We are IntechOpen, the world's leading publisher of Open Access books Built by scientists, for scientists

4,800
Open access books available

122,000
International authors and editors

135M
Downloads

Our authors are among the

154
Countries delivered to

TOP 1%
most cited scientists

12.2%
Contributors from top 500 universities

CLARIVATE ANALYTICS BOOK CITATION INDEX INDEXED

WEB OF SCIENCE™

Selection of our books indexed in the Book Citation Index in Web of Science™ Core Collection (BKCI)

Interested in publishing with us?
Contact book.department@intechopen.com

Numbers displayed above are based on latest data collected.
For more information visit www.intechopen.com

13

Scaling up instance selection algorithms by dividing-and-conquering

Aida de Haro-García, Juan Antonio Romero del Castillo and Nicolás García-Pedrajas
University of Córdoba
Spain

1. Introduction

The overwhelming amount of data that is available nowadays in any field of research poses new problems for machine learning methods. This huge amount of data makes most of the existing algorithms inapplicable to many real-world problems. Two approaches have been used to deal with this problem: scaling up machine learning algorithms and data reduction. Nevertheless, scaling up a certain algorithm is not always feasible. On the other hand, data reduction consists of removing from the data missing, redundant and/or erroneous data to get a tractable amount of data. The most common methods for data reduction are instance selection and feature selection.

However, these algorithms for data reduction have the same scaling problem they are trying to solve. For example, in the best case, most existing instance selection algorithms are $O\left(n^2\right)$, n being the number of instances. For huge problems, with hundreds of thousands or even millions of instances, these methods are not applicable. The same happens with feature selection algorithms.

The alternative is scaling up the machine learning algorithm itself. In the best case, this is an arduous task, and in the worst case and impossible one. In this chapter we present a new paradigm for scaling up machine learning algorithms based on the philosophy of divide-and-conquer. One natural way of scaling up a certain algorithm is dividing the original problem into several simpler subproblems and applying the algorithm separately to each subproblem. In this way we might scale up instance selection dividing the original dataset into several disjoint subsets and performing the instance selection process separately on each subset. However, this method does not work well, as the application of the algorithm to a subset suffers from the partial knowledge it has of the dataset. However, if we join this divide-and-conquer approach with the basis of the construction of ensembles of classifiers, the combination of weak learners into a strong one, we obtain a very powerful and fast method, applicable to almost any machine learning algorithm. This method can be applied in different ways. In this chapter we propose two algorithms, recursive divide-and-conquer and democratization, that are able to achieve very good performance and a dramatic reduction in the execution time of the instance selection algorithms.

We will describe these methods and we will show how they can achieve very good results when applied to instance selection. Furthermore, the methodology is applicable to other machine learning algorithms, such as feature selection and cluster analysis.

Instance selection (Liu & Motoda, 2002) consists of choosing a subset of the total available data to achieve the original purpose of the data mining application as if the whole data were used. Different variants of instance selection exist. Many of the approaches are based on some form of sampling (Cochran, 1997) (Kivinen & Mannila, 1994). There are other more modern methods that are based on different principles, such as, Modified Selective Subset (MSS) (Barandela et al., 2005), entropy-based instance selection (Son & Kim, 2006), Intelligent Multiobjective Evolutionary Algorithm (IMOEA) (Chen et al., 2005), and LVQPRU method (Li et al., 2005).

The problem of instance selection for instance based learning can be defined as (Brighton & Mellish, 2002) "the isolation of the smallest set of instances that enable us to predict the class of a query instance with the same (or higher) accuracy than the original set". It has been shown that different groups of learning algorithms need different instance selectors in order to suit their learning/search bias (Brodley, 1995). This may render many instance selection algorithm useless, if their philosophy of design is not suitable to the problem at hand.

We can distinguish two main models of instance selection (Cano et al., 2003): instance selection as a method for prototype selection for algorithms based on prototypes (such as *k*-Nearest Neighbors) and instance selection for obtaining the training set for a learning algorithm that uses this training set (such as decision trees or neural networks). This chapter is devoted to the former methods.

Regarding complexity, in the best case, most existing instance selection algorithms are of efficiency $O\left(n^2\right)$, n being the number of instances. For huge problems, with hundreds of thousands or even millions of instances, these methods are not applicable. Trying to develop algorithms with a lower efficiency order is likely to be a fruitless search. Obtaining the nearest neighbor of a given instance is $O(n)$. To test whether removing an instance affects the accuracy of the nearest neighbor rule, we must measure the effect on the other instances of the absence of the removed one. Measuring this effect involves recalculating, directly or indirectly, the nearest neighbors of the instances. The result is a process of $O\left(n^2\right)$. In this way, the attempt to develop algorithms of an efficiency order below this bound is not very promising.

Thus, the alternative is reducing the size n of the set to which instance selection algorithms are applied. In the construction of ensembles of classifiers the problem of learning from huge datasets has been approached by means of learning many classifiers from small disjoint subsets (Chawla et al., 2004). In that paper, the authors showed that it is also possible to learn an ensemble of classifiers from random disjoint partitions of a dataset, and combine predictions from all those classifiers to achieve high classification accuracies. They applied their method to huge datasets with very good results. Furthermore, the usefulness of applying instance selection to disjoint subsets has also been shown in (García-Pedrajas et al., 2009). In that work, a cooperative evolutionary algorithm was used. The training set was divided into several disjoint subsets and an evolutionary algorithm was performed on each subset of instances. The fitness of the individuals was evaluated only taking into account the instances in the subset. To account for the global view needed by the algorithm a global

population was used. This method is scalable to medium/large problems but cannot be applied to huge problems. Zhu & Wu (2006) also used disjoint subsets in a method for ranking representative instances.
Following this idea, we will present in this chapter two approaches for scaling up instance selection algorithms that are based on a divide-and-conquer approach. The presented methods are able to achieve very good performance with a drastic reduction in the time needed for the execution of the algorithms. The general idea underlying this work is dividing the original dataset into subsets and performing the instance selection process in each subset separately. Then, we must find a method for combining the separate applications of the instance selection algorithm to a final global result.
The rest of this paper is organized as follows: Section 2 revised some related work; Section 3 describes in depth our proposal; Section 4 shows the experiments performed with our methods; and finally Section 5 states the conclusions of our work.

2. Related work

As stated in the previous section, scaling up instance selection algorithms is a very relevant issue. The usefulness of applying instance selection to disjoint subsets has also been shown in (García-Pedrajas et al., 2009). In this work a cooperative evolutionary algorithm is used. Several evolutionary algorithms are performed on disjoint subsets of instances and a global population is used to account for the global view. This method is scalable to medium/large problems but cannot be applied to huge problems.
There are not many previous works that have dealt with instance selection for huge problems. Cano et al. (2005) proposed an evolutionary stratified approach for large problems. Although the algorithm shows very good performance, it is still too computationally expensive for huge datasets. Kim & Oommen (2004) proposed a method based on a recursive application of instance selection to smaller datasets.
In a recent paper, De Haro-García and García-Pedrajas (2009) showed that the application of a recursive divide-and-conquer approach is able to achieve a good performance while attaining a dramatic reduction in the execution time of the instance selection process.

