Int. J. Appl. Math. Comput. Sci., 2019, Vol. 29, No. 1, 151–168
DOI: 10.2478/amcs-2019-0012

amcs

ENSEMBLES OF INSTANCE SELECTION METHODS: A COMPARATIVE STUDY

MARCIN BLACHNIK [a]

[a]Department of Applied Informatics
Silesian University of Technology, Akademicka 2A, 44-100 Gliwice, Poland
e-mail: marcin.blachnik@polsl.pl

Instance selection is often performed as one of the preprocessing methods which, along with feature selection, allows a significant reduction in computational complexity and an increase in prediction accuracy. So far, only few authors have considered ensembles of instance selection methods, while the ensembles of final predictive models attract many researchers. To bridge that gap, in this paper we compare four ensembles adapted to instance selection: *Bagging*, *Feature Bagging*, *AdaBoost* and *Additive Noise*. The last one is introduced for the first time in this paper. The study is based on empirical comparison performed on 43 datasets and 9 base instance selection methods. The experiments are divided into three scenarios. In the first one, evaluated on a single dataset, we demonstrate the influence of the ensembles on the compression–accuracy relation, in the second scenario the goal is to achieve the highest prediction accuracy, and in the third one both accuracy and the level of dataset compression constitute a multi-objective criterion. The obtained results indicate that ensembles of instance selection improve the base instance selection algorithms except for unstable methods such as *CNN* and *IB3*, which is achieved at the expense of compression. In the comparison, *Bagging* and *AdaBoost* lead in most of the scenarios. In the experiments we evaluate three classifiers: *1*NN, *k*NN and SVM. We also note a deterioration in prediction accuracy for robust classifiers (*k*NN and SVM) trained on data filtered by any instance selection methods (including the ensembles) when compared with the results obtained when the entire training set was used to train these classifiers.

Keywords: machine learning, classification, instance selection, ensemble methods.

1. Introduction

A typical data mining process consists of four steps: data collection, data preprocessing, predictive modeling and, finally, post-processing. One of the key elements of the second stage (data preprocessing) is the selection of the training data. For example, Wilson (1972) showed that the correct selection of the training set samples (without noisy samples) significantly improves the accuracy of a classifier. A proper selection of the training set can be obtained using instance selection methods. These were initially designed to work with the nearest neighbor classifier (*1*NN), but are often used with other classifiers (García *et al*., 2016). Instance selection methods work as a data filter that, based on the built-in heuristic, evaluates which instances of the training set should be rejected and which should be kept to train the predictive model.

In the third stage, where the predictive model is built, ensembles of the base predictors are of particular importance (Woźniak *et al*., 2014). They help to increase prediction accuracy by grouping individual base expert models and make the final decision by collecting votes of the individual base models. Despite the great success of the ensemble methods at the predictive stage, their use at the preprocessing stage, especially in instance selection, has been modest. An increase in the development of instance selection ensembles can be observed in recent years (Arnaiz-González *et al.*, 2016a; García-Pedrajas and De Haro-García, 2014; García-Osorio *et al*., 2010).

However, there is currently no comprehensive study to compare different ensembles of instance selection methods except for boosting ones (García-Pedrajas and De Haro-García, 2014). This work bridges the gap by comparing various solutions, including *Bagging* and *Feature Bagging* based instance selection. It also introduces for the first time the method called *Additive Noise*, and all these solutions are compared to the instance selection ensemble based on *AdaBoost*. The methods presented in our study are evaluated on 43 datasets from the Keel

project repository using 9 different base instance selection methods. We analyze the ensembles in three scenarios. In the first one, we simply demonstrate on a single dataset how the ensembles influence the compression–accuracy relation, then in the second scenario we optimize ensembles to achieve a maximum prediction accuracy of the final predictor, and in the third scenario, both the criteria, the accuracy and training set size reduction, are evaluated simultaneously by the ensemble.

The next section describes instance selection methods, in particular those used in the experiments. Then ensemble methods are discussed, with the emphasis on the problem of adapting them to instance selection. The following section presents the experiments and the obtained results. The final section shortly concludes the paper, providing suggestions for the users applying ensembles of instance selection.

2. Basic concepts

The main purpose of this work is to compare various types of ensembles of instance selection. First, we provide an overview of instance selection methods and the state of the art of ensembles in application to predictive models. From these two we construct the ensembles of instance selection, which returns a weight vector indicating the importance of each training sample. This weight vector is then used with a procedure for determining an acceptance threshold to identify which instances should be kept and which should be removed. All these issues are described in this section.

2.1. Instance selection. The instance selection algorithm works as a data filter in the preprocessing stage, which prunes the training set before executing the learning process of the prediction model. It receives a dataset $\mathbf{T}$ and returns a dataset $\mathbf{P}$ such that $\mathbf{P} \subseteq \mathbf{T}$ and $\|\mathbf{P}\| \leq \|\mathbf{T}\|$. $\mathbf{T}$ consists of n pairs $\mathbf{T} = \{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \ldots, (\mathbf{x}_n, y_n)\}$, where each $\mathbf{x}$ is a vector in the domain X such that $\mathbf{x} \in X \subset \mathbb{R}^d$ and y takes on a value from among c symbols, i.e., $y \in \{s_1, s_2, \ldots, s_c\}$.

Instance selection is performed by removing the data samples (instances) which are inappropriate according to the implemented heuristics. Therefore one of the properties of instance selection is *compression* defined as

$$cmp = 1 - \frac{\|\mathbf{P}\|}{\|\mathbf{T}\|}. \tag{1}$$

At the beginning, the development of instance selection was aimed at improving the nearest neighbor's classifier (Hart, 1968; Wilson, 1972). This has led to the emergence of two families of instance selection methods, namely, *condensation methods*, which are responsible for limiting the set of reference objects, thus speeding up the decision making (these methods are characterized by large compression), and *noise filter methods*. The latter limit the effect of outliers and noisy samples also by eliminating the corresponding reference objects (usually compression is not too high). An interesting comparison of different instance selection methods can be found in the work of García *et al*. (2012).

It should be noted that a vast majority of algorithms are focused on classification problems, and only recently exploration of their modifications has begun, including regression tasks (Kordos and Blachnik, 2012; Arnaiz-González *et al.*, 2016b; Song *et al*., 2017) or stream analysis (Czarnowski and Jędrzejowicz, 2015; Shaker and Hüllermeier, 2012; Gunn *et al*., 2018). It is also worth mentioning that the use of instance selection methods is not limited to the nearest neighbor algorithm, but has also found applications to many other predictors. For example, Kordos and Rusiecki (2016) evaluated various approaches to MLP neural network training in the presence of noise. Their results pointed out that the *ENN* was among the best methods. García *et al*. (2016) considered instance selection as one of the most influential preprocessing methods.

The number of instance selection algorithms is very high: for example, García *et al*. (2012) compared over 40 algorithms and mentioned over 70 with their variants. Among them a group of 9 selected and the most commonly used instance selection algorithms are evaluated. When selecting the algorithms we tried to pick the most popular ones (see, e.g., García *et al*., 2016) which are often used by many authors (see also Jankowski and Grochowski, 2004; Wilson and Martinez, 2000), and those algorithms that were highlighted as outstanding by García *et al*. (2012). From the selected group we excluded methods which use meta-heuristics because they are characterized by long execution time, which limits their usability in the context of ensembles.

Below we provide a short description of each of the selected methods, while details can be found in the cited publications:

- *CNN*: condensed nearest neighbor rule (Hart, 1968) is an ancestor of all condensation methods. It tries to keep the performance of the original *1*NN by classifying all instances of the training set correctly. It starts by randomly selecting one representative instance per class and adds it to the reference set $\mathbf{P}$ (the dataset $\mathbf{T}$ remains unchanged while the algorithm works); then it starts the main loop, where each instance from $\mathbf{T}$ miss-classified by the nearest neighbor classifier trained on $\mathbf{P}$ is added to $\mathbf{P}$. The algorithm stops when all the instances in $\mathbf{T}$ are correctly classified.
- *IB3*: it was developed by Aha *et al*. (1991) as an

instance based learning algorithm, version 3. It is an extension of *IB2*, but uses a wait-and-see principle and validates statistical significance of removing or keeping an instance in $\mathbf{P}$. It starts with an empty set of selected instances $\mathbf{P}$ and then iterates over samples from $\mathbf{T}$. If an instance $\mathbf{x}$ is classified incorrectly by the nearest *acceptable instance* from $\mathbf{P}$, $\mathbf{x}$ is added to $\mathbf{P}$, and its classification record is updated (how many times a given instance was used to correctly classify other instances). Finally, the classification record is analyzed once more to see if any instance in $\mathbf{P}$ can be removed. To determine if an instance $\mathbf{p}$ from $\mathbf{P}$ is acceptable or should be removed, a significance test is used.

- *RNGE*: relative neighbor graph editing (Bhattacharya *et al*., 1984) builds a relative neighbor graph over the training data and then selects border samples and stores them in $\mathbf{P}$. Border samples are these instances which have at least one adjacent instance of another class. A relative neighbor graph is determined by validating the condition

$$\forall_{a\neq b\neq c} \|\mathbf{x}_a - \mathbf{x}_b\|_2 \leq \max\left(\|\mathbf{x}_a - \mathbf{x}_c\|_2, \|\mathbf{x}_b - \mathbf{x}_c\|_2\right) \quad (2)$$

between every three instances, where $\{\mathbf{x}_a, \mathbf{x}_b, \mathbf{x}_c\}$ are instances from $\mathbf{T}$ and $\|\cdot\|_2$ denotes the L_2 norm. The *RNGE* algorithm is an example of a larger group, which also includes Voronoi editing and Gabriel graph editing (*GGE*). As shown by Bhattacharya *et al*. (1984), the following relation takes place:

$$\mathbf{T} \supseteq \mathbf{P}_{\text{Voronoi}} \supseteq \mathbf{P}_{GGE} \supseteq \mathbf{P}_{RNGE}. \quad (3)$$

where $\mathbf{P}_{\text{Voronoi}}$ is the set of prototypes returned by the Voronoi tessellation and $\mathbf{P}_{GGE}$ is the set of prototypes returned by *GGE*. The main idea behind the whole family is to keep only these instances in $\mathbf{P}$ which have at least one nearest neighbor according to the graph structure from the opposite class; all other instances can be removed without affecting its performance.

- *ENN*: edited nearest neighbor (Wilson, 1972) is an ancestor of noise removal methods and is often used as an initial step before running condensation instance selection algorithms. It detects and removes all instances which may deteriorate the performance of the classifier. Initially $\mathbf{P} = \mathbf{T}$; then it analyzes the neighborhood of each query instance $\mathbf{x}$. If this instance is misclassified by its k neighbors, then it is marked for removal from $\mathbf{P}$. The actual removal is executed as a final step of the algorithm.

- *All-kNN* (Tomek, 1976) is an extension of the *ENN* algorithm where the *ENN* step is repeated for a range of $k = [k_{\min}, \ldots, k_{\max}]$ values. It is the first algorithm which can be recognized as an ensemble as it combines results for a set of individuals for different values of k.

