collective communication operations, specifically MPI_Reduce with a sum operator, allows us to avoid point-to-point communication primitives. This greatly simplifies the algorithm and leads to a more efficient implementation.

Algorithm 4: Parallel Voting

```
Input: A pattern vector $x$.
Result: An estimated label for $x$.

parallel for $r \in \{0, \dots, P-1\}$ do
    for $\varphi_i \in \Phi_r$ do
        if $\varphi_i(x) = 0$ then $\text{vote}_r[\text{i}] \leftarrow \text{vote}_r[\text{i}] + 1$;
        ;
    reduce all-to-all : $\text{vote[i]} \leftarrow \sum_{r=0}^{R-1} \text{vote}_r[\text{i}]$ ;
    ;
    $M \leftarrow \underset{0 \leq i < C}{\text{argmax}}\ \text{vote}[i]$;
    if $|M| = 1$ then
        return $m \in M$;
    else
        for $(i, j) \in M \times M,\ i < j$ do
            for $\psi_{(i,j)} \in \Psi_r$ do
                if $\psi_{(i,j)}(x) = 0$ then
                    $\text{vote}_r[\text{i}] \leftarrow \text{vote}_r[\text{i}] + 1$;
                else if $\psi_{(i,j)}(x) = 1$ then
                    $\text{vote}_r[\text{j}] \leftarrow \text{vote}_r[\text{j}] + 1$;
    reduce $(0 : P-1) \rightarrow 0$ : $\text{vote[i]} \leftarrow \sum_{r=0}^{R-1} \text{vote}_r[\text{i}]$ ;
    ;
    if $r = 0$ then
        return disambiguation as in Algorithm 2;
```

4 EVALUATION

The evaluation of our algorithms is two-fold. Firstly, we need to evaluate the ability of our bootstrapping strategy to improve the classification performance compared to the underlying classification method on actual data. Secondly, we have to evaluate the actual performance of our parallel algorithms under real-world conditions.

In both cases, we must select commonly used, freely available real-world benchmark data-sets to ensure comparability and repeatability. We therefore retrieved most of our data sets from the UCI Machine Learning Repository [12]. Based on the requirements of our approach, we primarily selected data sets with a high number of features, i.e. a high dimensionality, and a low number of training patterns. Our first choice was the Semeion Digit Recognition data set[1] consisting of 1 593 instances with 256 features in 10 classes. In order to test the parallel scalability on a larger collection, we have decided to use the Letter Recognition data set[2] with 20 000 instances of 16 features in 26 classes. Thirdly, we have extracted three-dimensional $8 \times 8 \times 8$ HSV color histograms from the images in the Wang image collection[3]. This collection consists of 1 000 instances with 512 features and 10 classes.

Table 3 shows the results for the individual categories of the Semeion data set. These were obtained with 10 iterations, which were performed on a training set composed of 50 patterns per class.

For some categories, e.g. for the numbers seven and nine, we observe a significant improvement in the classification performance. In Table 4, the comparison of the classification performance of our approach is summarized. For reasons of clarity and comprehensibility, the average precision and recall have been calculated over all classes. The measurements clearly indicate that our bootstrapping strategy allows us to overcome SCALLOP's weaknesses in working with these data sets.

Category	0	1	2	3	4	5	6	7	8	9
P_{new}	**0.93**	**0.57**	**0.79**	**0.81**	**0.66**	**0.67**	**0.96**	**0.87**	**0.88**	**0.67**
P_{old}	0.65	0.32	0.32	0.55	0.54	0.46	0.22	0.13	0.49	0.12
R_{new}	**0.90**	**0.89**	**0.64**	**0.85**	**0.87**	**0.88**	**0.78**	0.67	**0.71**	**0.91**
R_{old}	0.36	0.77	0.60	0.40	0.34	0.44	0.22	**0.80**	0.25	0.75

Table 3. Classification performance for individual categories of the Semeion data set, comparing the precision P_{new} and recall R_{new} of our approach to the precision P_{old} and recall R_{old} of the adapted SCALLOP

Collection Name	Semeion	Letter Recognition	Wang
P_{new}	**0.81**	**0.65**	**0.86**
P_{old}	0.39	0.17	0.44
R_{new}	**0.79**	**0.62**	**0.85**
R_{old}	0.44	0.41	0.65

Table 4. Classification performance for the three data sets, comparing the average precision P_{new} and recall R_{new} of our approach to the average precision P_{old} and recall R_{old} of the adapted SCALLOP over all categories

[1] `http://archive.ics.uci.edu/ml/datasets/Semeion+Handwritten+Digit`

[2] `http://archive.ics.uci.edu/ml/datasets/Letter+Recognition`

[3] `http://wang.ist.psu.edu/docs/related.shtml`

To evaluate the speed-up of our parallel voting strategy experimentally, we have implemented our algorithm in C using OpenMPI and conducted response time measurements on a small cluster consisting of eight nodes interconnected via an Infiniband network. Each node is equipped with dual Intel Xeon E5520 processors clocked at 2.27 GHz with 4 cores and 48 GiB of RAM. Figure 3 depicts our timing measurements and the corresponding speed-up results.