3. Scaling up instance selection algorithms using divide-and-conquer philosophy

As stated in the previous section, scaling up instance selection algorithms is a very relevant issue. In this section we will discuss our proposals based on a divide-and-conquer approach. The two methods aim a general objective of scaling up instance selection algorithms. However, individually they have two different aims. The first one, based on recursively using the principle of divide-and-conquer, has as main goal the reduction of the time needed by the instance selection process. In this way, it trades time for performance, allowing a small increase on the testing error achieved by the algorithms. The second one, based on combining the divide-and-conquer approach with principles from ensembles of classifiers, has as special objective reducing the time of the algorithms, but keeping their performance as much as possible.

3.1 Recursive divide-and-conquer approach

As we have said, in the best case, most existing instance selection algorithms are of efficiency $O\left(n^2\right)$. For huge problems, with hundreds of thousands or even millions of instances, these methods are not applicable. Trying to develop algorithms with a lower efficiency order is likely to be a fruitless search.

Following the divide-and-conquer philosophy, we can develop a methodology based on applying the instance selection algorithm to subsets of the whole training set. A simple approach consists of using a stratified random sampling (Liu & Motoda, 2002) (Cano et al., 2005), where the original dataset is divided into many disjoint subsets, and then apply instance selection over each subset independently. However, due to the fact that to select the nearest neighbor of an instance we need to know the whole dataset, this method is not likely to produce good results. In fact, in practice its performance is poor. However, the divide-and-conquer principle of this method is an interesting idea for scaling up instance selection algorithms. Furthermore, divide-and-conquer methodology has the additional advantage that we can adapt the size of the subproblems to the available resources.

Following this philosophy we start performing a partition of the dataset and then applying the instance selection algorithm to every subset independently. This step is able to be performed fast and obtains good results in terms of testing error. However, its results in terms of storage reduction are poor. To avoid this drawback we apply this method recursively. After an instance selection step is performed the remaining instances are rejoined to obtain again subsets of approximately the same size and the instance selection process is repeated. Fig. 1 shows and outline of the process.

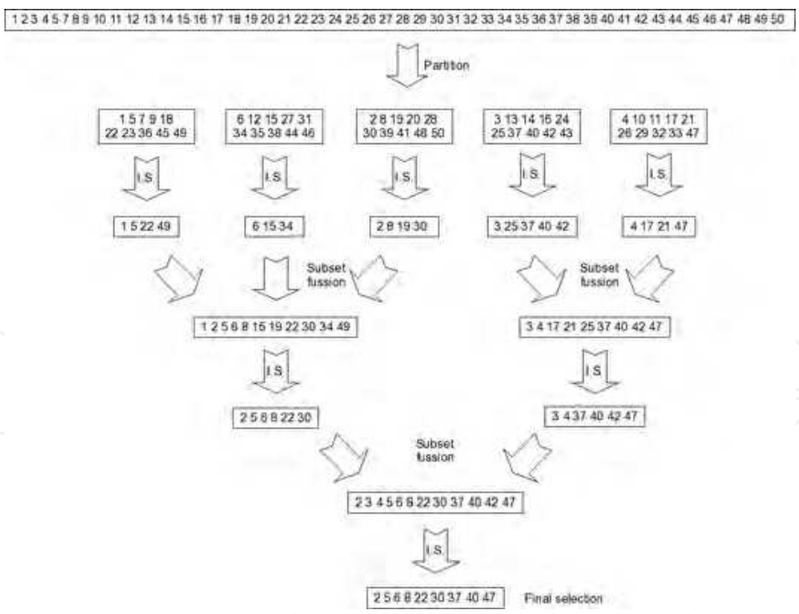


Fig. 1. Outline of recursive divide-and-conquer instance selection method

This method is applicable to any instance selection algorithm, as the instance selection algorithm is a parameter of the method. More formally, first, our method divides the whole training set, T, into disjoint subsets, t_i, of size s such as $T = \cup t_i$. s is the only

parameter of the algorithm. In this study the dataset is randomly partitioned, although other methods may be devised. Then, the instance selection algorithm of our choice is performed over every subset independently. The selected instances in each subset are joined again. With this new training set constructed with the selected instances, the process is repeated until a certain stop criterion is fulfilled. The process of combining the instances selected by the execution of the instance selection algorithm over each dataset can be performed in different ways. We can just repeat the partition process as in the original dataset. However, as the first partition is performed we can take advantage of this performed task. In this way, instead of repeating the partitioning process, we join together the subsets of selected instances until new subsets of approximately size s are obtained. The detailed process is shown in Fig. 2.

Data: A training set $T=\{x_1, y_1, \ldots, x_n, y_n\}$, and subset size s.
Result: The reduced training set $S \subseteq T$.

$S=T$
divide instances into disjoint subsets $t_i : T=\cup t_i$ of size s

repeat until stop criterion

  for each subset t_i **do**
    apply instance selection algorithm to t_i to obtain $s_i \subseteq t_i$
    remove from S the instances removed from t_i
  end for

  fusion subsets s_i to obtain new subsets t_j of size s

end repeat

return S

Fig. 2. Recursive divide-and-conquer instance selection algorithm

The stop criterion may be obtained in different ways. We can have a goal in terms of testing error or reduction of storage and stop the algorithm when that goal is achieved. However, to avoid the necessity of setting any additional parameter, we obtain the stop criterion by means of cross-validation. We apply the algorithm using a cross-validation setup and obtain the number of steps before the testing error starts to grow. This number of steps gives the stopping criterion.

3.2 Democratic instance selection

The above method is very fast as it will be shown in the experimental results. However, it has the drawback of worsening the testing error achieved by some algorithms for certain problems. To improve the results of our approach in this aspect, we have developed a second method we called *democratic* instance selection. The method is also based on the general divide-and-conquer approach but including ideas from ensembles of classifiers.

Democratic instance selection is based on repeating several rounds of a fast instance selection process. Each round on its own would not be able to achieve a good performance. However, the combination of several rounds using a voting scheme is able to match the performance of an instance selection algorithm applied to the whole dataset with a large reduction in the time of the algorithm. Thus, in a different setup from the case of ensembles of classifiers, we can consider our method a form of "ensembling" instance selection.