- *Drop3* n: decremental reduction optimization procedure, version 3. In fact, it is a family of algorithms among which we selected *Drop3* as suggested by Wilson and Martinez (2000). This algorithm can be characterized as dropping instances from $\mathbf{P}$ while at the beginning of the algorithm $\mathbf{P} = \mathbf{T}$. *Drop* algorithms are based on the analysis of what would happen if an instance were removed: would then classification accuracy also drop? *Drop3* is a modification of *Drop2* where initially the *ENN* algorithm is executed to remove noise samples. In *Drop2* the effect of removal of an instance from $\mathbf{P}$ is validated using the entire training set $\mathbf{T}$, and the instances are analyzed in the order according to the distance to the nearest enemy (an enemy is an instance from a different class).

- *ICF*: iterative case filtering (Brighton and Mellish, 2002) is a two-step algorithm; first it applies the *ENN* algorithm to prune noisy samples, then in the second step it finds a *local set* $\phi(\mathbf{x})$ for every instance $\mathbf{x}_i$. The *local set* is defined by the largest hypersphere centered at $\mathbf{x}$, which includes only instances of the same class as $\mathbf{x}$. The *local set* is then used to calculate two statistics:

$$Coverage(\mathbf{x}) = \{\mathbf{x}' \in \mathbf{T} : \mathbf{x} \in \phi(\mathbf{x}')\}, \quad (4)$$
$$Reachability(\mathbf{x}) = \{\mathbf{x}' \in \mathbf{T} : \mathbf{x}' \in \phi(\mathbf{x})\}, \quad (5)$$

and if $Reachability(\mathbf{x}_i) > Coverage(\mathbf{x}_i)$, then the sample $\mathbf{x}_i$ is removed from $\mathbf{P}$.

- *HMN-E*: hit miss network editing (Marchiori, 2008) starts with constructing a hit miss network (HMN), i.e., a graph where each data sample $\{\mathbf{x}_i, y_i\}$ is a vertex V_i and edges point to the nearest neighbor of each class, so the out-degree of each vertex is equal to the number of classes. In addition, each vertex of HMN is represented by two counters, *hit* and *miss*. Denoting by $\{\mathbf{x}_j, y_j\}$ the nearest neighbor of instance i, the hit miss counter of V_j is increased if $y_i = y_j$ and $y_i \neq y_j$, respectively. The HMN is then used to select examples with four rules describing the relation between these counters.

- *CCIS*: class conditional instance selection (Marchiori, 2010) is an algorithm which consists of two steps. In the first one, denoted as the class conditional selection (CC), an HMN graph is constructed, which is then used with

the K-divergence: an instance scoring function. The obtained scores are evaluated first by discarding samples with a negative or zero score, and instances with the highest scores are iteratively added to the final subset $\mathbf{P}$. The loop ends when the empirical error increases. The first step (CC) is followed by thin-out selection (THIN), which selects instances close to the decision boundary of the *1*NN rule to further increase the compression rate.

2.2. Ensemble methods. The idea of the ensembles is to replace a single predictor with a group of experts, who decide together (Schapire, 1990; Jacobs *et al*., 1991; Wolpert, 1992; Rokach, 2009; Galar *et al*., 2011). In this scenario each basic expert casts a vote or votes, which are then collected and the final decision is made (see Fig. 1). This basic concept is applicable only when the results returned by the individual experts differ; otherwise, ensembles do not give any benefits. The diversity can be introduced by considering a mixture of prediction models (such as neural networks, decision trees and *k*NN), or by dataset modifications where each individual base model M_i is constructed using a modified training set. The modified training set ensures that even the same type of prediction model returns different results for different modifications. Basically, there are five scenarios of dataset modifications which are applicable: *Bagging*—sampling of the input data (Bauer and Kohavi, 1999), *Feature Bagging*—sampling of the feature space (Skurichina and Duin, 2001), *Additive Noise*—adding a noise component to the input space (Raviv and Intrator, 1996), *Labels Modifications*—appropriate class label encoding in multi-class classification problems and *Boosting*—sampling the input space according to the cost function distribution (Freund and Schapire, 1997).

2.3. Ensembles of instance selection. The construction of ensembles of instance selection is not a new concept, but it has been developed only by few authors. A general scheme of the ensembles of instance selection is shown in Fig. 2.

The first idea was proposed by Tomek (1976) and is based on collecting votes of the *ENN* model for a range of *k*-values. Another solution was to use the boosting algorithm for the instance selection, as shown by Sebban *et al*. (2002). An indirect solution was to use a combination of instance selections to build an ensemble of classifiers, where the diversity of the base classifiers is ensured by the variety of the datasets provided as the output of instance selection, as discussed by García-Pedrajas (2009). García-Osorio *et al*. (2010) proposed another variant of instance selection through the democratic vote of the classifiers, where the instances which are usually misclassified are marked for rejection.

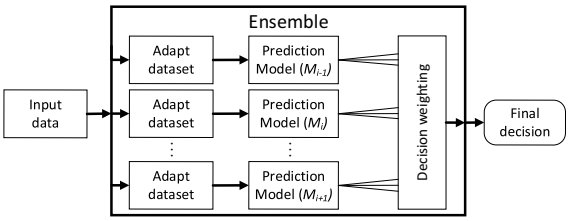


Fig. 1. General structure of ensemble methods.

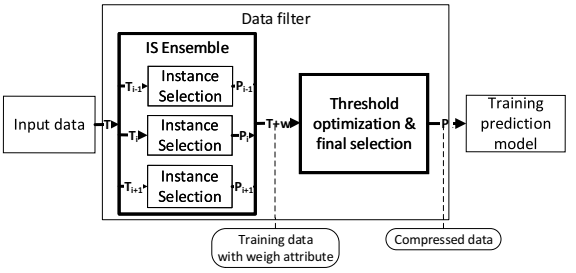


Fig. 2. Structure of an instance selection ensemble.

Blachnik and Kordos (2014) as well as Blachnik (2014) adapted two other ensembles for instance selection, namely, *Bagging* and *Feature Bagging*. Finally, perhaps the most advanced study in this area was conducted by García-Pedrajas and De Haro-García (2014), who compared different boosting methods in application to instance selection including *AdaBoost*, *FloatBoost, MultiBoost, ReweightBoost*. They also showed that the problem of ensembles of instance selection can be analyzed as an ensemble of a two-class classification problem, where each training instance is assigned to one of two categories: selected/removed, which we will call the *instance selection label*. However, the ensemble of instance selection differs from ensembles of classifiers. In the latter the output of individuals can be combined in different ways, for example, using the weighted vote, or combined by a higher level predictor as in *stacking* (Wolpert, 1992) (for a more complex voting scheme, the interested reader is referred to the works of Kuncheva *et al*. (2001) or Kuncheva (2004)).

In the case of ensembles of instance selection we do not know the true *instance selection label*, i.e., we do not know precisely which instances should be kept and which removed. We just have a collection of predicted *instance selection labels* and we cannot train the combiner until we execute a classifier which allows us to asses a given subset of instances. This leads to methods which do not require any supervision in building an ensemble, such as *Bagging* or *Feature Bagging*. Otherwise, a tool is needed which assesses the quality of the predicted *instance selection la-*

Algorithm 1. Ensemble of instance selection: *Additive Noise*.

Input: Training set $T = \{(\mathbf{x}_1, y_1), \ldots (\mathbf{x}_n, y_n)\}$, $\mathbf{x} \in \mathbb{R}^m$. l: number of iterations, σ: noise level.

Result: Instance weights: $\mathbf{w} = \{w_1, w_2, \ldots, w_n\}$

$\forall_{j=1,\ldots,n} \; w_j = 0$ /* Initialize instance weights */

for $i = 1 \ldots l$ **do**

  $\mathbf{T}'_i = \mathbf{T} + N(0, \sigma)$ /* Add d-dimensional Gaussian noise with $\mu = 0$ and given σ to each instance in $\mathbf{T}$ */

  $\mathbf{P}_i = \texttt{ISModel}(\mathbf{T}'_i)$ /* Run the base instance selection */

  $\forall_{j \in \mathbf{P}_i} \; w_j += 1/l$ /* Increase counter for selected instances */

end

return $\mathbf{w}$

Algorithm 2. Ensemble of instance selection: *Bagging* and *Feature Bagging*. The difference appears in sampling method ($\mathbf{T}'_i = \texttt{Sample}(\mathbf{T}, r)$).

Input: Training set $T = \{(\mathbf{x}_1, y_1), \ldots (\mathbf{x}_n, y_n)\}$, $\mathbf{x} \in \mathbb{R}^m$, l: number of iterations, r: instance sampling ratio.

Result: Instance weights: $\mathbf{w} = \{w_1, w_2, \ldots, w_n\}$

$\forall_{j=1,\ldots,n} \; w_j = 0$ /* Initialize instance weights */

for $i = 1, \ldots, l$ **do**

  $\mathbf{T}'_i = \texttt{Sample}(\mathbf{T}, r)$ /* Sample instances from $\mathbf{T}$ */

  $\mathbf{P}_i = \texttt{ISModel}(\mathbf{T}'_i)$ /* Run the base instance selection */

  $\forall_{j \in \mathbf{P}_i} \; w_j += 1/l$ /* Increase counter for selected instances */

end

return $\mathbf{w}$

bels, such as in *Boosting*, that incorporates a classifier to change the sampling distribution.

Drawing inspiration from classifier-ensembles, here we present one more instance-selection ensemble, which is based on *Additive Noise*. It is very similar to *Bagging* and *Feature Bagging*, but instead of sampling the dataset $\mathbf{T}$, the diversity is introduced by adding a Gaussian noise with $\mu = 0$ and given σ.

For the purpose of the study, four methods were evaluated: three which do not require any extra predictor to construct the ensemble, such as *Bagging*, *Feature Bagging*, *Additive Noise* and one which, belongs to the *Boosting* family. Here we use *AdaBoost*, which, according to García-Pedrajas and De Haro-García (2014), was among the top solutions. All these methods are sketched and discussed below:

- *Bagging* (see Algorithm 2), where each set of prototypes $\mathbf{P}_i$ is constructed using an instance selection base method denoted by $ISModel(\cdot)$ and the dataset $\mathbf{T}_i$ is generated as a random subset of the original training set $\mathbf{T}$; the data are sampled from a uniform distribution and each selected instance $\mathbf{p} \in \mathbf{P}$ has the same weight $w = 1/l$.

- *Feature Bagging* (see Algorithm 2), where the main difference appears only in the sampling scheme. Similarly, each set of prototypes $\mathbf{P}_i$ is obtained using also $ISModel(\cdot)$, but the dataset $\mathbf{T}_i$ is generated from the training set $\mathbf{T}$ by a random selection of a subset of attributes ($\mathbf{T}_i$ contains all n instances of $\mathbf{T}$). The attributes are sampled from a uniform distribution and each vote has the same weight $w = 1/l$.

- *Additive Noise* (see Algorithm 1), where each set of selected samples $\mathbf{P}_i$ is obtained using $ISModel(\cdot)$ and the dataset $\mathbf{T}_i$ is generated from the training set $\mathbf{T}$ by adding noise to each input instance and each vote has the same weight. As already mentioned, this method originates from the concept initially proposed for classifiers ensembles and described by Raviv and Intrator (1996).