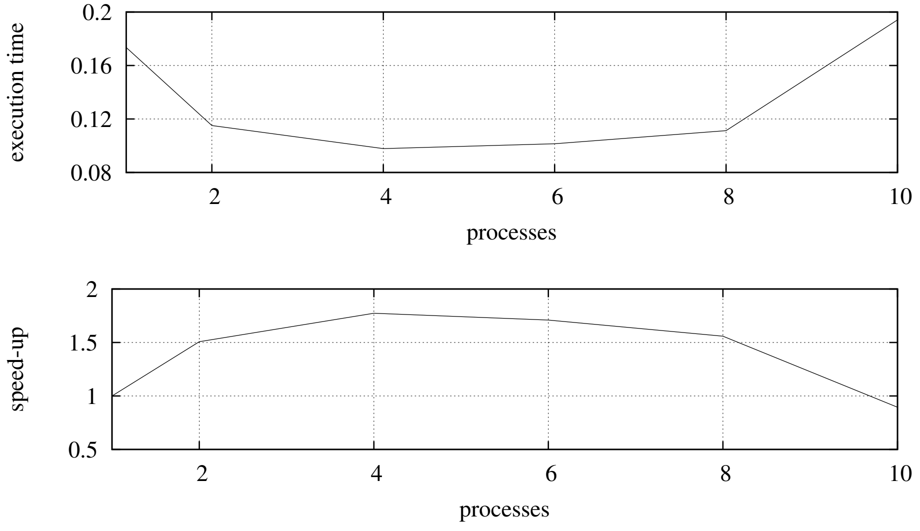
Due to the fact that each construction of a recognizer or discriminator, i.e. φ_i^t or $\psi_{(i,j)}^t$, is independent from each other and hence no communication is needed, it is obvious that under the assumption that these constructions all take the same time to be executed, the whole classifier construction can be done with ideal speed-up. However, we have implemented Algorithm 3 just to validate this in practice for our SCALLOP adaptation. As our expectations were fully satisfied, we omit further discussion of execution time and speed-up for the classifier construction.

Although some parts of the voting – the actual classification – are also embarrassingly parallel and hence could be done with ideal speed-up, the final steps of both voting stages constitute a bottleneck since all local results need to be gathered and evaluated to compute the global result. From a theoretical point of view, the second stage of the parallel voting also has potential for ideal speed-up (aside from the communication effort and the bottleneck at the very last step), but de facto the patterns which need to pass the second stage will not follow the uniform distribution for which our static parallelization algorithm is a priori perfect. In the – theoretically and empirically extremely unlikely – worst case it could happen that there is no speed-up at all for the second phase of the voting.

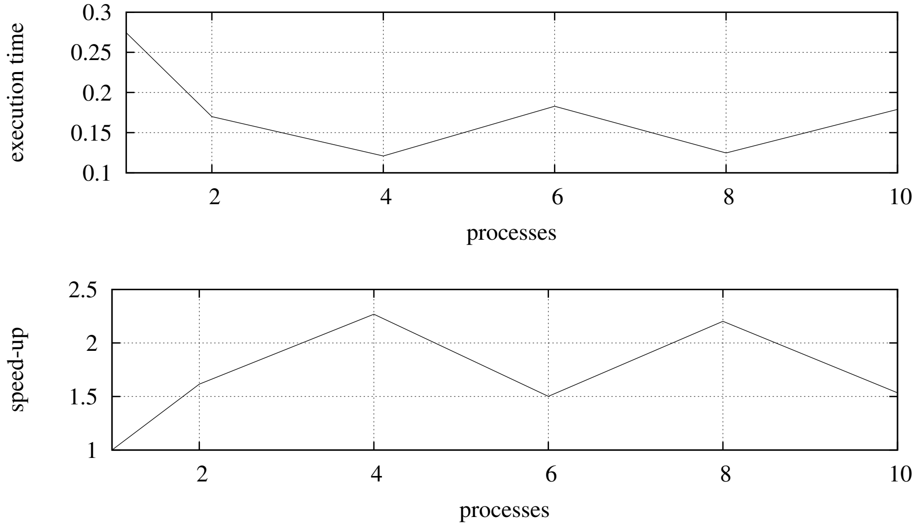
We can clearly see that for the two small data sets Wang and Semeion (Figures 3 a) and 3 b)), we can get a reasonable speed-up for up to eight processes. These results indicate that we can obtain some speed-up by programming a multicore CPU with MPI, but the low number of classes $C = 10$ limits the utility of the parallel voting process. The Letter Recognition data set with more classes ($C = 26$) has a better ratio between communication and processing and therefore the speed-up values in Figure 3 c) are very promising. This demonstrates the utility of our approach for larger numbers of classes. It is obvious that a large number of patterns N which has to be classified, as well as a large number of classes C, leads to a better ratio between execution and communication time and hence to a better speed-up.

