# Data Set Partitioning in Evolutionary Instance Selection

Mirosław Kordos[1(✉)], Łukasz Czepielik[1], and Marcin Blachnik[2]

[1] Department of Computer Science and Automatics, University of Bielsko-Biala, Willowa 2, Bielsko-Biała, Poland
mkordos@ath.bielsko.pl

[2] Department of Applied Informatics, Silesian University of Technology, Krasińskiego 8, Katowice, Poland
marcin.blachnik@polsl.pl

**Abstract.** Evolutionary instance selection outperforms in most cases non-evolutionary methods, also for function approximation tasks considered in this work. However, as the number of instances encoded into the chromosome grows, finding the optimal subset becomes more difficult, especially that running the optimization too long leads to over-fitting. A solution to that problem, which we evaluate in this work is to reduce the search space by clustering the dataset, run the instance selection algorithm for each cluster and combine the results. We also address the issue of properly processing the instances close to the cluster boundaries, as this is where the drop of accuracy can appear. The method is experimentally verified on several regression datasets with thousands of instances.

## 1 Introduction

Data preprocessing is frequently the most important step in data mining, even more important than choice of the learning model and its parameters, as even the best model cannot produce good outcome if the data quality is poor. An important step of data pre-processing is data selection, this is feature selection and instance selection. So far much more research has been conducted on feature selection than on instance selection and a few papers proposed joint approach [1]. There are two objectives of data selection: to improve the prediction accuracy and to reduce the training dataset size to make the model learning faster and data analysis easier. In classification tasks the accuracy is usually expressed by the percentage of correctly classified instances and in regression tasks by *rmse* (root mean square error).

Most of non-evolutionary methods of instance selection belong to two groups. The first group is used prior to the model learning and it decides upon the removal of particular instances by examining their local neighborhood. A number of such methods mostly for classification tasks and also a few for regression were proposed in the literature. A comprehensive review can be found in [2] and [3].


H. Yin et al. (Eds.): IDEAL 2018, LNCS 11314, pp. 631-641, 2018.
https://doi.org/10.1007/978-3-030-03493-1_66

The second group embeds instance selection into the model learning by some modifications or additions to the learning algorithm [4].

The purpose of instance selection methods can be divided into three categories: noise filters, data condensation methods and methods joining both tasks. The noise filters are designed to remove only those instances, which constitute noise in the data. Removing them only slightly reduces the dataset size (usually below 10%), but it improves prediction accuracy.

Data condensation methods are frequently used as the second step, after noise has been removed. Their purpose is to find and remove the irrelevant instances, this is instances that can be removed from the training set $T$ without impacting the prediction accuracy of a model trained on this set. In case of the condensation methods, usually stronger data reduction causes also decrease in prediction accuracy. However, that can still be useful if the reduction is very strong and the accuracy loss is minimal, because this will allow to have a better insight into the data properties and to make some preliminary experiments on the reduced dataset, so that they run quickly, what is especially important for the learning models with high computational cost.

In evolutionary methods based on genetic algorithms the dataset is usually represented by a chromosome, where each position (allele) corresponds to one instance; 0 at this position means the instance will be rejected and 1 means it will be selected. Then the optimization runs and the fitness function, which is a weighted sum of the model accuracy and the training dataset reduction is evaluated. That is the general principle and particular solutions may implement it in various ways.

One advantage of evolutionary-based methods is that there is no need to define the exact rules for the instance removal. The other advantage is that most cases better results in terms of accuracy-reduction balance can be obtained [5,6].

In evolutionary instance selection, which we discussed in Sect. 2, the final objectives are: minimization of training dataset size (number of instances) and minimization prediction error (*rmse* in case of regression) on the test set. However, the test set is unknown during the optimization and thus the second objective cannot be minimized directly, instead the error on the training set is minimized. Thus there is the same problem as with learning the predictive models: running the optimization on the training set for too many iterations leads to over-fitting and thus increases of error on the test set.

There is no simple general solution to the problem of uncertainty in genetic optimization as shown in [7] as the fitness function evaluation is different as well between problems as in different phases of the optimization in a single problem. The problem gets more serious for larger datasets and in this paper we address this issue by partitioning the training dataset. Then we run the optimization on the parts, and merge the results properly, as discussed in Sect. 3. The experimental evaluation presented in Sect. 4 proves that this approach allows for obtaining better results in terms of *rmse* - data reduction balance.

## 2 Evolutionary Instance Selection

There have been already some propositions in the literature to use genetic algorithms for instance selection, which confirmed that these approaches can find better solutions that the classical methods based on the local distances or neighborhood [5,6,8–11].

The way in which genetic algorithms work is with very high probability well known to the reader so we will not explain this, especially that all the details can be easily found in literature. In our application a single objective genetic algorithm is used [12,13].