- *Boosting* (see Algorithm 3), where each set of selected samples $\mathbf{P}_i$ is obtained using $ISModel(\cdot)$ and the dataset $\mathbf{T}_i$ is obtained from the training set $\mathbf{T}$ by selecting instances from the distribution derived from the probability of misclassification of an instance, so that the misclassified instances are more likely to be drawn. In addition, each instance in $\mathbf{P}_i$ casts a vote weighted by the accuracy of that classifier. Perhaps the best known implementation of the boosting algorithm is *AdaBoost* (Freund *et al*., 1996) and its variants described by Zhu *et al*. (2009), which were adapted to the problem of instance selection by García-Pedrajas and De Haro-García (2014). In the experiments, $1NN(\mathbf{P}_i)$ was used to determine the weight of votes.

Note that, according to Fig. 2, all of the described solutions return as an output the entire training set $\mathbf{T}$ with an extra attribute representing the collected votes. Thus each training instance is annotated by the value which describes how often it was selected by the ensemble member or, in other words, how often it occurs in the selected samples $\mathbf{P}_i$. Here this value is called the *weight* (w_i: weight value of instance i) and allows ordering instances from the least influential, selected by none of the ensemble members, up to the most influential (selected

by all members). Note that the final subset of instances needs to be determined by some threshold value denoted as θ above which all instances are selected ($\mathbf{P} \subset \mathbf{T} \leftrightarrow (\mathbf{x}_i, y_i) \in \mathbf{T}, w_i > \theta$). This aspect is analyzed in the following section.

2.4. Optimization of the threshold. The selection of the appropriate acceptance threshold has critical impact on the final results. It requires a judge who rates various thresholds and determines the best one. It is usually implemented with a classifier that for each threshold θ returns a performance achieved for given subset of instances. Note that the proper estimation of the accuracy requires an internal cross-validation procedure. Here, in the ensembles of instance selection, usually *1*NN is used as it does not require any parameter tuning and can be efficiently implemented.

The set of thresholds considered $\Theta = \{\theta_1, \theta_2, \ldots, \theta_z\}$ can be determined in various ways. A naive solution is to manually set up thresholds, for example, $\Theta = \{0.2, 0.4, 0.6, 0.8\}$, and then conduct the assessment. This solution was used, e.g., by Blachnik and Kordos (2014) or Blachnik (2014), but it can lead to nonoptimal results, since the distribution of votes is nonlinear and varies between individual algorithms (e.g., see Fig. 3).

García-Pedrajas and De Haro-García (2014) proposed a better solution, where all possible threshold values were analyzed; the collection of votes was sorted and for each unique value of weight the performance was evaluated. This solution is appropriate for methods which cast equal vote weights, such as *Bagging*, *Feature Bagging* and *Additive Noise*, because the number of unique values is at most equal to that of ensemble models.

However, for boosting methods, such a solution may lead to a high increase in computational cost, since each component model returns a different weight. In this case, each instance may have a different weight. Hence the analysis requires n threshold evaluations (n is the number of training samples). Therefore, in this paper if the number of possible thresholds exceeds that of individual ensemble members (as for the *AdaBoost* algorithm), we propose discretization of the weight attribute to limit the number of bins to that of ensemble models. For this task, we use clustering based discretization with the fuzzy C-means (FCM) algorithm (Bezdek *et al.*, 1984). Clustering based discretization avoids the problem of grouping similar objects into two different discretization bins (this problem is an issue in equal-width and equal-frequency discretization), therefore assuring much more accurate results. The last problem within the threshold optimization procedure, which needs to be defined, is the performance measure, which considers compression and prediction accuracy together. As these two objectives are frequently contradictory, i.e., high accuracy requires more training samples and high compression may cause loss of accuracy, a compromise can be defined as

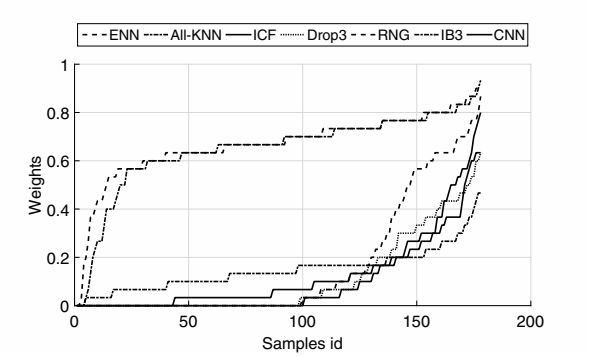


Fig. 3. Example of instances weight distribution returned by the *Bagging* ensemble for all of the base methods. Results were obtained for the well-known *wine* dataset.

$$perf = \alpha \times acc + (1 - \alpha) \times cmp, \tag{6}$$

where cmp is compression measure (see (1)), acc is any classification accuracy measure such as AUC, mean recall or mean precision, although here in the paper we use the standard definition:

$$acc = \frac{\#\text{correctly classified instances}}{\#\text{all classified instances}}, \tag{7}$$

and α is the trade-off coefficient, which determines the relevance or influence of the individual components.

2.5. Other issues with ensembles of instance selection. Two more problems facing ensembles of instance selection or, more precisely, of the *Data Filtering* process (see Fig. 2) are discussed in this section. The first one is computational complexity. As for just the first three methods, namely, *Bagging*, *Feature Bagging* and *Additive Noise*, they can be easily parallelized without any memory or time side effects. In other words, using concurrent data access, the memory usage is constant and, similarly, the time does not change, because the only overhead appears in collecting and aggregating votes cast by individuals, which is a very cheap procedure (summation over weights returned by the individuals).

In the case of a single processor environment the execution time is $O(l \times O_{IS} + O_{Agg})$, where l is the size of the ensemble and O_{IS} is the computational complexity of the base instance selection algorithm (see Table 1) and O_{Agg} is the aggregation time. On the other hand, in a multiprocessor environment with shared dataset access, the instance selection stage can be parallelized into l processors; then the execution time is $O(O_{IS} + O_{Agg} + c)$ where c represent a constant responsible for parallel environment preparation, but since $O_{IS} \gg O_{Agg} \gg c$, we obtain $O(O_{IS})$.

Table 1. Computational complexity of the evaluated instance selection algorithms. Note that the theoretical values can be reduced by distance caching and spatial data structures such as KD-Tree, Locality Sensitive Hashing, etc.

CNN	*IB3*	*RNGE*	*ENN*	*All-kNN*	*Drop3*	*ICF*	*HMN-E*	*CCIS*
n^3	n^2	n^3	n^2	n^2	n^3	n^3	n^2	n^2

Another situation appears in the case of *AdaBoost*, because here the iterations are mutually dependent, so the next iteration relies on the results of the previous one, meaning it cannot be parallelized. Moreover, *AdaBoost* also requires the built-in classifier to be executed to adapt the distribution weights. This leads to an execution time of order $O(l \times (O_{IS} + O_P) + O_{Agg})$, where O_P is the execution time of the predictor and l now represents the number of iterations. In the case of the *1*NN classifier it leads to $O(l \times (O_{IS} + n^2))$.

The total computational complexity of the *Data Filtering* process also involves a threshold optimization procedure. This stage can be very time consuming because it requires a classifier to be executed for each acceptance threshold θ. It even increases when the model is sensitive to hyper-parameters, which requires precise tuning. For example, it appears for *k*NN, SVM or neural networks. In this case the computational complexity of the threshold optimization procedure can be expressed as $O(l \times \epsilon \times \omega \times O_P)$, where l[1] is the size of Θ , ϵ represents the number of repetitions of the cross-validation procedure (usually $\epsilon = 10$), ω is the number of evaluated parameters and O_P is the computational complexity of the final predictor. To reduce the computational complexity, we use *1*NN, which can be very efficiently implemented and does not require any parameter tuning. A naive implementation leads to $O(l \times 10 \times n^2)$.

This analysis illustrates that proper implementation of instance selection ensembles horizontally scales very well and on multi-core processors using a computer cluster can be used without significant time drawbacks.

The second problem is the size of the ensemble. In contrast to the ensemble of classifiers, here the size plays a less important role. It is responsible for collecting enough votes to stabilize the instance weight distribution returned by the ensemble, so that the order of instances sorted according to this weight does not change or changes insignificantly. This ranking (results of the sorting) is then used by the threshold optimization procedure, which finally determines which instances to keep and which to reject. This is different from the ensemble of classifiers, since in *Bagging*, *Feature Bagging*, *Additive Noise* and *AdaBoost* the final classification directly depends on the collected votes and thus on the size of the ensemble—there is no threshold optimization step.

[1] Above, l was used to represent the size of the ensemble, but according to the threshold optimization procedure the size of the ensemble is equivalent to the number of thresholds, so we use the same symbol.

3. Experimental setup

For a fair comparison of the described solutions, a number of experiments were carried out on the data sets available in the Keel project repository (Alcalá-Fdez *et al.*, 2011). For this purpose, 43 datasets containing only numerical attributes were selected from the repository. This allowed Euclidean distances to be used in the experiments without the need for special preprocessing or the application of heterogeneous distance measures. A description of the data sets is given in Table 2.

The experimental procedure consisted of dataset preparation, including normalizing attributes to the range $[0, 1]$, and then performing the 10-fold cross-validation test. Within the cross-validation, first the process of instance selection on the training set was performed. At the output it generated an attribute called the *weight*, which describes the frequency of selection of the individual instances, so each instance of the training set was assigned a weight value representing how often it was selected by the internal methods of the ensemble, and then the weights were normalized in the range $[0, 1]$. The next step was to optimize the acceptance threshold as described in Section 2.4. In that optimization process, internal cross-validation was used with the *1*NN classifier as discussed in Section 2.5. As a result of the threshold optimization procedure, all the instances which had higher *weight* than the acceptance threshold θ were used to train the final classifiers.

Here we evaluated *1*NN, *k*NN and Gaussian SVM. The *1*NN classifier was used as it is the default model used in conjunction with instance selection algorithms and it is also characterized by a high evaluation speed (it does not require any parameter optimization), while *k*NN and SVM require careful parameter optimization, which was achieved using grid search with an internal cross-validation procedure. Finally, the best prediction model obtained from the parameter optimization procedure was applied to the test dataset of the outer cross-validation test and the results were expressed by the mean value and the standard deviation of both accuracy and compression.

The procedure is visualized in Fig. 4. Figure 4(a) shows the main process and Fig. 4(b) represents the internal cross-validation procedure used to optimize the hyper-parameters of the classifier. For all ensembles, 30

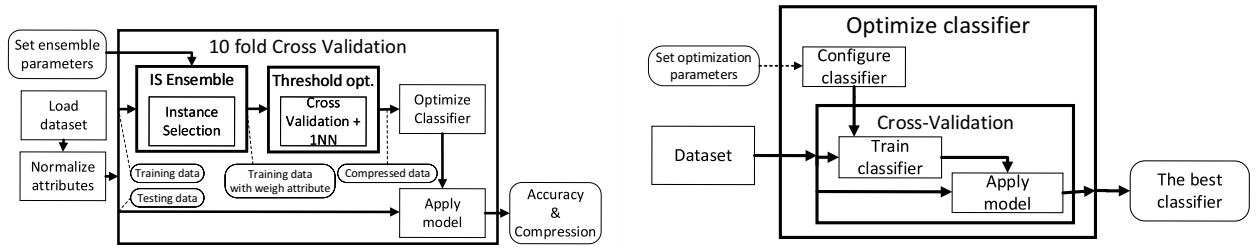


Fig. 4. Scheme of the process used in the experiments including the classifier optimization procedure: main process (a), classifier optimization procedure (b).