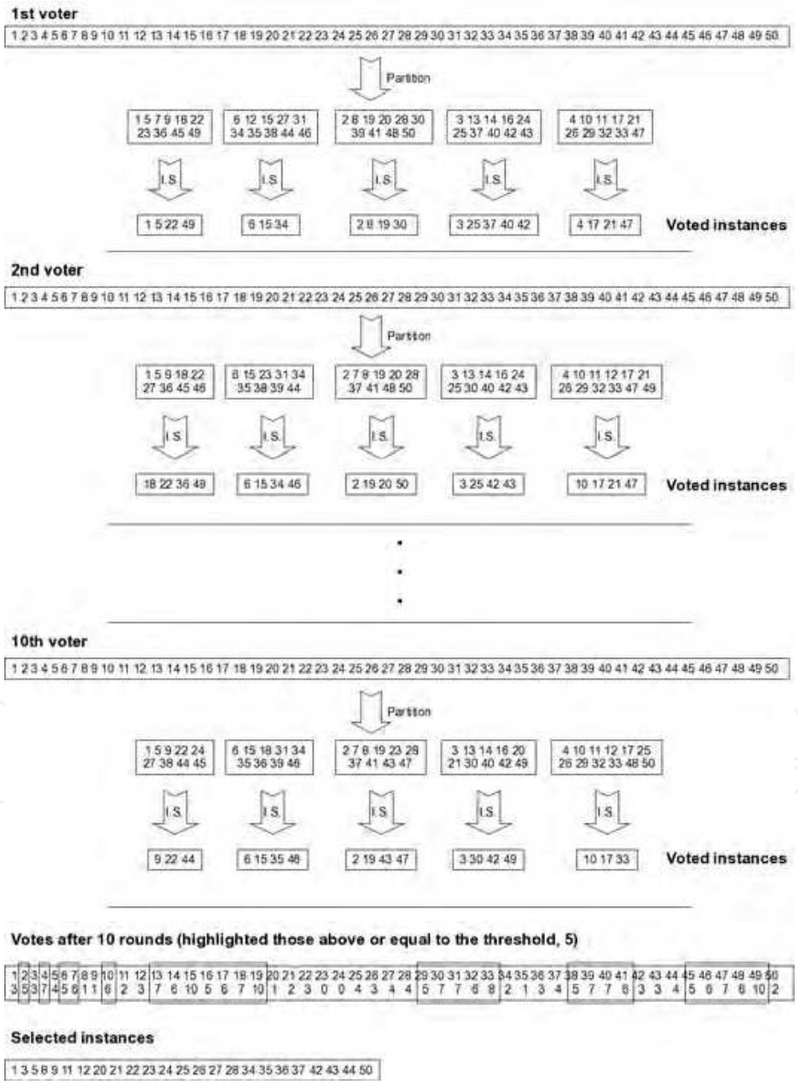


Fig. 3. Outline of democratic instance selection method

In classification, several weak learners are combined into an ensemble which is able to improve the performance of any of the weak learners isolated (García-Pedrajas et al., 2007). In our method, the instance selection algorithm applied to a partition into disjoint subsets of the original dataset can be considered a *weak instance selector*, as it has a partial view of the dataset. The combination of these weak selectors using a voting scheme is similar to the combination of different learners in an ensemble using a voting scheme. Fig. 3 shows a general outline of the method.

An important issue in our method is determining the number of votes needed to remove an instance from the training set. Preliminary experiments showed that this number highly depends on the specific dataset. Thus, it is not possible to set a general pre-established value usable in any dataset. On the contrary, we need a way of selecting this value directly from the dataset in run time.

A first natural choice would be the use of a cross-validation procedure. However, this method is very time consuming. A second choice is estimating the best value for the number of votes from the effect on the training set. This latter method is the one we have chosen. The election of the number of votes must take into account two different criteria: training error, ε_t , and storage, or memory, requirements m . Both values must be minimized as much as possible. Our method of choosing the number of votes needed to remove an instance is based on obtaining the threshold number of votes, v , that minimizes a fitness criterion, $f(v)$, which is a combination of these two values:

$$f(v) = a\varepsilon_t(v) + (1-a)m(v), \tag{1}$$

where α is a value in the interval *[0, 1]* which measures the relative relevance of both values. In general, the minimization of the error is more important than storage reduction, as we prefer a lesser error even if the reduction is smaller. Thus, we have used a value of $\alpha = 0.75$. Different values can be used if the researcher is more interested in reduction than in error. *m* is measured as the percentage of instances retained, and ε_t is the training error. However, estimating the training error is time consuming if we have large datasets. To avoid this problem the training error is estimated using only a small percentage of the whole dataset, which is 1% for medium and large datasets, and 0.1% for huge datasets.

Data: A training set $T = \{(x_1, y_1), \ldots, (x_n, y_n)\}$, subset size *s*, and number of rounds *r*.

Result: The set of selected instances $S \subset T$.

for i = 1 **to** *r* **do**
  divide instances into disjoint subsets $t_i : \cup t_i = T$ of size *s*
  for each t_i **do**
    apply instance selection algorithm to t_i
    store votes of removed instances from t_i
  end for

obtain threshold of votes, v , to remove an instance
$S=T$
remove from S all instances with a number of votes above or equal to v
return S

Fig. 4. Democratic instance selection algorithm

More formally, the process is the following: We perform r rounds of the algorithm and store the number of votes received by each instance. Then, we must obtain the threshold number of votes, v, to remove an instance. This value must be $v \in [1,r]$. We calculate the criterion $f(v)$ (eq. 1) for all the possible threshold values from 1 to r, and assign v to the value which minimizes the criterion. After that, we perform the instance selection removing the instances whose number of votes is above or equal to the obtained threshold v. Fig. 4 shows the steps of this algorithm.

4. Experimental setup and results

In order to make a comprehensive comparison between the standard algorithms and our proposal we have selected a set of 30 problems from the UCI Machine Learning Repository (Hettich et al., 1998). A summary of these data sets is shown in Table 1. We have selected datasets with, at least, 1000 instances. For estimating the storage reduction and generalization error we used a *k*-fold cross-validation method. In this method the available data is divided into *k* approximately equal subsets. Then, the method is learned *k* times, using, in turn, each one of the *k* subsets as testing set, and the remaining *k-1* subsets as training set. The estimated error is the average testing error of the *k* subsets. A fairly standard value for *k* is *k* = 10.

4.1 Evaluating instance selection methods

The evaluation of a certain instance selection algorithm is not a trivial task. We can distinguish two basic approaches: direct and indirect evaluation (Liu & Motoda, 2002). Direct evaluation evaluates a certain algorithm based exclusively on the data. The objective is to measure at which extent the selected instances reflect the information present in the original data. Some proposed measures are entropy (Cover & Thomas, 1991), moments (Smith, 1998), and histograms (Chaudhuri et al., 1998).
Indirect methods evaluate the effect of the instance selection algorithm on the task at hand. So, if we are interested in classification we evaluate the performance of the used classifier when using the reduced set obtained after instance selection as learning set.

		Features				
Data set	Instances	Real	Binary	Nominal	Classes	1-NN error
abalone	4177	7	-	1	29	0.8034
adult	48842	6	1	7	2	0.2005
car	1728	-	-	6	4	0.1581
gene	3175	-	-	60	3	0.2767
german	1000	6	3	11	2	0.3120
hypothyroid	3772	7	20	2	4	0.0692
isolet	7797	617	-	-	26	0.1443
krkopt	28056	6	-	-	18	0.4356
kr vs. kp	3196	-	34	2	2	0.0828
letter	20000	16	-	-	26	0.0454
magic04	19020	10	-	-	2	0.2084
mfeat-fac	2000	216	-	-	10	0.0350
mfeat-fou	2000	76	-	-	10	0.2080
mfeat-kar	2000	64	-	-	10	0.0435
mfeat-mor	2000	6	-	-	10	0.2925
mfeat-pix	2000	240	-	-	10	0.0270
mfeat-zer	2000	47	-	-	10	0.2140
nursery	12960	-	1	7	5	0.2502
optdigits	5620	64	-	-	10	0.0256
page-blocks	5473	10	-	-	5	0.0369
pendigits	10992	16	-	-	10	0.0066
phoneme	5404	5	-	-	2	0.0952
satimage	6435	36	-	-	6	0.0939
segment	2310	19	-	-	7	0.0398
shuttle	58000	9	-	-	7	0.0010
sick	3772	7	20	2	2	0.0430
texture	5500	40	-	-	11	0.0105
waveform	5000	40	-	-	3	0.2860
yeast	1484	8	-	-	10	0.4879

Table 1. Summary of datasets used in our experiments

Therefore, when evaluating instance selection algorithms for instance learning, the most usual way of evaluation is estimating the performance of the algorithms on a set of benchmark problems. In those problems several criteria can be considered, such as (Wilson & Martínez, 2000): storage reduction, generalization accuracy, noise tolerance, and learning speed. Speed considerations are difficult to measure, as we are evaluating not only an

algorithm but also a certain implementation. However, as the main aim of our work is scaling up instance selection algorithms, execution time is a basic issue. To allow a fair comparison, we have performed all the experiments in the same machine, a bi-processor computer with two Intel Xeon QuadCore at 1.60GHz.