Each individual in the genetic population encodes the entire training set and the value at each chromosome position indicates whether the instance is present (value=1) or not (value=0). In the sample chromosome below, the dataset consists of 20 instances; the first instance is selected, the second rejected, the next three instances are selected, next two rejected and so on.

|1|0|1|1|1|0|0|1|1|1|0|1|1|0|0|1|0|1|1|1|1|

A prediction model is used during the optimization to calculate the error on the training set. (We call it an inner prediction model to distinguish it from the final prediction model, which is trained on the reduced dataset and predicts the outputs for the test set.) Thus, if there are e.g. 96 individuals and the optimization runs for 25 iterations, the error on the training set has to be evaluated 2400 times. In order to significantly reduce computational cost of calculating the error, we use k-NN with pre-calculated and sorted distance matrices as the inner prediction model. We calculate and sort the distances between each pair of instances in the training set only once before the optimization starts. Then during the optimization we read the output values of the $k$ nearest selected neighbors from the sorted arrays. If a given nearest neighbor is rejected in the current individual, then we go to the next one until we read from the sorted array the outputs of $k$ selected instances. Then we predict the output value as the average of the $k$ selected nearest neighbor outputs and calculate the $rmse$ over the training dataset. This is really very fast, making the computational cost of this method comparable to the cost of the non-evolutionary instance selection methods.

Most of evolutionary approaches to instance selection define the fitness function in a similar way, as a weighted sum of the achieved reduction and inverse of the error on the training set. Thus to obtain a set of solutions the optimization has to be performed several times with various weights $\alpha$. These two objectives are composed into the following fitness function:

$$fitness = \left(\alpha * \frac{avgRMSE}{RMSE} + (1-\alpha) * \frac{avgNumInstances}{numInstances}\right)^{v} \tag{1}$$

where exponent $v$ usually gradually increases as the optimization progresses (e.g. from $v = 2$ to $v = 4$) and each individual is selected as a parent in the crossover process with a probability proportional to its fitness.

Another solution is to use multi-objective evolutionary algorithms [14], where such a fitness function does not need to be defined, but only the criteria: *rmse* and instance reduction [16,17].

We use multi-point multi-parent crossover with $M$ parents and $M-1$ crossover points. $M$ parents are randomly selected proportionally to their fitness for each child; each of them gives its fragment of genetic material into one segment of the child, between two successive crossover points. After producing offspring with the crossover operator (with probability of 100%), we sort all parents and children together and $P$ best individuals are advanced to the next iteration. We use the mutation probability of 0.001. Such parameters were selected based on our previous work [5]. In the experiments with we used two $\alpha$ values: 0.9 and 0.8. We used population size $P = 96$, because it is within the range of optimal population size and our servers had 48 CPU cores so parallel calculations scaled efficiently ($2 * 48 = 96$).

## 3 The Proposed Improvement

As it was mentioned, one of the final objectives is the *rmse* on the test set, but the fact the objective that we optimize directly is *rmse* on the training set, because during the optimization the test set is unknown and this requires early stopping, before the over-fitting occurs. For big datasets, there is a problem with determining the optimal stopping point. We observed that in some areas of the dataset the optimization converges faster than in others. Goldberg wrote that genetic algorithms work "by building short, low order, and highly fit schemata (blocks), which are recombined (crossed over), and re-sampled to form strings of potentially higher fitness" [12]. In this way, the over-fitting already may start occurring within such a block, while in other parts of the chromosome still more optimization is needed.

Even in tasks, where the objectives are directly optimized and thus there is no risk of over-fitting, partitioning the search space can be useful, because the already created blocks can be disturbed in the process of creating new ones and some researchers have already tried this approach. In [19] a differential evolutionary algorithm with space partitioning was implemented by dividing the search variables into groups of partitions, so each partition contained a certain number of variables and was manipulated as a subspace in the search process. In [18] the search space partitioning was applied to multi-objective genetic algorithms.

We compared the results of the evolutionary based instance selection with the instance selection performed by the regression version of the DROP3-RT algorithm [20] (which according to the comparative study in [3] was one of the best classical instance selection algorithms). For smaller datasets the differences were huge: the evolutionary method was much better, but for the dataset sizes of about 10.000 instances the differences begun to get much smaller and in some cases even the *rmse* of DROP3-RT was lower. Introducing the data partitioning and performing the instance selection separately inside each partition again allowed for gaining almost as significant advantage over the DROP-3 results as for the smaller datasets.