Table 2. Datasets used in the experiments with basic statistics.

id	Dataset (T)	# Samples	# Attributes	# Classes
1	appendicitis	106	7	2
2	balance	625	4	3
3	banana	5300	2	2
4	bands	365	19	2
5	bupa	345	6	2
6	cleveland	297	13	5
7	coil2000	9822	85	2
8	glass	214	9	6
9	haberman	306	3	2
10	hayes-roth	160	4	3
11	heart	270	13	2
12	hepatitis	80	19	2
13	ionosphere	351	33	2
14	iris	150	4	3
15	led7digit	500	7	10
16	letter	20000	16	26
17	magic	19020	10	2
18	mammographic	830	5	2
19	marketing	6876	13	9
20	monk-2	432	6	2
21	movement_libras	360	90	15
22	newthyroid	215	5	3
23	optdigits	5620	64	10
24	page-blocks	5472	10	5
25	penbased	10992	16	10
26	phoneme	5404	5	2
27	pima	768	8	2
28	ring	7400	20	2
29	satimage	6435	36	6
30	segment	2310	19	7
31	sonar	208	60	2
32	spambase	4597	57	2
33	spectfheart	267	44	2
34	tae	151	5	3
35	texture	5500	40	11
36	thyroid	7200	21	3
37	titanic	2201	3	2
38	twonorm	7400	20	2
39	vehicle	846	18	4
40	wdbc	569	30	2
41	wine	178	13	3
42	wisconsin	683	9	2
43	yeast	1484	8	10

component base models were evaluated. It is enough to stabilize the weight distribution of the base methods. An overview of the parameter settings is shown in Table 3.

For the calculations, the *RapidMiner* software with the *Information Selection* extension (developed by the author of this paper) was used. The implementations of the *HMN-E* were imported from the *WekaIS* package developed by Arnaiz-González *et al.* (2016b) and *CCIS* from the Keel project (Herrera, 2005).

The results of the experiments are presented in tabular and graphical form. The statistical significance of the results was verified using the Friedman rank test to check the significance of the difference between results and then the Wilcoxon rank test to compare pairs of algorithms, in particular the base instance selection algorithm with the ensemble methods. In addition, the Bonferroni–Holm procedure (Abdi, 2010) was used to fix the problem of multiple comparisons. All tests were performed for a significance level of 0.05. The detailed results are available at `www.prules.org/materials/isensemble`.

4. Results and a discussion

The conducted experiments were divided into three stages according to the evaluation criteria. The purpose of the first stage was to indicate the influence of the threshold θ on the relationship between compression and accuracy. This analysis was carried out for a single dataset, for which a front of the obtained solution was visualized on the compression–accuracy plot, referring it to the base instance selection method and the solution obtained by a classifier trained on the dataset without any instance selection.

The other two stages were carried out on the entire collection of the data sets to assess the quality of each of the ensemble methods considering Eqn. (6) as the overall performance measure, which combines prediction accuracy and compression into a single objective function. In the first group of experiments, the ensemble was

Table 3. Method configuration parameters.

Method	Parameters
CNN	-
IB3	$k = 3, \sigma_{\min} = 0.7, \sigma_{\max} = 0.9$
RNGE	-
Drop3	$k = 3$
ICF	$k = 3$
ENN	$k = 3$
All-kNN	$k_{\min} = 3, k_{\max} = 5$
AdaBoost	-
Bagging	$r = 0.9$
Feature Bagging	$r = 0.95$
Additive Noise	$\sigma = 0.1$
*k*NN	$k \in [1, 3, 4, 5, 6, 9, 11, 13, 17, 20]$
SVM	$\gamma \in [0.1, 0.3, 0.5, 0.7] \times 10^{-1}, C \in 10^{[-3,\ldots,2]}$

optimized to achieve the highest prediction accuracy ($\alpha = 1$ in (6)), but the compression was also recorded. The second group of experiments was aimed at analyzing how changing parameter α influences the output of the performance measure (6). In the second scenario the threshold optimization procedure took into account both compression and prediction accuracy. In the second procedure we evaluated only the results of the *1*NN classifier as the other methods behave similarly. Also, we wanted to reduce the computational effort because the parameter optimization procedure for *k*NN and SVM is computationally expensive and we needed to repeat it for every value of α.

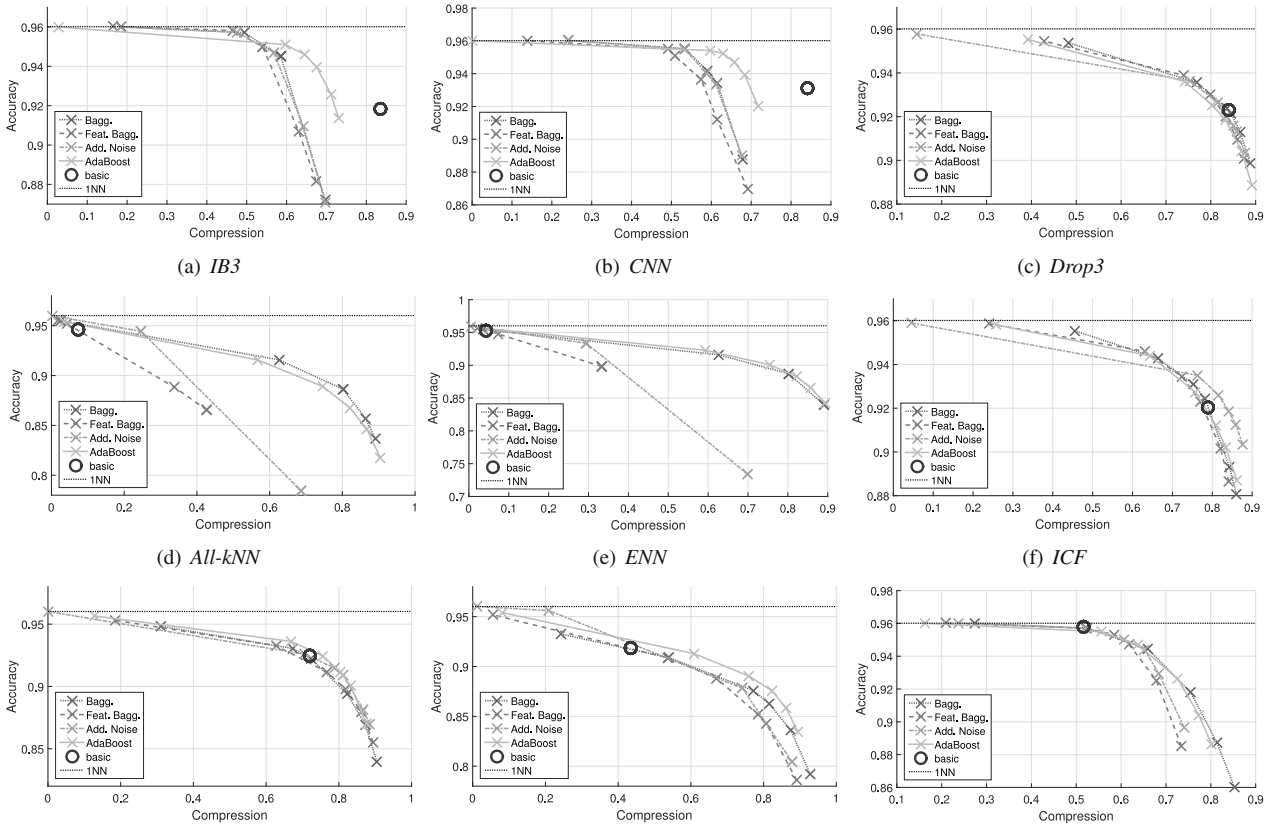
4.1. Scenario 1. The aim of the first scenario of the experiments is to show the influence of the threshold θ on the compression—accuracy relation, and to refer this solution to the results obtained by base instance selection. For that purpose we selected the *letter* dataset, which is the largest of the evaluated datasets, and executed the experiments, where for each acceptance threshold we recorded the accuracy of the *1*NN classifier and compression, which allowed as to draw solution curves also called front. The obtained results are shown in Fig. 5.

Analysis of the results indicate that, in most cases, the shape of the solution curve has a relatively flat segment for low compression values (the derivative close to zero), followed by an inflection section, ending with a high drop in accuracy with a relatively small decrease in compression (high value of the module of the derivative). This shape of the curve is desirable and expected, although it depends on the location of the base solution. If the latter is outside the curve at a large distance to the right as for *IB3* and *CNN*, the ensembles may cause a significant reduction in compression with no gain in accuracy.

The benefits appear when one maximizes only prediction accuracy; then for these two methods we observe a significant accuracy gain. If the solution of the base instance selection lies near the curves, as in the remaining majority, then we can distinguish the following three scenarios. When the base solution lies in a steep section, as for *Drop3*, *ICF*, then with a small loss of compression we can expect a relatively significant improvement in accuracy. If, on the other hand, the solution is in a flat section, such as for *ENN* or *All-kNN*, then with a small decrease in accuracy we can expect a significant improvement in compression. Finally, if the base solution lies at the beginning of the inflection point, as for *RNGE* or *CCIS*, then this solution can be treated as optimal from the point of view of a given base method. The worst scenario occurs when the relationship between compression and accuracy is linear with a medium angle of the slope, such as for *All-kNN* and *ENN*, for *Feature Bagging*; then we observe low usefulness of a given ensemble method.

It should be noted that these results were obtained for a single dataset, so for the purpose of generalization we carried out the following two scenarios of experimental research.

4.2. Scenario 2. As mentioned above, the experiments of *Scenario 2* were conducted to compare different ensemble methods in terms of maximum prediction accuracy. This scenario imitates use-cases where instance selection is employed to boost the accuracy of the



(a) *IB3* (b) *CNN* (c) *Drop3*

(d) *All-kNN* (e) *ENN* (f) *ICF*

(g) *CCIS* (h) *HMN-E* (i) *RNGE*

Fig. 5. Influence of the acceptance threshold of the ensembles on the relation between compression and accuracy for the *letter* dataset for the base methods and the *1*NN classifier.

prediction model. For this purpose, we compared the basic instance selection method (without an ensemble), the four ensembles methods discussed earlier and the results obtained for the classifier trained without instance selection. The experiments were repeated for 9 different instance selection methods described in Section 2.1 and three classifiers: *1*NN, *k*NN and SVM. Separately for each *basic* instance selection, the obtained results for each dataset were ranked from the worst ($rank = 1$) to the best ($rank = 5$) and then averaged. This allows us to fairly assess and compare the ensembles—see Table 4.