One of the advantages of our approach is that it can be applied to any kind of instance selection method. As the instance selection method to apply is just a parameter of the algorithm, there is no restriction in the algorithm selected. In the experiments we have used several of the most widely used instance selection methods.

In order to obtain an accurate view of the usefulness of our method, we must select some of the most widely used instance selection algorithms. We have chosen to test our model using several of the most successful state-of-the-art algorithms. Initially, we used the algorithm ICF (Brighton & Mellish, 2002). ICF (Iterative Case Filtering) is based on the concepts of *coverage* and *reachability* of an instance *c*, which are defined as follows:

$$\text{Coverage}(c) = c' \in T : c \in LocalSet(c')$$

$$\text{Reachable}(c) = c' \in T : c' \in LocalSet(c)$$

The local-set of a case *c* is defined as "the set of cases contained in the largest hypersphere centered on *c* such that only cases in the same class as *c* are contained in the hypersphere" (Brighton & Mellish, 2002) so the hypersphere is bounded by the first instance of different class. The coverage set of an instance includes the instances that have this as one of their neighbors and the reachable set is formed by the instances that are neighbors to this instance. The algorithm is based on repeatedly applying a deleting rule to the set of retained instances until no more instances fulfill the deleting rule.

In addition to this method, it is worth mentioning Reduced Nearest Neighbor (RNN) rule (Gates, 1972). This method is extremely simple, but it also shows an impressive performance in terms of storage reduction. In fact, it is the best of the methods used in these experiments in reducing storage requirements, as will be shown in the next section. However, it has a serious drawback, its computational complexity. Among the standard methods used this is the one that shows a worst scalability, taking several hundreds hours in the worst case. Therefore, RNN is the perfect target for our methodology, an instance selection method highly efficient but with a serious scalability problem. So we have also tested our approach using RNN, as base instance selection method.

The same parameters were used for the standard version of every algorithm and its application within our methodology. All the standard methods have no relevant parameters, the only value we must set is *k*, the number of nearest neighbors. Both, for ICF and RNN, we used $k = 3$ neighbors. This is a fairly standard value (Cano et al., 2003). Our method has two parameters: subset size, *s*, for both methods, and number of rounds, *r*, for the democratic approach. Regarding subset size we must use a value large enough to allow for a meaningful application of the instance selection algorithm on the subset, and small enough to allow a fast execution, as the time used by our method grows with *s*. As a compromise value we have chosen $s = 100$. For the number of rounds we have chosen a small value to allow for a fast execution, $r = 10$. The application of our recursive divide-and-conquer method with a certain instance selection algorithm X will be named RECURIS.X and the democratic approach named DEMOIS.X.

4.2 Recursive divide-and-conquer approach

In this section we show the results using the recursive approach. First, we compare the proposed approach against standard instance selection methods in terms of testing error and storage requirements. In the next section we will show execution time results.

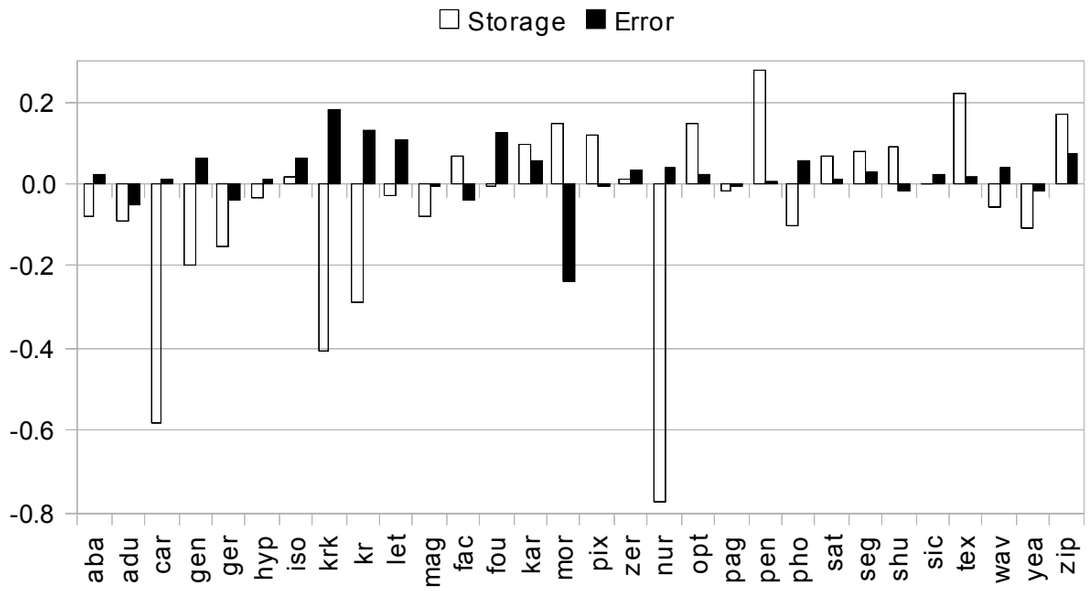


Fig. 5. Results of standard ICF method and its recursive counterpart for testing error and storage requirements

Fig. 5 shows the results comparing standard ICF and its recursive counterpart. The figure (as well as the following ones) shows for each dataset the difference between the standard method and our approach, a negative value meaning a better results of our proposal. The figure shows that in terms of storage reduction our method is better in general, achieving for some datasets, namely car, gene, german, krkopt, krvskp and nursery, significant improvements over the standard method. In terms of testing error RECURIS.ICF is slightly worse than standard ICF.

Fig. 6 shows the results for RNN as base instance selection method. This a very good test of our approach, as RNN is able to achieve very good results in terms of storage reduction while keeping testing error in moderate bounds. However, RNN has a big problem of scalability. The results show that our method is able to mostly keep the good performance of RNN in terms of storage requirements, although with a general worst behavior. However, this is compensated by a better testing error for most datasets.