As the last comment on Figure 3, we would like to mention the influence of architecture, interconnection and cache memory, which are reasons for non-smooth speed-up curves. Clearly, different results can be expected if, for example, 4 processes are allocated on a single processor or 2 processes on each of 2 processors. Furthermore, differences with regard to communication times are likely to occur if 2 processes are allocated on the same processor or not. With this in mind, it is not completely surprising that under certain circumstances, such as the case $P = 4$ in Figure 3 c) shows, a super-linear speed-up can be observed.

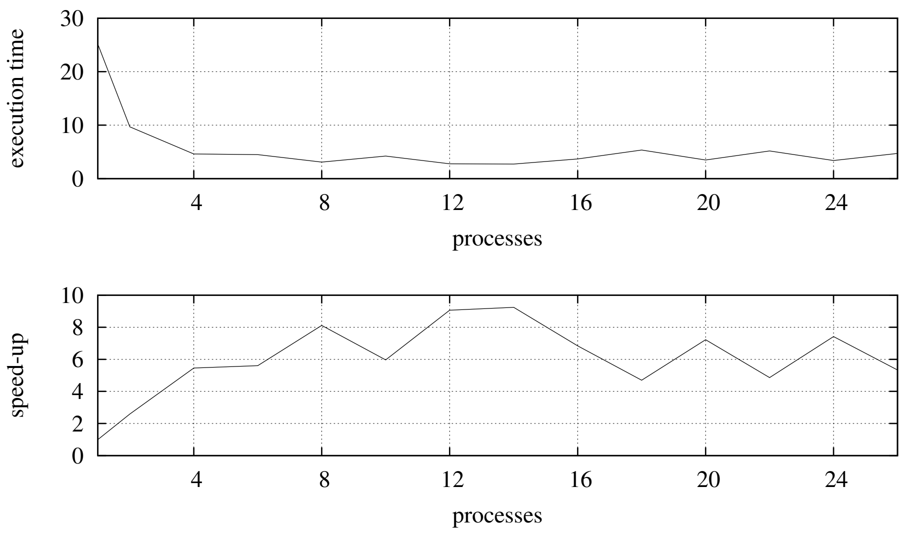
Recapitulating, we have demonstrated that our parallel voting strategy can be used to accelerate the classification on one or more multi-core processors.


execution time
0.2
0.16
0.12
0.08
2
4
6
8
10
processes
speed-up
2
1.5
1
0.5
2
4
6
8
10
processes

a)


execution time
0.3
0.25
0.2
0.15
0.1
2
4
6
8
10
processes
speed-up
2.5
2
1.5
1
2
4
6
8
10
processes

b)



c)

Figure 3. Parallel Voting Performance Measurements: a) Wang ($n = 512$, $N = 1000$, $C = 10$), b) Semeion ($n = 256$, $N = 1593$, $C = 10$), c) Letter Recognition ($n = 16$, $N = 20000$, $C = 26$)

5 SUMMARY, CONCLUSIONS AND FUTURE WORK

In this paper, we have introduced a novel method for the construction of classifiers for problems with a high dimensionality but a low number of training patterns, enabling us to obtain an up to three-fold increase in precision and significantly increased recall values in our experiments. It is based on a combination of a bootstrapping method with a one-versus-all classifier construction method. By resampling the training examples, we construct a range of two-class discriminators that determine if a new data point belongs to one particular class, or to any other class.

We combine these individual classifiers into a two-stage meta-classifier architecture, which treats the output of the individual classifiers as votes. Since our approach is a meta-classification framework that relies on a voting in an ensemble of elementary classifiers, the construction of the individual classifiers is embarrassingly parallel and can be achieved with a trivial data replicated, task parallel approach.

The classification process requires some more thought, and we have thus outlined parallelization methods for the classifier construction and the voting. A parallel classification algorithm based on message-passing has been derived, and we have conducted an evaluation of the run-time performance on server-grade hardware. Our results indicate that parallel voting is indeed a useful approach for data sets with a large number of classes.

For the reason of a more balanced workload, we aim to design a more dynamic distribution of the second stage of our algorithm. Despite the fact that the parallel implementation is not the core contribution of this paper, it would be interesting to test the performance of such an improved parallel approach for other architecture concepts (e.g. GPU), which could lead to a reduction of the communication overhead. Last but not least, in place of adapted SCALLOP, we plan to further investigate and apply our approach to other well-known classifiers, for example k-nearest neighbor or proper classifiers from the artificial neural network area.