Similarly to ranking, for each ensemble and base instance selection method we averaged the accuracies obtained over all evaluated datasets. This shows how much the results differ. The compression depicted in that table is independent of the classifier and constant for each basic instance selection—see Fig. 4. Note that in this scenario the threshold of the ensemble was optimized to achieve the highest prediction accuracy ($\alpha = 1$), so the compression was ignored in threshold optimization. In Table 4 the row denoted as *Average* represents the average over values in columns.

For statistical comparison, first, we used the Friedman test to indicate if there are differences among the types of ensembles on the evaluated datasets. Then the results were evaluated with the Wilcoxon rank test, and the methods which were statistically significantly better than the reference base instance selection method are marked with the $(+)$ sign while significantly worse results with $(-)$ sign (including the Bonferroni–Holm correction). Similarly, we performed a significance test against the classifier trained on the entire training set $\mathbf{T}$ to indicate if instance selection and the ensembles significantly improved ($\oplus$ sign) or worsened ($\ominus$ sign) the results.

The results presented in Table 4 are also shown in Fig. 6. The x-axis represents the gain in terms of average compression, and the gain in terms of average prediction accuracy is shown on the y-axis. The gain means the difference between the results obtained for a given ensemble method (R_{Ens}) and the reference base method (R_{Base}), so it takes the form ($R_{Ens} - R_{Base}$), where R denotes accuracy and compression, for the x-axis and the y-axis, respectively. Different symbols in the figure correspond to different classifiers, so the crosses represent results obtained for *1*NN, circles for *k*NN and squares for SVM. We placed all of the evaluated classifiers into a single graph to make the comparison simpler. As

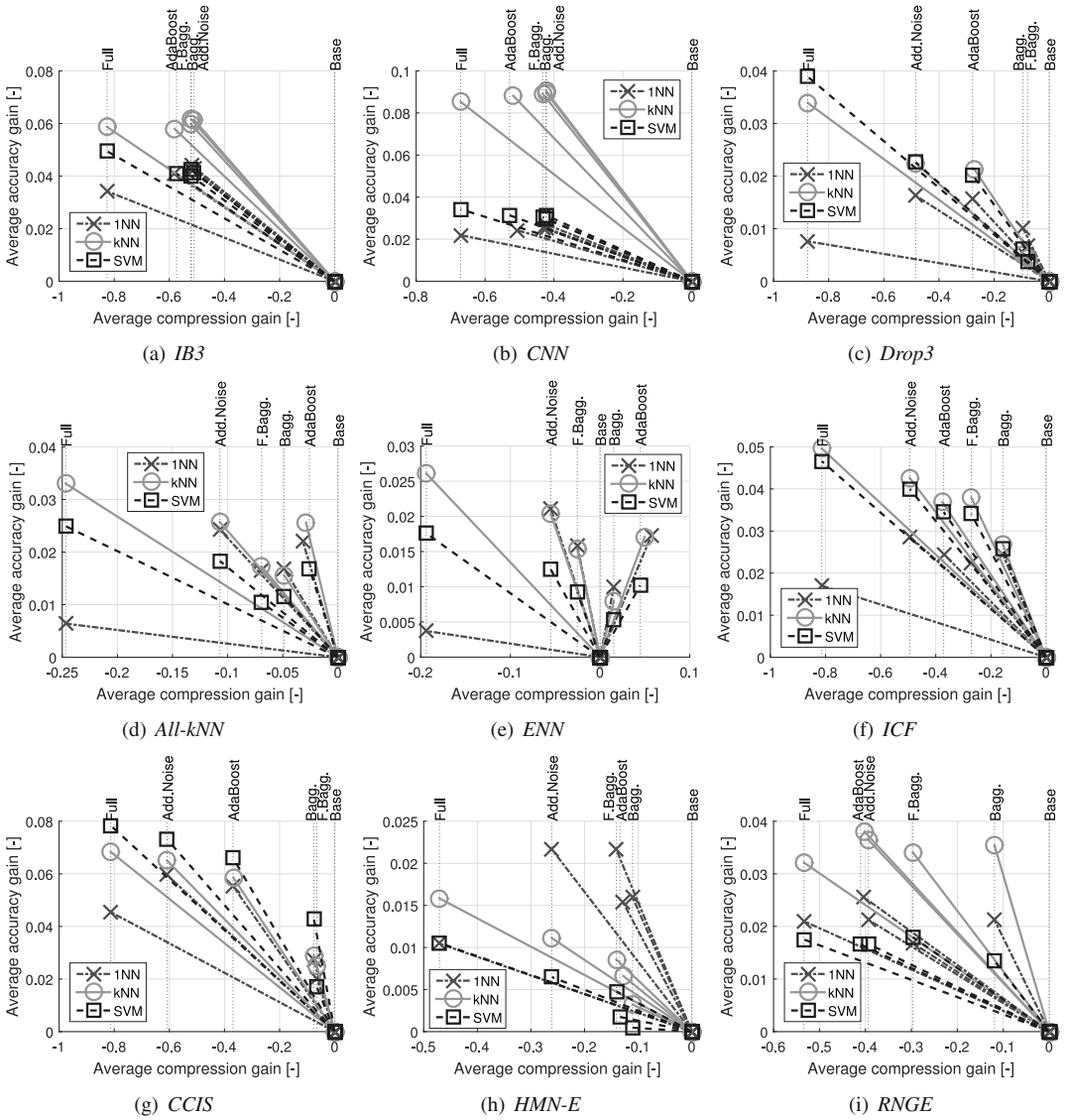
Table 4. Average ranks and mean of prediction accuracy and compression for the ensembles and the base models for the *1*NN, *k*NN and SVM classifiers. The rows denoted as *Average* represent the means of the above values for a given ensemble.

		base			*Bagging*			*Feature Bagging*			*Additive Noise*			*AdaBoost*			**T**		
		Rnk	$\overline{acc}$		Rnk	$\overline{acc}$		Rnk	$\overline{acc}$		Rnk	$\overline{acc}$		Rnk	$\overline{acc}$		Rnk	$\overline{acc}$	
*1*NN	*IB3*	1.48	0.77	⊖	3.98	0.81	+	4.05	0.81	+	3.70	0.81	+	4.02	0.81	+	3.78	0.80	+
	CNN	1.26	0.78	⊖	3.97	0.81	+	3.91	0.81	+	3.80	0.81	+	3.72	0.81	+	4.35	0.80	+
	ICF	2.24	0.79		3.84	0.81	+	3.64	0.81	+	3.94	0.82	+	3.80	0.81	+	3.53	0.80	
	Drop3	2.70	0.80		3.73	0.81	+	3.51	0.81		3.86	0.81		3.55	0.81		3.65	0.80	
	RNGE	1.64	0.78	⊖	2.95	0.80	+⊖	3.16	0.80	'+⊖	4.21	0.80	'+⊖	4.27	0.81	+	4.77	0.80	+
	All-kNN	2.70	0.80		3.77	0.81	+	3.76	0.81	+	3.98	0.82		3.56	0.82	+	3.24	0.80	
	ENN	2.83	0.80		3.55	0.81		4.03	0.82	+	4.02	0.82		3.44	0.82		3.13	0.80	
	HMN-E	2.60	0.79		3.65	0.81	+	4.14	0.82	+	3.66	0.82		3.43	0.81		3.51	0.80	
	CCIS	2.08	0.76	⊖	3.51	0.80	+	3.42	0.81	+	4.23	0.82	+	3.94	0.81	+	3.81	0.80	+
	Average	2.17	0.79		3.66	0.81		3.74	0.81		3.93	0.81		3.75	0.81		3.75	0.80	
*k*NN	*IB3*	1.33	0.77	⊖	3.77	0.83	+	4.13	0.83	+	4.01	0.83	+	3.55	0.83	+	4.22	0.83	+
	CNN	1.33	0.75	⊖	4.17	0.84	+⊕	3.97	0.84	+	3.98	0.84	+	3.90	0.83	+	3.66	0.83	+
	ICF	1.93	0.78	⊖	3.24	0.81	+⊖	3.51	0.82	+⊖	4.00	0.82	+⊖	3.49	0.82	+⊖	4.83	0.83	+
	Drop3	2.06	0.80	⊖	3.30	0.81	+⊖	3.50	0.81	+⊖	3.71	0.82	+⊖	3.60	0.82	+⊖	4.83	0.83	+
	RNGE	1.53	0.80	⊖	3.49	0.83	+	3.83	0.83	+	4.08	0.84	+	4.35	0.84	+	3.72	0.83	+
	All-kNN	2.30	0.80	⊖	3.51	0.81	+	3.55	0.82	+	3.62	0.82	+	3.77	0.82	+	4.26	0.83	+
	ENN	2.59	0.81	⊖	2.94	0.81		3.78	0.82	+	3.88	0.83		3.45	0.82		4.35	0.83	+
	HMN-E	2.72	0.82		3.29	0.82		3.52	0.82		4.20	0.83		3.16	0.82		4.10	0.83	
	CCIS	2.12	0.76	⊖	2.98	0.81	+⊖	3.38	0.81	+⊖	4.28	0.83	+	3.50	0.82	+⊖	4.74	0.83	+
	Average	1.99	0.79		3.41	0.82		3.68	0.82		3.97	0.83		3.64	0.83		4.30	0.83	
SVM	*IB3*	1.23	0.80	⊖	4.12	0.84	+	3.74	0.84	+	3.62	0.84	+	3.87	0.84	+	4.42	0.85	+
	CNN	2.06	0.81	⊖	3.43	0.84	+	3.79	0.84	+	4.02	0.84	+	3.84	0.84	+	3.86	0.85	+
	ICF	2.09	0.80	⊖	3.51	0.83	+⊖	3.34	0.83	+⊖	3.79	0.84	+	3.77	0.83	+	4.50	0.85	+
	Drop3	2.27	0.81	⊖	2.98	0.82	+⊖	3.49	0.82	+⊖	3.59	0.83	⊖	3.52	0.83	+⊖	5.15	0.85	+
	RNGE	2.23	0.83	⊖	3.36	0.84	+	4.01	0.85	+	3.71	0.85	+	3.91	0.85	+	3.78	0.85	+
	All-kNN	2.23	0.82	⊖	3.45	0.83	+⊖	3.63	0.83	+⊖	3.70	0.84	+	3.37	0.84	+⊖	4.62	0.85	+
	ENN	2.92	0.83	⊖	2.95	0.83	⊖	3.43	0.84	⊖	3.73	0.84		3.58	0.84	⊖	4.38	0.85	+
	HMN-E	3.07	0.84		3.07	0.84	⊖	3.30	0.84		3.65	0.84		3.55	0.84		4.36	0.85	
	CCIS	1.91	0.77	⊖	2.92	0.82	+⊖	3.16	0.83	+⊖	4.27	0.84	+	3.97	0.83	+⊖	4.78	0.85	+
	Average	2.22	0.81		3.31	0.83		3.54	0.84		3.79	0.84		3.71	0.84		4.43	0.85	
Compression	*IB3*	6.00	0.82	⊕	3.63	0.30	-⊕	3.16	0.31	-⊕	3.53	0.32	-⊕	3.55	0.24	-⊕	1.13	0.00	-
	CNN	5.95	0.67	⊕	3.91	0.25	-⊕	3.16	0.24	-⊕	3.67	0.25	-⊕	3.22	0.16	-⊕	1.08	0.00	-
	ICF	5.77	0.81	⊕	4.58	0.66	-⊕	3.65	0.54	-⊕	2.72	0.32	-⊕	3.22	0.44	-⊕	1.06	0.00	-
	Drop3	5.86	0.88	⊕	4.44	0.72	-⊕	4.12	0.70	-⊕	2.56	0.39	-⊕	3.01	0.60	-⊕	1.01	0.00	-
	RNGE	5.70	0.53	⊕	4.98	0.41	-⊕	3.44	0.24	-⊕	2.74	0.14	-⊕	3.00	0.13	-⊕	1.14	0.00	-
	All-kNN	4.99	0.25	⊕	4.00	0.20	-⊕	3.45	0.18	-⊕	2.88	0.14	-⊕	4.65	0.22	-⊕	1.02	0.00	-
	ENN	4.27	0.19	⊕	4.40	0.21	-⊕	3.27	0.17	-⊕	2.95	0.14	-⊕	5.09	0.25	-⊕	1.02	0.00	-
	HMN-E	5.26	0.47	⊕	4.14	0.36	-⊕	3.81	0.33	-⊕	2.74	0.21	-⊕	4.03	0.34	-⊕	1.01	0.00	-
	CCIS	5.74	0.81	⊕	4.44	0.65	-⊕	4.09	0.62	-⊕	2.30	0.20	-⊕	3.33	0.44	-⊕	1.09	0.00	-
	Average	5.50	0.60		4.28	0.42		3.57	0.37		2.90	0.23		3.68	0.31		1.06	0.00	