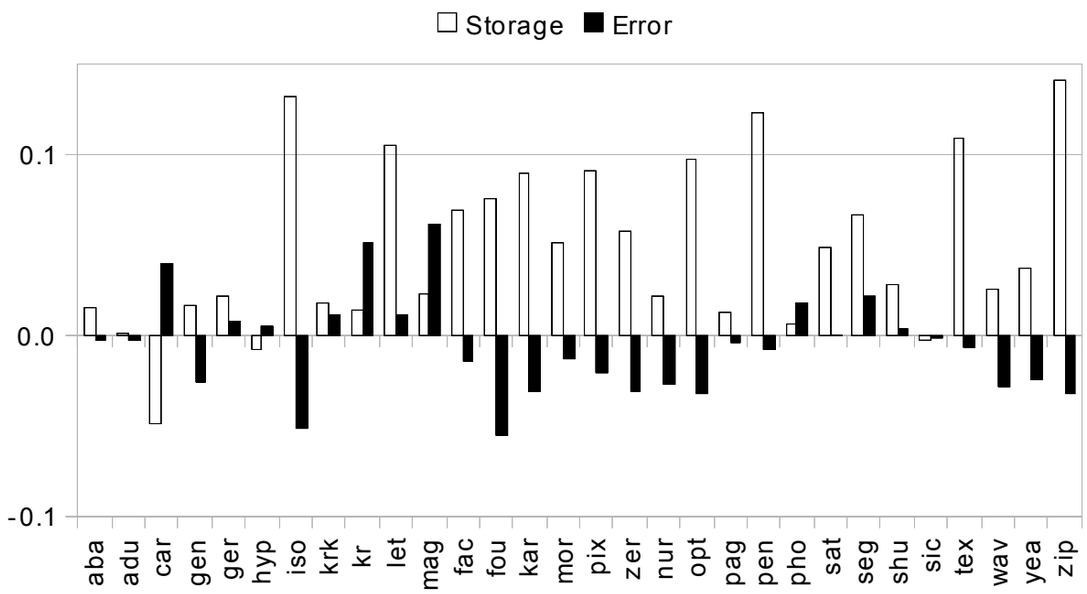


Fig. 6. Results of standard RNN and its recursive counterpart in terms of testing error and storage requirements

As an alternative to these standard methods, genetic algorithms have been applied to instance selection, considering this task to be a search problem. The application is easy and straightforward. Each individual is a binary vector that codes a certain sample of the training set. The evaluation is usually made considering both data reduction and classification accuracy. Examples of applications of genetic algorithms to instance selection can be found in (Kuncheva, 1995), (Ishibuchi & Nakashima, 2000) and (Reeves & Bush, 2001). Cano et al. (2003) performed a comprehensive comparison of the performance of different evolutionary algorithms for instance selection. They compared a generational genetic algorithm (Goldberg, 1989), a steady-state genetic algorithm (Whitley, 1989), a CHC genetic algorithm (Eshelman, 1990), and a population based incremental learning algorithm (Baluja, 1994). They found that evolutionary based methods were able to outperform classical algorithms in both classification accuracy and data reduction. Among the evolutionary algorithms, CHC was able to achieve the best overall performance.
In evolutionary computation, a population (set) of individuals (solutions to the problem faced) are codified following a code similar to the genetic code of plants and animals. This population of solutions is evolved (modified) over a certain number of generations (iterations) until the defined stop criterion is fulfilled. Each individual is assigned a real value that measures its ability to solve the problem, which is called its *fitness*.

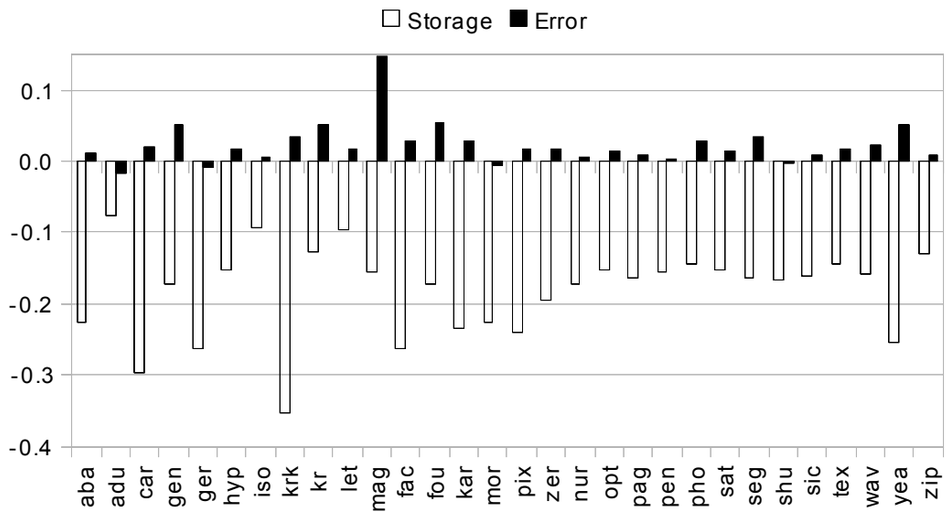
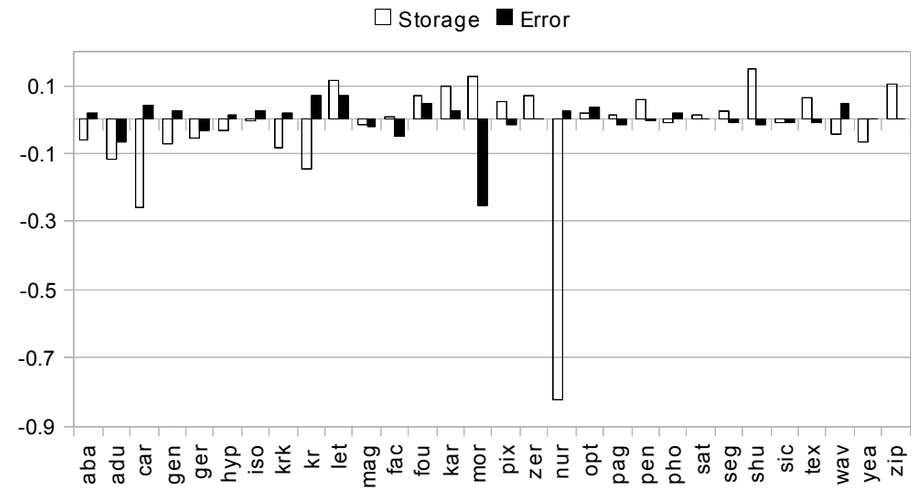


Fig. 7. Results of standard CHC and its recursive counterpart in terms of testing error and storage requirements

In each iteration new solutions are obtained combining two or more individuals (crossover operator) or randomly modifying one individual (mutation operator). After applying these two operators a subset of individuals is selected to survive to the next generation, either by sampling the current individuals with a probability proportional to their fitness, or by selecting the best ones (elitism). The repeated processes of crossover, mutation and selection are able to obtain increasingly better solutions for many problems of Artificial Intelligence.
Nevertheless, the major problem addressed when applying genetic algorithms to instance selection is the scaling of the algorithm. As the number of instances grows, the time needed for the genetic algorithm to reach a good solution increases exponentially, making it totally useless for large problems. As we are concerned with this problem, we have used as fifth instance selection method a genetic algorithm using CHC methodology. The execution time of CHC is clearly longer than the time spent by ICF, so it gives us a good benchmark to test our methodology on an algorithm that, as RNN, has a big scalability problem.
For CHC, see Fig. 7, the results show that the recursive approach is able to improve the results of the standard algorithm in terms of storage requirements but the error is worse than when using the whole dataset. However, the achieved storage reduction is relevant, and our method is clearly worse than standard CHC only in magic04 problem.
An interesting side result is the problem of scalability of CHC algorithm, which is more marked for this algorithm than for the previous ones. In other works, (Cano et al., 2003) (García-Pedrajas et al., 2009), CHC algorithm was compared with standard methods in small to medium problems. For those problems, the performance of CHC was better than the performance of other methods. However, as the datasets are larger, the scalability problem of CHC manifests itself. In our set of problems, CHC clearly performs worse than ICF and RNN in terms of storage reduction. We must take into account that for CHC we need a bit in the chromosome for each instance in the dataset. This means that for large problems, such as adult, krkopt, letter, magic or shuttle, the chromosome has more than 10000 bits, making the convergence of the algorithm problematic. Thus, CHC is, together with RNN, an excellent example of the applicability of our approach.