Acknowledgements

The authors wish to thank the reviewers for their helpful comments. The third co-author was partly supported by the VEGA grant No. 2/0003/11 from the Scientific Grant Agency of the Ministry of Education and Slovak Academy of Sciences, Slovakia.

REFERENCES

[1] MacQueen, J. B.: Some Methods for Classification and Analysis of Multivariate Observations. Proceedings of 5th Berkeley Symposium on Mathematical Statistics and Probability 1967, pp. 281–297.

[2] Bauer, E.—Kohavi, R.: An Empirical Comparison of Voting Classification Algorithms: Bagging, Boosting, and Variants. Machine Learning, Vol. 36, July 1999, pp. 105–139.

[3] Schapire, R. E.: The Strength of Weak Learnability. Machine Learning, Vol. 5, July 1990, pp. 197–227.

[4] Yu, C.—Skillicorn, D. B.: Parallelizing Boosting and Bagging. Technical report, Queen's University, Kingston, Canada 2001.

[5] Palit, I.—Reddy, C. K.: Scalable and Parallel Boosting with MapReduce. IEEE Trans. Knowl. Data Eng., 2011, No. 99, p. 1.

[6] Ng, R. T.—Han, J.: CLARANS: A Method for Clustering Objects for Spatial Data Mining. IEEE Trans. on Knowl. and Data Eng., Vol. 14, 2002, pp. 1003–1016.

[7] Rifkin, R.—Klautau, A.: In Defense of One-vs-All Classification. Journal of Machine Learning Research, Vol. 5, 2004, pp. 101–141.

[8] Efron, B.: Bootstrap Methods: Another Look at the Jackknife. Annals of Statistics, Vol. 7, 1979, pp. 1–26.

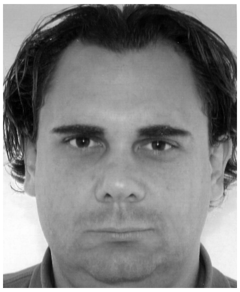
[9] Ferrer-Troyano, F. R.—Aguilar-Ruiz, J. S.—Riquelme, J. C.: Mining Low Dimensionality Data Streams of Continuous Attributes. Proc. EPIA 2003, pp. 264–278.

[10] Discovering Decision Rules from Numerical Data Streams. Proceedings of the 2004 ACM Symposium on Applied Computing, ser. SAC '04, pp. 649–653.

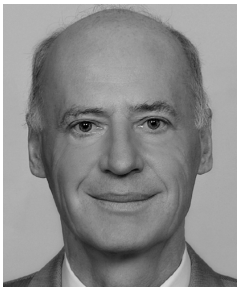
[11] Message Passing Interface Forum, MPI: A Message-Passing Interface Standard, Version 2.2, 2009.

[12] Frank, A.—Asuncion, A.: UCI Machine Learning Repository. `http://archive.ics.uci.edu/ml`, University of California, Irvine, School of Information and Computer Sciences 2011.

Verena Christina Horak studied mathematics, computer sciences and philosophy at the University of Salzburg, Austria, where she received Bachelor degrees in applied informatics (2005) and mathematics (2005), and Master degrees in mathematics (2008), applied mathematics (2008) and applied informatics (2010).



Tobias Berka received a B. Sc. and M. Sc. with honors in computer sciences from the University of Salzburg, Austria, in 2008 and 2009, and his Ph. D. in early 2012 as a student of Prof. Marian Vajteršic. As a visiting researcher, he has worked at the Computer Laboratory of the University of Cambridge, U.K., INRIA Saclay, France, and Purdue University, U.S.A. He has published five journal articles and seventeen peer-reviewed full papers, including two full papers as sole author. His research is in the field of parallel algorithms and parallel computing with applications in information retrieval and data mining. In 2012, he was awarded a highly competitive ERCIM Fellowship.



Marian Vajteršic graduated in numerical mathematics from Comenius University, Bratislava (Slovak Republic) in 1974. He received his C. Sc. (candidate of sciences) degree in mathematics from the same university in 1984 and he defended there the Dr. Sc. (doctor of sciences) degree in 1997. In 1995, he obtained the habilitation degree in numerical mathematics and parallel processing from the University of Salzburg (Austria). His research activity is focused on the area of parallel numerical algorithms for high-performance computer systems. He is author of two monographs, co-author of five other books and of more than 100 scientific papers. Since 1974, he is with the Slovak Academy of Sciences in Bratislava, Slovakia. As a visiting professor he has been with the universities of Vienna, Bologna, Milan, Linz, Salzburg, Amiens and Munich. Since 2002 he is a Professor for Scientific Computing and Computer Architecture at the Department of Computer Sciences of the University of Salzburg, Austria.