each graph represents the gains, the origin corresponds to the base instance selection method. This makes it easier to interpret, as positive values mean an improvement while negative represent a deterioration in comparison with the reference base method.

The summary of Table 4 is depicted in Fig. 7. The bar plot shows the average rank values from Table 4 (the higher, the better). The highest bars in that graph appear for classifiers trained without instance selection. In general, this confirms the results obtained by Jankowski and Grochowski (2004) as well as Grochowski and Jankowski (2004) who also reported the problem of performance degradation of robust classifiers trained on the data filtered by instance selection. However, the ensembles often allow to keep the performance not significantly different. The exception is *Drop3* and *ICF*, where the results over all evaluated datasets were significantly worse than the reference classifiers SVM and *k*NN trained on the entire training set (see the ⊖ marks). For SVM, also *CCIS*, *ENN* and *All-kNN* could not be improved by the ensemble and achieve significantly worse results than the reference classifier. In these cases, only *AdaBoost* worked correctly, leading to insignificantly worse results (note that the threshold optimization procedure did not allow including the entire training set).

The only exception is the *1*NN classifier, for which ensembles allow keeping the same accuracy as the classifier trained on the entire training set, but reducing the size of the stored samples. A comparison of



(a) *IB3* (b) *CNN* (c) *Drop3*

(d) *All-kNN* (e) *ENN* (f) *ICF*

(g) *CCIS* (h) *HMN-E* (i) *RNGE*

Fig. 6. Gain plot showing the difference between the results obtained for the ensembles (including results obtained for the classifier trained on the entire training set) and the reference results obtained by the base instance selection method. The values on the x-axis represent the difference in terms of mean compression, and those on the y-axis represent the differences in terms of mean accuracy $\overline{R}_{Ens.} - \overline{R}_{base}$. The gray shades represent results obtained for the evaluated classifiers.

the ensembles and base instance selection indicates a significant improvement in the ranks of accuracy over base instance selection. This is confirmed by the significant tests (see the $+$ marks) and nicely visualized in the graphs of Fig. 6, and in Table 4. Note that for *IB3* and *CNN* (nonstable methods) the results of the ensembles are not significantly different from those of the reference classifiers trained without instance selection. This is due to a very low compression (see Fig. 6) rate, which means that almost all samples are included in the training set P.

Considering the compression, according to the average rank shown in Fig. 8 the worst results were obtained by *Additive Noise* and the best ones by the base instance selection algorithms. In almost all cases the compression obtained by the ensembles was significantly worse than for the base method. Only *AdaBoost* for *ENN* and *All-kNN* as well as *Bagging* for *ENN* allowed an improvement of the compression, but not significant.

Finally, an overall comparison is given in Fig. 9, where all methods are combined in a single plot for the *1*NN classifier. The values represent the mean values over ranks obtained for each dataset, for which all the base methods with the ensembles were ranked at once. From these results we can see that the highest accuracy is obtained with the *Additive Noise* method, which wraps *ENN* and *All-kNN* base instance selection. In the bottom

Algorithm 3. Ensemble of instance selection: *AdaBoost*.

Input: Training set $T = \{(\mathbf{x}_1, y_1), \ldots (\mathbf{x}_n, y_n)\}$, $\mathbf{x} \in \mathbb{R}^m$, l: number of iterations.

Result: Instance weights: $\mathbf{w} = \{w_1, w_2, \ldots, w_n\}$

$\forall_{j=1,\ldots,n}\ v_j = 1/n$ /* Initialize internal sampling weights */

$\forall_{j=1,\ldots,n}\ w_j = 1/n$ /* Initialize instance weights */

$\mathbf{T}'_1 = \mathbf{T}$ **for** $i = 1, \ldots, l$ **do**

  $\mathbf{P}_i = \texttt{ISModel}(\mathbf{T}'_i)$ /* Run the base instance selection */

  $\mathbf{id} = \forall_{j=1,\ldots,n}\ 1NN(\mathbf{P}_i, \mathbf{x}_j) \neq y_j$ /* Get identifiers of incorrectly classified instances based on $1NN$ classifier trained on $\mathbf{P}_i$ */

  $\epsilon_i = \frac{1}{n}\sum_{\mathbf{id}} v_i$ /* Calculate error */

  if $\epsilon_i > 0.5$ **then**

    $\alpha_i = 0 \quad \mathbf{v} \leftarrow \forall_{j=1,\ldots,n}\ v_j = 1/n$ /* Reset sampling weights */

  else if $\epsilon_i = 0$ **then**

    $\alpha_i = 1 \quad \mathbf{v} \leftarrow \forall_{j=1,\ldots,n}\ v_j = 1/n$ /* Reset sampling weights */

  else

    $\alpha_i = \frac{1}{2}\log_{10}\frac{1-\epsilon_i}{\epsilon_i}$ **foreach** $\mathbf{x}_j \in \mathbf{T}$ **do**

      if $1NN(\mathbf{P}_i, \mathbf{x}_j) == y_j$ **then**

        $v_j = \frac{v_j}{2(1-\epsilon_i)}$

      else

        $v_j = \frac{v_j}{2\epsilon_i}$

      end

    end

    $\mathbf{v} \leftarrow \forall_{j=1,\ldots,n}\ v_j = \frac{v_j}{\sum v_j}$ /* Normalize sampling weights */

  end

  $\mathbf{T}'_i = \texttt{Sample}(T, \mathbf{w})$ /* Bootstrap sample from $\mathbf{T}$ according to weights distribution $\mathbf{w}$ */

  $\mathbf{T}'_i = \texttt{Unique}(\mathbf{T}'_i)$ /* Remove repeated instances from $\mathbf{T}'_i$ */

  $\forall_{j\in\mathbf{P}_i}\ w_j += \alpha_i$ /* Increase counter for selected instances */

end

$\mathbf{w} \leftarrow \forall_{j=1,\ldots,n}\ v_j = \frac{w_j}{\max w_j}$ /* Normalize instance weights */

return $\mathbf{w}$

Algorithm 4. Threshold optimization algorithm.

Input: Training set $T = \{(\mathbf{x}_1, y_1), \ldots (\mathbf{x}_n, y_n)\}$, $\mathbf{x} \in \mathbb{R}^m$, $\mathbf{w}$: instance weight vector, l: maximum number of thresholds, α: compression accuracy trade-off parameter.

Result: Selected subset: $\mathbf{P} \subset \mathbf{T}$

$\mathbf{\Theta} = \texttt{Unique}(\mathbf{w})$ /* Find set of unique weight values */

$s = |\mathbf{\Theta}|$ /* Get number of possible thresholds */

if $s > l$ **then**

  $\mathbf{\Theta} = \texttt{Discretize}(\mathbf{w}, l)$ /* Discretize $\mathbf{w}$ into l bins and keep l thresholds */

else

  $l = s$ /* Set the number of thresholds in case $s < l$ */

end

for $i = 1, \ldots, l$ **do**

  $\theta = \Theta_i$ /* Select threshold */

  CrossValidation $(\mathbf{T})$: $\overline{perf}$

    $\mathbf{P} = \{\mathbf{x}_i \in \mathbf{T}_{Train} : w_i > \theta\}$ /* Select all instances such that $w_i > \theta$ */

    $\mathbf{y}'_{Test} = \texttt{Predictor}(\mathbf{P}, \mathbf{T}_{Test})$ /* Train Predictor (1NN) on $\mathbf{P}$ and apply to $\mathbf{T}_{Test}$ */

    $acc = \texttt{Accuracy}(\mathbf{y}_{Test}, \mathbf{y}'_{Test})$ /* Calculate accuracy */

    $cmp = 1 - \frac{|\mathbf{P}|}{|\mathbf{T}_{Train}|}$ /* Calculate compression */

    $perf = \alpha \times acc + (1 - \alpha) \times cmp$ /* Get final performance */

  end

  if $\overline{perf} > perf_{Best}$ **then** /* Keep best performance */

    $perf_{Best} = \overline{perf} \quad \theta_{Best} = \theta$

  end

end

$\mathbf{P} = \{\mathbf{x}_i \in \mathbf{T} : w_i > \theta_{Best}\}$ /* Select instances according to θ_{Best} */

return $\mathbf{P}$

right corner are located all of the base instance selection methods. Here noticeable is *Drop3*, which has the highest compression rank and the highest accuracy rank among the base methods. Moreover, the accuracy rank of *Drop3* is comparable to the ranks of the ensembles. Note that these results of the ensembles were obtained to maximize the accuracy and ignore the compression ($\alpha = 1$), so the distribution of the results may change when compression is also considered.

4.3. Scenario 3. The key objective of instance selection is to compress the training data to increase the training speed but also the execution speed of a

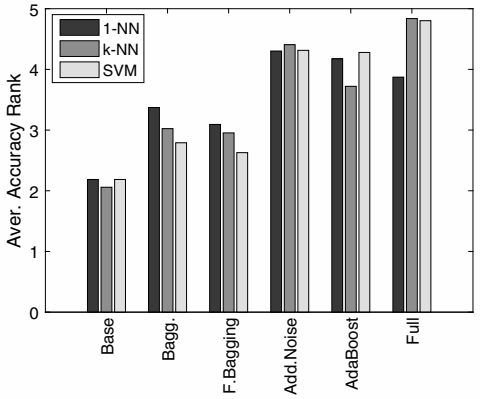


Fig. 7. Comparison of the average rank of accuracy over all evaluated instance selection methods for three classifiers: *1*NN, *k*NN and SVM trained using a dataset compressed by base instance selection and four ensembles of instance selection.