4.3 Democratic approach

In this section we show the results using the democratic approach. Results for ICF and DEMOIS.ICF are plotted in Fig. 8.

In terms of testing error, DEMOIS.ICF is able to match the results of ICF for most of the datasets. In terms of storage reduction the average performance of both algorithms is similar, with a remarkably good performance of DEMOIS.ICF for nursery and car datasets.

The next experiment is conducted using as base instance selection algorithm RNN. The results are plotted in Fig. 9. As we stated in the previous section, this is a perfect example of the potentialities of our approach. In our experiments RNN showed the best performance in terms of storage reduction. However, the algorithm has a very serious problem of scalability. As an extreme example, for adult problem it took more than 500 hours per experiment. This scalability problem prevents is application in those problems where it would be most useful.

Fig. 8. Results of standard ICF method and its democratic counterpart for testing error and storage requirements

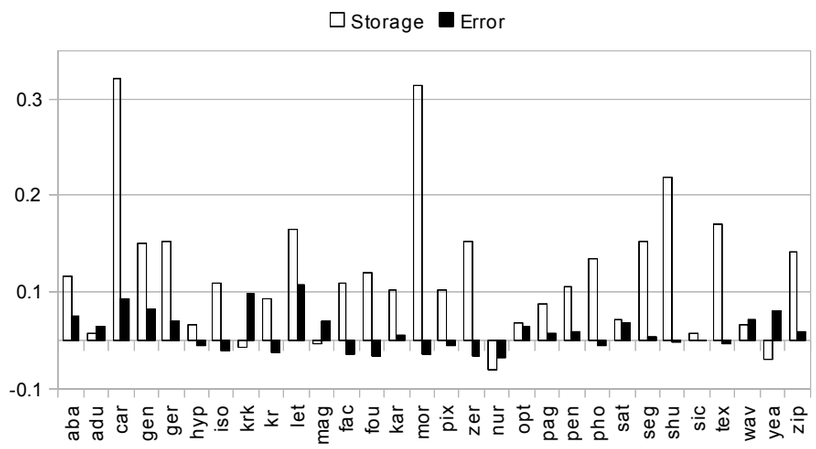


Fig. 9. Results of standard RNN method and its democratic counterpart for testing error and storage requirements

The figure shows how DEMOIS.RNN is able to solve the scalability problem of RNN. In terms of testing error, it is able to achieve a similar performance as standard RNN. In terms of storage reduction our algorithm performs worse than RNN. However, the performance of DEMOIS.RNN is still very good, in fact, better than any other of the previous algorithms. So, our approach is able to scale RNN to complex problems, improving its results in terms of testing error, but with a small worsening of the storage reduction. In terms of execution time the results are remarkable, the reduction of the time consumed by the selection process is large, with the extreme example of the two most time consuming datasets, adult and krkopt, where the speed-up is more than a hundred times (see next section).

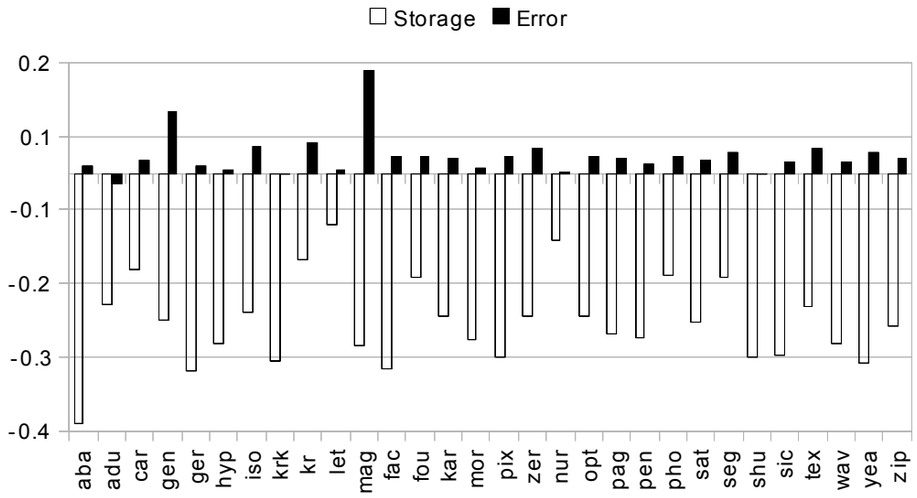


Fig. 10. Results of standard CHC method and its democratic counterpart for testing error and storage requirements

Fig. 10 plots the results of CHC algorithm. For this method, the scaling up of CHC provided by DEMOIS.CHC is evident not only in terms of running time, with a large reduction in all 30 datasets, but also in terms of storage reduction. DEMOIS.CHC is able to improve the reduction of CHC in all 30 datasets, with an average improvement of more than 20%, from an average storage of CHC of 31.83% to an average storage of 11.58%. The bad side effect is a worse testing error, which is however not very marked and compensated by the improvement in running time and storage reduction. As a summary, for CHC the results show that the democratic approach is able to improve the results of the standard algorithm in terms of storage requirements but the error is worse than when using the whole dataset. However, as it was the case for the recursive approach, there is a clear gaining in storage reduction with a moderately worse testing error.

4.4 Time

As we have estated our main aim is the scaling up of instance selection algorithms. In the previous sections we have shown that our methodology is able to match the performance of standard instance selection algorithms. In this section we show the results of execution time spent by each algorithm, showing a dramatic advantage of our approach. Fig. 11, 12 and 13 show the execution time of ICF, RNN and CHC methods respectively. The figures show execution time, in seconds, plotted against problem size.

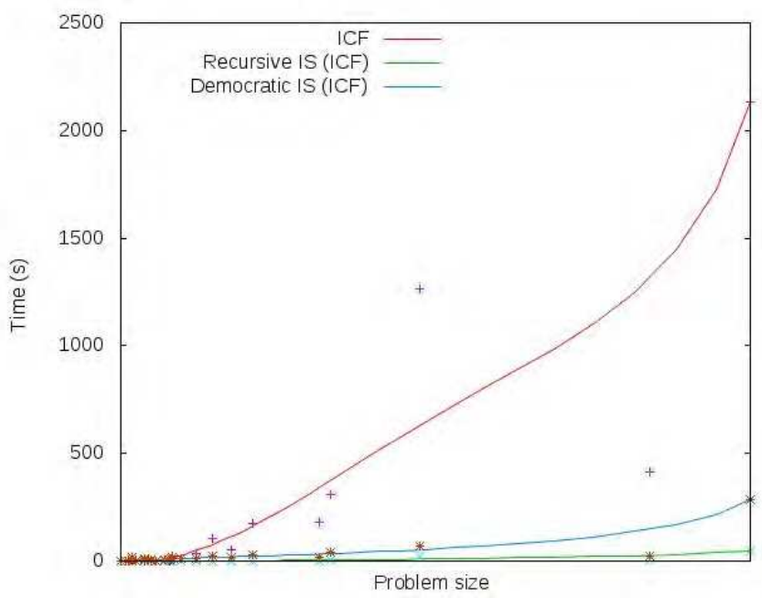


Fig. 11. Execution time using ICF as base instance selection algorithm

All the three figures show the excellent behavior of the two described methods. Both behave almost linearly as the problem size grows. On the other hand, ICF shows it is a quadratic complexity method and RNN and CHC behave far worse.