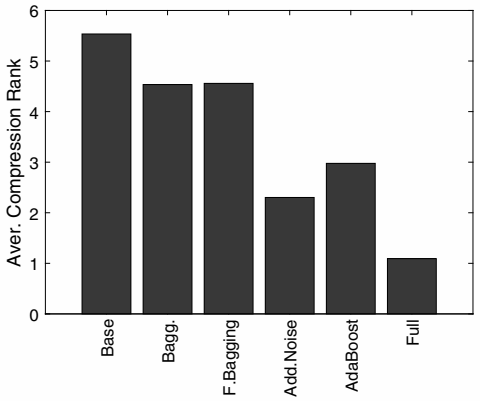


Fig. 8. Comparison of the average rank of compression over all evaluated instance selection methods for base instance selection and four ensembles of instance selection.

classifier (it reduces the execution time at least for most of distance based classifiers such as all analyzed methods: *1*NN, *k*NN or SVM). Thus the second group of the experiments was dedicated to analyze the relation between the compression and prediction accuracy. In the experiments, the relationship expressed in (6) was evaluated for $\alpha = [0.5, 0.6, 0.7, 0.8, 0.9, 1]$. The collected performances were ranked separately for each α value, as in the previous experiments. The obtained values are presented as charts (Fig. 10) independently for each of the base methods. In that figure the x-axis represents the values of α while the y-axis stands for the value of ranks of the evaluated methods recorded for a given value of α. Here, the experiments were performed only for the *1*NN classifier.

The analysis of the results shown in Fig. 10 indicates that for the unstable condensation base methods: *CNN* and *IB3* the best solution is obtained without any ensemble. However, for $\alpha = 1$ the ensembles starts to dominate. This agrees with the results observed in *Scenario 1*. That is the result of a long tail in the estimated weight distribution (for example, see Fig. 3). However, for the remaining instance selection base algorithms the ensemble methods dominate. For the condensation group: *Drop3*, *ICF*, both *Bagging* and *Additive Noise* win. For the noise removal methods like *ENN* and *All-kNN* the first place is taken by *AdaBoost* and right behind it is *Bagging*. Similar results are obtained for methods with moderate compression like *HMN-E* and *CCIS*, where again *Bagging* and *AdaBoost* dominate, but for *RNGE AdaBoost* is outperformed by *Additive Noise*.

In order to draw an overall comparison of the obtained results, the area under the curves shown in each chart of Fig. 10 was determined. This allows us to describe the results with a single value, thus facilitating the interpretation of the results. The obtained values are listed in Table 5 as well as represented as a stacked bar chart for each ensemble (see Fig. 11). The last row in the table and the hight of the bars indicate the best methods, which are *Bagging* with *AdaBoost* right behind it, then the third place is taken by *Additive Noise* and the last one belongs to *Feature Bagging*. A relatively high bar was also obtained for the base method, but as can be seen, it is determined by the performance of unstable condensation methods such as *IB3* and *CNN*, which do not work well with the ensembles.

5. Conclusions

In the paper we introduced one new type of ensemble, called *Additive Noise*, to cover all basic types of ensembles and compared all four ensembles of instance selection methods: *Bagging*, *Feature Bagging*, *AdaBoost* and the new *Additive Noise*. All these methods were compared and tested using 9 different base methods, namely, *CNN*, *IB3*, *ICF*, *Drop3*, *ICF*, *All-kNN*, *ENN*, *HMN-E* and *CCIS*. The study included a detailed analysis of three scenarios. In the first one, the ensembles were evaluated in order to analyze the influence of the threshold θ on the accuracy–compression relation. In the second scenario the ensembles were optimized to achieve maximum prediction accuracy. Finally, in the third scenario, accuracy and compression were combined into a single fitness function and optimized to imitate a use-case when both compression and accuracy are desired.

From *Scenario 1* we can see how the ensembles influence instance selection: we can observe desired and undesired behaviors of the ensembles. The best solution is when the base model lies on the solution curve in the flat section (for low compression) or the steep section—then we can respectively improve compression without significant accuracy loss or improve accuracy without a significant compression loss. However, it is possible that the solution of the base model lies outside

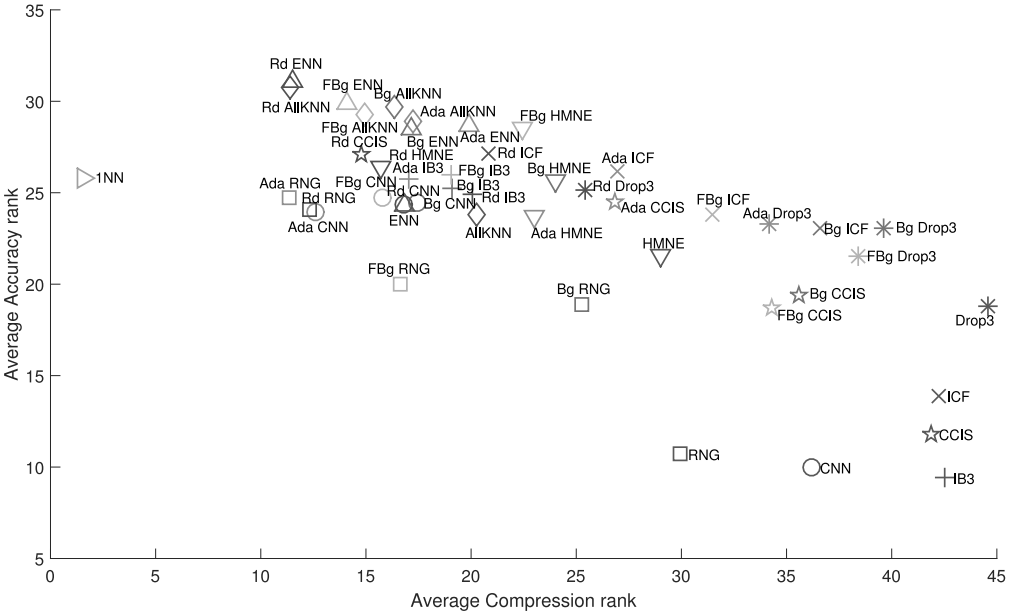
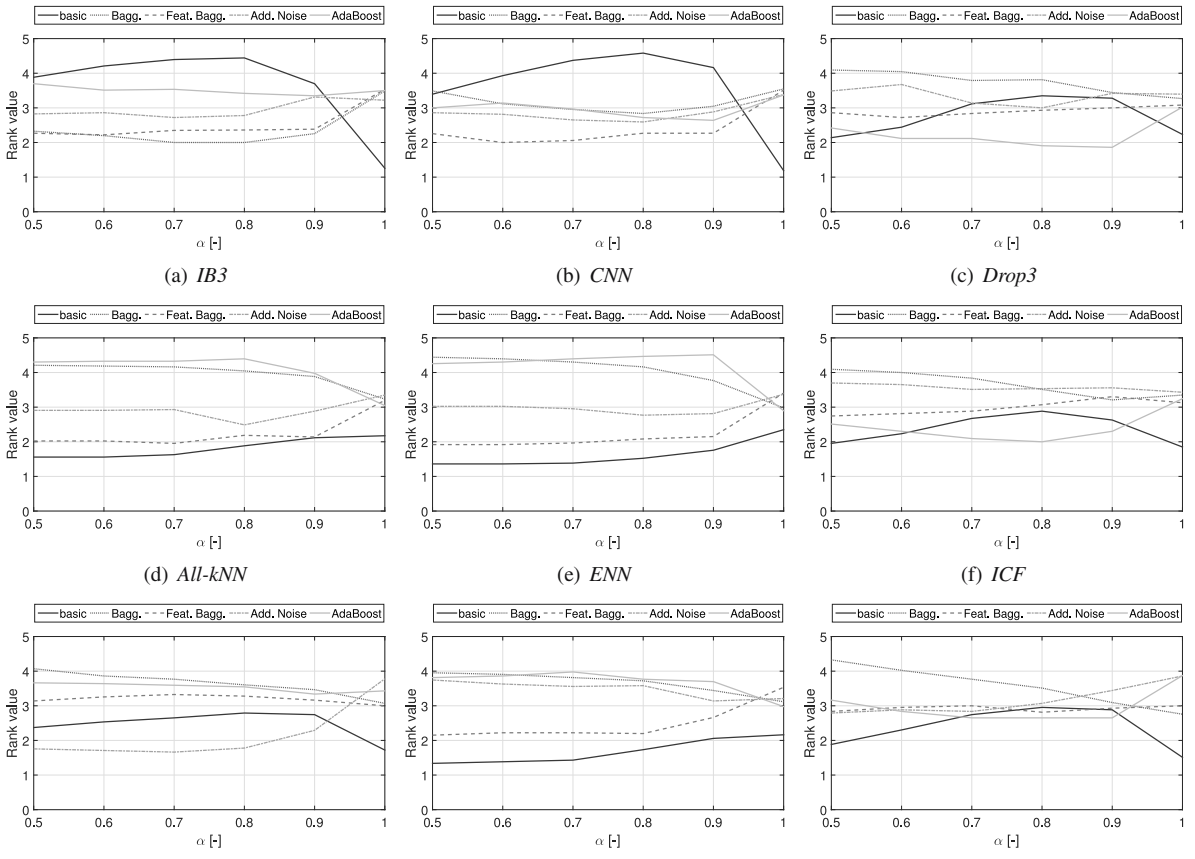


Fig. 9. Overall comparison of all evaluated ensembles (for $\alpha = 1$) and base methods. The values represent the average rank.



(a) *IB3* (b) *CNN* (c) *Drop3*

(d) *All-kNN* (e) *ENN* (f) *ICF*

(g) *CCIS* (h) *HMN-E* (i) *RNGE*

Fig. 10. Relation between the value of α and the average rank of the performance evaluated using (6).

Table 5. Area under the curves shown in Fig. 10 for each base model.

	base	*Bagging*	*Feature Bagging*	*Additive Noise*	*AdaBoost*
IB3	1.93	1.14	1.22	1.47	1.74
CNN	1.93	1.55	1.15	1.41	1.47
Drop3	1.44	1.88	1.45	1.67	1.07
ICF	1.23	1.83	1.50	1.78	1.16
RNGE	1.26	1.79	1.46	1.56	1.43
All-kNN	0.91	2.00	1.09	1.43	2.07
ENN	0.79	2.03	1.08	1.48	2.13
HMN-E	0.84	1.84	1.21	1.74	1.87
CCIS	1.28	1.83	1.61	1.02	1.77
Sum	11.60	15.88	11.77	13.55	14.70

Table 6. Recommendations for the selection of the ensemble type for a specific instance selection base method. The number of checkmarks (at most three) indicates the quality of the preference.

Method		*Bagging*	*Feature Bagging*	*Additive Noise*	*AdaBoost*
Unstable condensation	*CNN*	✓			
	IB3	✓			
Stable condensation	*ICF*	✓✓✓		✓✓	
	Drop3	✓✓✓		✓✓	
	CCIS	✓✓✓	✓		✓✓✓
Medium compression	*HMN-E*	✓✓✓		✓✓	✓✓✓
	RNGE	✓✓	✓	✓✓	✓
Noise filtering	*ENN*	✓✓		✓	✓✓✓
	All-kNN	✓✓		✓	✓✓✓

the solution curve, and then we may expect a loss in compression. It occurs for unstable base methods, when in each iteration of the ensemble different and not repeatable instances are selected. Finally, it may happen that the solution curve is linear with a medium slope angle; then the ensembles do not bring significant benefit unless we are interested only in accuracy.

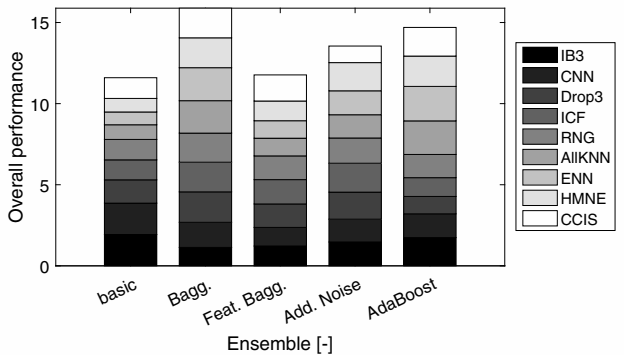


Fig. 11. Stacked bar chart of the results shown in the rows of Table 5. It represents the quality of the ensemble methods for each of the base instance selection algorithms. The values are calculated taking the area under the ranks obtained for each ensemble method for changing α values (see Fig. 11)

Scenario 2 indicates two conclusions: one that ensembles allows an improvement in the accuracy in comparison with base instance selection (in most cases it is a statistically significant improvement except *CNN* and *IB3*), but they decrease the accuracy of the final classifiers (except for *1*NN) trained on the entire training set. In the comparison the best results are obtained for *Bagging* and *AdaBoost*, but *Bagging* has a better average compression rank and is easy to parallelize, so we recommend it in practical applications. On the average, the best compression was obtained for the base instance selection. The only exception is *ENN* and *All-kNN*, but the difference was not significant.

Finally, in *Scenario 3* we compared both compression and accuracy aggregated into a single objective function. The significance of the first and second criteria was modeled by the α parameter. From this experiment we can see that there are methods called unstable (sensitive to initialization) for which ensembles have limited applicability (only when α is close to 1); then for the remaining methods *Bagging* and *AdaBoost* usually led followed by *Additive Noise*. *Feature Bagging* takes the last place among the ensembles.

The performed study does not allow us to indicate a single winner in terms of a pair *base method–ensemble*. One type of ensemble works better with the one base

method, but for others may lead to poor results. In general we do not recommend to use ensembles for unstable methods which are sensitive to random initialization, for all others our recommendations are included in Table 6. The number of check marks (✓) indicates how strongly a given type of ensemble is recommended for use with a particular instance selection method.

References

Abdi, H. (2010). Holm's sequential Bonferroni procedure, *Encyclopedia of Research Design* **1**(8): 620–627.

Aha, D., Kibler, D. and Albert, M. (1991). Instance-based learning algorithms, *Machine Learning* **6**(1): 37–66.

Alcalá-Fdez, J., Fernández, A., Luengo, J., Derrac, J., García, S., Sanchez, L. and Herrera, F. (2011). Keel data-mining software tool: Data set repository, integration of algorithms and experimental analysis framework, *Journal of Multiple-Valued Logic & Soft Computing* **17**: 255–287.

Arnaiz-González, Á., Blachnik, M., Kordos, M. and García-Osorio, C. (2016a). Fusion of instance selection methods in regression tasks, *Information Fusion* **30**: 69–79.

Arnaiz-González, Á., Díez-Pastor, J., Rodríguez, J.J. and García-Osorio, C.I. (2016b). Instance selection for regression: Adapting DROP, *Neurocomputing* **201**: 66–81.

Bauer, E. and Kohavi, R. (1999). An empirical comparison of voting classification algorithms: Bagging, boosting, and variants, *Machine Learning* **36**(1): 105–139.

Bezdek, J.C., Ehrlich, R. and Full, W. (1984). FCM: The fuzzy C-means clustering algorithm, *Computers & Geosciences* **10**(2–3): 191–203.

Bhattacharya, B., Poulsen, R. and Toussaint, G. (1984). Application of proximity graphs to editing nearest neighbor decision rules, *International Symposium on Information Theory, Santa Monica, CA, USA*, pp. 97–108.

Blachnik, M. (2014). Ensembles of instance selection methods based on feature subset, *IEEE Procedia Computer Science* **35**: 388–396.

Blachnik, M. and Kordos, M. (2014). Bagging of instance selection algorithms, *International Conference on Artificial Intelligence and Soft Computing, Zakopane, Poland*, pp. 40–51.

Brighton, H. and Mellish, C. (2002). Advances in instance selection for instance-based learning algorithms, *Data Mining and Knowledge Discovery* **6**(2): 153–172.

Czarnowski, I. and Jędrzejowicz, P. (2015). Ensemble online classifier based on the one-class base classifiers for mining data streams, *Cybernetics and Systems* **46**(1–2): 51–68.

Freund, Y. and Schapire, R.E. (1996). Experiments with a new boosting algorithm, *International Conference on Machine Learning, Bari, Italy*, pp. 148–156.

Freund, Y. and Schapire, R.E. (1997). A decision-theoretic generalization of on-line learning and an application to boosting, *Journal of Computer and System Sciences* **55**(1): 119–139.

Galar, M., Fernández, A., Barrenechea, E., Bustince, H. and Herrera, F. (2011). An overview of ensemble methods for binary classifiers in multi-class problems: Experimental study on one-vs-one and one-vs-all schemes, *Pattern Recognition* **44**(8): 1761–1776.

García-Osorio, C., de Haro-García, A. and García-Pedraja, N. (2010). Democratic instance selection: A linear complexity instance selection algorithm based on classifier ensemble concepts, *Artificial Intelligence* **174**(4–5): 410–441.

García, S., Derrac, J., Cano, J.R. and Herrera, F. (2012). Prototype selection for nearest neighbor classification: Taxonomy and empirical study, *IEEE Transactions on Pattern Analysis and Machine Intelligence* **34**(3): 417–435.

García-Pedrajas, N. (2009). Constructing ensembles of classifiers by means of weighted instance selection, *IEEE Transactions on Neural Networks* **20**(2): 258–277.

García-Pedrajas, N. and De Haro-García, A. (2014). Boosting instance selection algorithms, *Knowledge-Based Systems* **67**: 342–360.

García, S., Luengo, J. and Herrera, F. (2016). Tutorial on practical tips of the most influential data preprocessing algorithms in data mining, *Knowledge-Based Systems* **98**: 1–29.

Grochowski, M. and Jankowski, N. (2004). Comparison of instance selection algorithms. II: Results and comments, Lecture Notes in Computer Science, Vol. 3070, pp. 580–585.

Gunn, I.A., Arnaiz-González, Á. and Kuncheva, L.I. (2018). A taxonomic look at instance-based stream classifiers, *Neurocomputing* **286**: 167–178.

Hart, P. (1968). The condensed nearest neighbor rule, *IEEE Transactions on Information Theory* **14**(3): 515–516.

Herrera, F. (2005). Keel, knowledge extraction based on evolutionary learning, Spanish National Projects TIC2002-04036-C05, TIN2005-08386-C05 and TIN2008-06681-C06, `http://www.keel.es`.

Jacobs, R.A., Jordan, M.I., Nowlan, S.J. and Hinton, G.E. (1991). Adaptive mixtures of local experts, *Neural Computation* **3**(1): 79–87.

Jankowski, N. and Grochowski, M. (2004). Comparison of instance selection algorithms. I: Algorithms survey, *International Conference on Artificial Intelligence and Soft Computing, Zakopane, Poland*, Vol. 3070, pp. 598–603.

Kordos, M. and Blachnik, M. (2012). Instance selection with neural networks for regression problems, *International Conference on Artificial Neural Networks, Lausanne, Switzerland*, pp. 263–270.

Kordos, M. and Rusiecki, A. (2016). Reducing noise impact on MLP training, *Soft Computing* **20**(1): 49–65.

Kuncheva, L. (2004). *Combining Pattern Classifiers: Methods and Algorithms*, Wiley, Hoboken, NJ.

Kuncheva, L.I., Bezdek, J.C. and Duin, R.P. (2001). Decision templates for multiple classifier fusion: An experimental comparison, *Pattern Recognition* **34**(2): 299–314.

Marchiori, E. (2008). Hit miss networks with applications to instance selection, *Journal of Machine Learning Research* **9**(Jun): 997–1017.

Marchiori, E. (2010). Class conditional nearest neighbor for large margin instance selection, *IEEE Transactions on Pattern Analysis and Machine Intelligence* **32**(2): 364–370.

Raviv, Y. and Intrator, N. (1996). Bootstrapping with noise: An effective regularization technique, *Connection Science* **8**(3–4): 355–372.

Rokach, L. (2009). Taxonomy for characterizing ensemble methods in classification tasks: A review and annotated bibliography, *Computational Statistics & Data Analysis* **53**(12): 4046–4072.

Schapire, R.E. (1990). The strength of weak learnability, *Machine Learning* **5**(2): 197–227.

Sebban, M., Nock, R. and Lallich, S. (2002). Stopping criterion for boosting-based data reduction techniques: From binary to multiclass problem, *Journal of Machine Learning Research* **3**(Dec): 863–885.

Shaker, A. and Hüllermeier, E. (2012). IBLStreams: A system for instance-based classification and regression on data streams, *Evolving Systems* **3**(4): 235–249.

Skurichina, M. and Duin, R.P. (2001). Bagging and the random subspace method for redundant feature spaces, *International Workshop on Multiple Classifier Systems, Cagliari, Italy*, pp. 1–10.

Song, Y., Liang, J., Lu, J. and Zhao, X. (2017). An efficient instance selection algorithm for k nearest neighbor regression, *Neurocomputing* **251**: 26–34.

Tomek, I. (1976). An experiment with the edited nearest-neighbor rule, *IEEE Transactions on Systems, Man, and Cybernetics* **6**: 448–452.

Wilson, D. (1972). Asymptotic properties of nearest neighbor rules using edited data, *IEEE Transactions Systems, Man and Cybernetics* **2**: 408–421.

Wilson, D. and Martinez, T. (2000). Reduction techniques for instance-based learning algorithms, *Machine Learning* **38**(3): 257–268.

Wolpert, D.H. (1992). Stacked generalization, *Neural Networks* **5**(2): 241–259.

Woźniak, M., Graña, M. and Corchado, E. (2014). A survey of multiple classifier systems as hybrid systems, *Information Fusion* **16**: 3–17.

Zhu, J., Zou, H., Rosset, S. and Hastie, T. (2009). Multi-class AdaBoost, *Statistics and Its Interface* **2**(3): 349–360.

Marcin Blachnik received his MSc in electronics and telecommunications in 2002 and his PhD in computer science in 2007, both from the Silesian University of Technology, Poland. He is currently an assistant professor in the Department of Industrial Informatics there. His interests include data mining, data reduction, similarity based methods, meta-learning and its applications to industrial processes.

Received: 20 March 2018
Revised: 18 September 2018
Re-revised: 24 October 2018
Accepted: 7 November 2018