From a theoretical point of view the two algorithms presented in this chapter are of linear complexity. For the recursive approach we divide the dataset into n_s subsets of size s. Then, we apply the instance selection algorithm to each subset. The time needed for performing the selection in each subset will be fixed as the size of each subset is always s, regardless the number of instances of the datasets. More instances means a larger n_s. Thus, the complexity of each step of the recursive algorithm will be linear as n_s depends linearly on n, the size of the dataset. The algorithm performs a few of these steps before reaching the stopping criterion, and thus the whole method is of linear complexity.

The democratic approach also divides the dataset into partitions of disjoint subsets of size s. Thus, the chosen instance selection algorithm is always applied to a subset of fixed size, s, which is independent from the actual size of the dataset. The complexity of this application of the algorithm depends on the base instance selection algorithm we are using, but will always be small, as the size $s$is always small. Let K be the number of operations needed by the instance selection algorithm to perform its task in a dataset of size s. For a dataset of n instances we must perform this instance selection process once for each subset, that is n/s times, spending a time proportional to $(n/s)K$. The total time needed by the algorithm to perform r rounds will be proportional to $r(n/s)K$, which is linear in the number of instances, as K is a constant value.

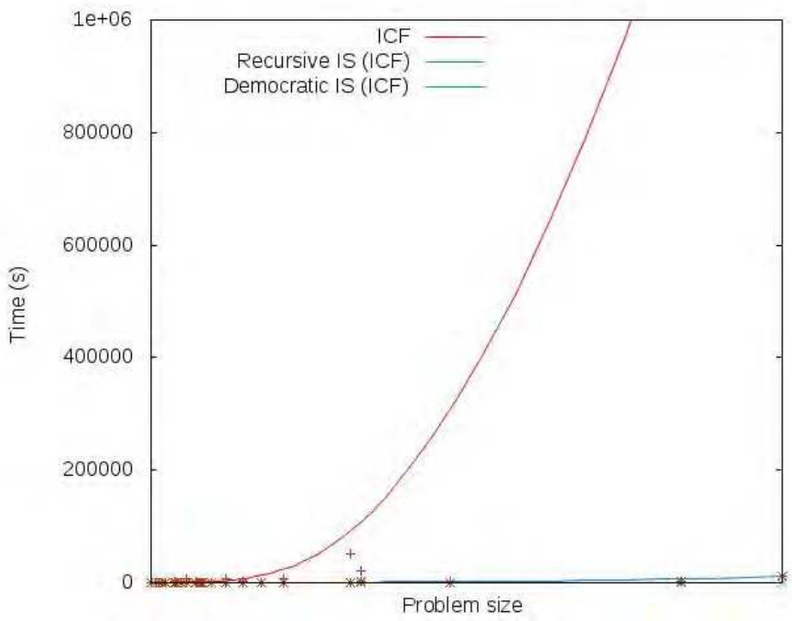


Fig. 12. Execution time using RNN as base instance selection algorithm.

Thus, the gaining in execution time would be greater as the size of the datasets is larger. If the complexity of the instance selection algorithm is greater, the reduction of the execution will be even better. The method has the additional advantage of allowing an easy parallel implementation. As the application of the instance selection algorithm to each subset is independent from all the remaining subsets, all the subsets can be processed at the same time, even for different rounds of votes. Also, the communication between the nodes of the parallel execution is small.

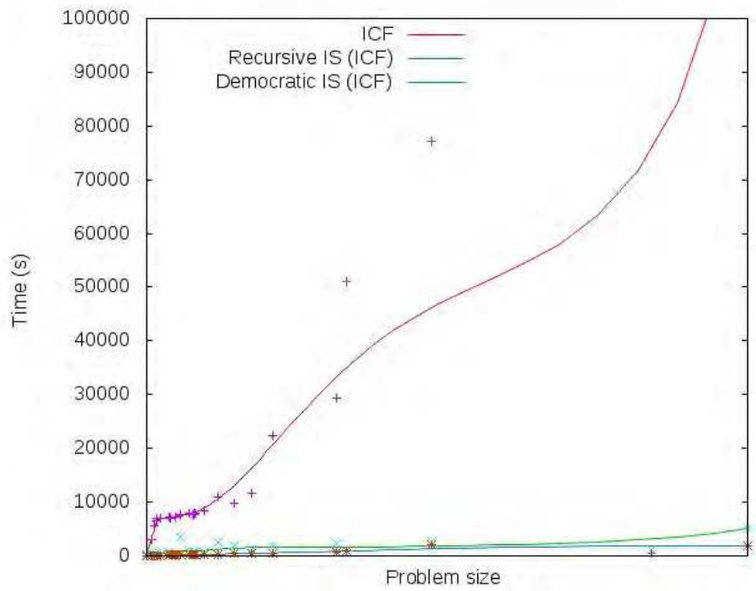


Fig. 13. Execution time using CHC as base instance selection algorithm

An additional process completes the method, the determination of the number of votes. Regarding the determination of the number of votes, the process can be made in different ways. If we consider all the training instances, the cost of this step would be $O\left(n^2\right)$.

However, to keep the complexity linear we use a random subset of the training set for determining the number of votes, with a limit on the maximum size of this subset that is fixed for any dataset. In this way, from medium to large datasets we use the 10% of the training set, for huge problems the 1%, and the percentage is further reduced as the size of the dataset grows. In fact, we have experimentally verified that we can consider any reasonable bound[1] in the number of instances without damaging the performance of the algorithm. Using a small percentage does not harm the estimation of the threshold of votes. With this method the complexity of this step is $O(1)$ as the number of instances used is bounded regardless the size of the dataset.

Finally, we consider the partition of the dataset apart from the algorithm as many different partition methods can be devised. The performed random partition is of complexity $O(n)$.

7. Conclusions

In this chapter we have shown two new methods for scaling up instance selection algorithms. These methods are applicable to any instance selection method without any modification. The methods consist of a recursive procedure, where the dataset is partitioned into disjoint subsets, an instance selection algorithm is applied to each subset, and then the selected instances are rejoined to repeat the process, and a democratic approach where several rounds of approximate instance selection are performed and the result is obtained by a voting scheme.

Using three well-known instance selection algorithms, ICF, RNN and a CHC genetic algorithm, we have shown that our method is able to match the performance of the original algorithms with a considerable reduction in execution time. In terms of reduction of storage requirements, our approach is even better than the use of the original instance selection algorithm over the whole dataset. Additionally, our method is straightforwardly parallelizable without modifications.

The proposed methods allow the application of instance selection algorithms to almost any problem size. The behavior is linear in the number of instances as it has been shown both theoretically and experimentally.

Furthermore, this philosophy can be extended to other learning algorithms such as feature selection or clustering, which means it is a powerful tool for scaling up machine learning algorithms.

8. References

Barandela, R., Ferri, F. J. & Sánchez, J. S. (2005). Decision boundary preserving prototype selection for nearest neighbor classification. *International Journal of Pattern Recognition and Artificial Intelligence*, vol. 19, no. 6, 787-806.

1 This *reasonable* bound can be from a few hundreds to a few thousands, even for huge datasets.

Baluja, S. (1996). Evolution of an Artificial Neural Network Based Autonomous Land Vehicle Controller. *IEEE Transactions on Systems, Man, and Cybernetics – Part B: Cybernetics, vol.* 26, no. 3, 450–463.

Brighton, H. & Mellish, C. (2002). Advances in instance selection for instance-based learning algorithms. *Data Mining and Knowledge Discovery*, vol. 6, 153-172.

Brodley, C. E. (1995). Recursive automatic bias selection for classifier construction. *Machine Learning*, vol. 20, no. 1/2, 63–94.

Cano, J. R., Herrera, F. & Lozano, M. (2003). Using evolutionary algorithms as instance selection for data reduction in KDD: An experimental study. *IEEE Transactions on Evolutionary Computation*, vol. 7, no. 6, 561-575.

Cano, J. R., Herrera, F. & Lozano, M. (2005). Stratification for scaling up evolutionary prototype selection. *Pattern Recognition Letters,* vol. 26, no. 7, 953–963.

Chaudhuri, S., Motwani, R. & Narasayya, V. (1998). Random sampling for histogram construction: How much is enough?. In: *Proceedings of ACM SIGMOD, International Conference on Management of Data*, ACM Press, Haas, L., and Tiwary, A., (Eds.), 436-447.

Chawla, N. W., Hall, L. O., Bowyer, K. W. & and Kegelmeyer, W. P. (2004). Learning Ensembles from Bites: A Scalable and Accurate Approach. *Journal of Machine Learning Research*, vol. 5, 421–451.

Chen, J. H., Chen, H. M. & Ho, S. Y. (2005). Design of nearest neighbor classifiers: multi-objective approach. *International Journal of Approximate Reasoning,* vol. 40, no. 1-2, 3–22.

Cochran, W. (1977). *Sampling techniques*, John Wiley & Sons, New York, USA.

Cover, T. M. & Thomas, J. A. (1991). *Elements of Information Theory*, John Wiley & Sons, Inc.

De Haro-García, A. & García-Pedrajas, N. (2009). A divide-and-conquer recursive approach for scaling up instance selection algorithms. *Data Mining and Knowledge Discovery*, vol. 18, 392-418.

Eshelman, L. J. (1990). *The CHC Adaptive Search Algorithm: How to Have Safe Search When Engaging in Nontraditional Genetic Recombination*, Morgan Kauffman.

García-Pedrajas, N., García-Osorio, C. & Fyfe, C. (2007): Nonlinear Boosting Projections for Ensemble Construction. *Journal of Machine Learning Research*, vol. 8, 1–33.

García-Pedrajas, N., Romero del Castillo, J. A. & Ortiz-Boyer, D. (2009). A cooperative coevolutionary algorithm for instance selection for instance-based learning, *Machine Learning*, in press.

Gates, G. W. (1977). The reduced nearest neighbor rule. *IEEE Transactions on Information Theory*, vol. 18, no. 3, 431-433.

Goldberg, D. E. (1989). Genetic Algorithms in Search, Optimization and Machine Learning, Addison—Wesley, Reading, USA.

Hettich, S., Blake, C.L. & Merz, C.J. (1998). UCI Repository of machine learning databases, University of California, Irvine, Dept. of Information and Computer Sciences.

Ishibuchi, H. & Nakashima, T. (2000). Pattern and Feature Selection by Genetic Algorithms in Nearest Neighbor Classification. Journal of Advanced Computational Intelligence and Intelligent Informatics, vol. 4, no. 2, 138-145.

Kim, S.-W. & Oommen, B. J. (2004). Enhancing Prototype Reduction Schemes With Recursion: A Method Applicable for "Large" Data Sets. *IEEE Transactions on Systems, Man, and Cybernetics--Part B: Cybernetics*, vol. 34, no. 3, 1384–1397.

Kivinen, J. & Mannila, H. (1994). The power of sampling in knowledge discovery, In: *Proceedings of the thirteenth ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems*, ACM Press, 77–85.

Kuncheva, L. (1995). Editing for the *k*-nearest neighbors rule by a genetic algorithm. *Pattern Recognition Letters*, vol. 16, 809-814.

Li, J., Manry, M. T., Yu, C. & Wilson, D. R. (2005). Prototype classifier design with pruning, *International Journal of Artificial Intelligence Tools*, vol. 14, no. 1-2, 261–280.

Liu, H. & Motoda, H. (2002). On issues of instance selection. *Data Mining and Knowledge Discovery*, vol. 6, 115-130.

Reeves, C. R. & Bush, D. R. (2001). Using genetic algorithms for training data selection in RBF networks. In: I*nstances Selection and Construction for Data Mining*, Liu, H. & Motoda, (Ed.), 339-356, Kluwer, Norwell, USA.

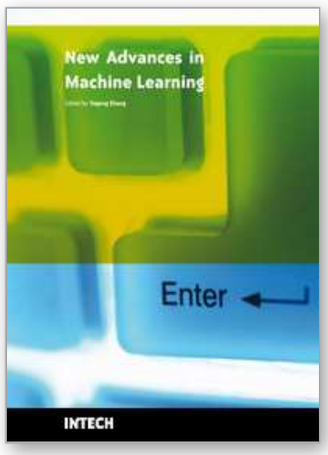
Smith, P. (1998). *Into Statistics*, Springer-Verlag, Berlin, Germany.

Son, S. H. & Kim, J. Y. (2006). Data reduction for instance-based learning using entropy-based partitioning, In: Proceedings *of the International Conference on Computational Science and Its Applications - ICCSA 2006*, Springer, 590–599.

Whitley, D (1986). The GENITOR Algorithm and Selective Pressure, In: Proceedings of the 3rd International Conference on Genetic Algorithms, 116–121, Morgan Kaufmann.

Wilson, D. R. & Martínez, T. R. (2000). Reduction techniques for instance-based learning algorithms. *Machine Learning*, vol. 38, 257-286.

Zhu, X., and Wu, X. (2006). Scalable representative instance selection and ranking. In: *Proceedings of the 18th International Conference on Patter Recognition (ICPR'06)*, IEEE Computer Society, 352 – 355.



New Advances in Machine Learning

Edited by Yagang Zhang

ISBN 978-953-307-034-6
Hard cover, 366 pages
Publisher InTech
Published online 01, February, 2010
Published in print edition February, 2010

The purpose of this book is to provide an up-to-date and systematical introduction to the principles and algorithms of machine learning. The definition of learning is broad enough to include most tasks that we commonly call “learning” tasks, as we use the word in daily life. It is also broad enough to encompass computers that improve from experience in quite straightforward ways. The book will be of interest to industrial engineers and scientists as well as academics who wish to pursue machine learning. The book is intended for both graduate and postgraduate students in fields such as computer science, cybernetics, system sciences, engineering, statistics, and social sciences, and as a reference for software professionals and practitioners. The wide scope of the book provides a good introduction to many approaches of machine learning, and it is also the source of useful bibliographical information.

InTech Europe
University Campus STeP Ri
Slavka Krautzeka 83/A
51000 Rijeka, Croatia
Phone: +385 (51) 770 447
Fax: +385 (51) 686 166
www.intechopen.com

InTech China
Unit 405, Office Block, Hotel Equatorial Shanghai
No.65, Yan An Road (West), Shanghai, 200040, China
中国上海市延安西路65号上海国际贵都大饭店办公楼405单元
Phone: +86-21-62489820
Fax: +86-21-62489821

IntechOpen