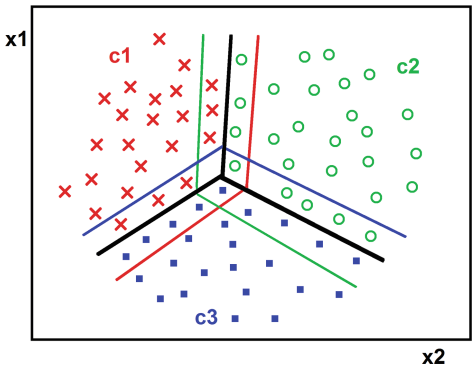


**Fig. 1.** The exact and the extended clusters (Color figure online)

We use the k-means clustering to partition the $n$ instances into $k$ ($k < n$) sets (clusters) $S_1, S_2, \ldots, S_k$ in order to minimize the within-cluster sum of squares (or variance):

$$\arg\min_{\mathbf{S}} \sum_{c=1}^{k} \sum_{\mathbf{x} \in S_c} \|\mathbf{x} - \boldsymbol{\mu}_c\|^2 = \arg\min_{\mathbf{S}} \sum_{c=1}^{k} |S_c| \mathrm{Var} S_c \tag{2}$$

where $c$ is the number of the cluster. As k-means is a well known algorithm and its description can be easily found in the literature [15] so we do not describe it here. However, it is worth mentioning, that we use also the output variable of the instances as one of the input variables to the clustering, as this proved to produce more adequate results for the purpose of instance selection.

Of course not each optimization problem can be partitioned in this way, because frequently there are mutual interactions between many positions in the chromosome. However, in the case of predicting the output of regression datasets it can be done, because there are no interactions between very remote instances.

Figure 1 shows a dataset with two attributes $x1$ and $x2$ partitioned into three clusters $c1$, $c2$ and $c3$ with the thick black lines. The thin red lines represent the boundaries of the extended red cluster $c1$; this is the red cluster itself and all instances from the other clusters that are the nearest neighbors for any instance from the red cluster.

**Algorithm 1.** The instance selection process

```
for m=1 to 10 (crossvalidation) do
    Partition the m − th training T_m set into C clusters
    Optionally determine the extended clusters E for each of the C clusters
    for c=0 ... C do
        Generate initial currentPopulation of P individuals
        calculate fitness f (Eq. 1) for currentPopulation individuals
        for i=0 ... numIterations do
            apply the crossover operation to generate the newPopulation of P individuals
            calculate fitness f (Eq. 1) for newPopulation individuals
            sort together currentPopulation and newPopulation individuals by fitness
            select the best P individuals into currentPopulation
            apply the mutation operation
        end for
    end for
    Merge the selected instances from all C clusters to form the final reduced training set T_mr
    calculate the reduction_m of the training set T_mr
    calculate the rmse_m on the corresponding test set S_m using the predictor trained on T_mr
end for
average the reduction and rmse over the m sets to get the final result
```

This approach usually allows for improving both criteria: obtaining higher reduction with slightly lower *rmse*, but definitely the reduction criterion gets more improved than the *rmse* criterion. There is some "boundary effect"; this is we can observe some problems with the instances that are so close to the cluster borders that some of their nearest neighbors belong to another cluster. When we want to predict the output of such an instance using only the neighbors from its own cluster, the results are inadequate. For this reason we cannot further decrease *rmse* using only that approach. Moreover, the decreasing prediction error inside the clusters is partially canceled out by the increasing error on the borders of the clusters. As a result the optimal size of the cluster in our experimental evaluation ranged from 500 to 1.500 instances.

In order to further improve the results, we propose to use a pair of datasets for each cluster: the dataset that forms the current cluster $T_c$ and the extended dataset $T_e$ which consists of the current cluster $T_c$ and all $k$ nearest neighbors of all instances from $T_c$, no matter to which cluster they belong, as shown in Fig. 1. We perform instance selection only within $T_c$, so only the instances from $T_c$ are considered for removal. However, while predicting with k-NN the outputs of instances from $T_c$ (which we need to obtain *rmse*), all the instances from $T_e$ are considered as their potential neighbors. The method is shown in the pseudo-code 1.

## 4 Experiments and Results

The two criteria to asses the quality of the solution are: data reduction and *rmse* on the test set. Because of limited space, we present here only results obtained with single-objective genetic algorithm and with 1-NN and optimal $k$

k-NN as the inner-evaluation model and as the final prediction model 1 (optimal $k$ means $k$ for which the lowest $rmse$ was obtained on the whole training set in 10-fold crossvalidation). The source code of the software we created to perform the experiments (which also implements multi-objective instance selection algorithms and other prediction models) and the datasets can be found in supplementary materials to this paper at www.kordos.com/ideal2018.

As the dataset partitioning is useful for bigger datasets, we conducted the experiments on the 16 biggest benchmark regression datasets from the well known KEEL Repository [21]. The datasets were standardized to enable better comparison of the results in the tables. These datasets describe the realwold problems and are commonly used for evaluating different algorithms and thus provide much more reliable results than generating some artificial datasets (someone could argue that the artificial datasets were especially generated in such a way to show the desired results).

First we randomly changed the order of the instances in each dataset and then we used linear sampling in 10-fold crossvalidation to ensure that both of the methods (with and without data partitioning) use exactly the same training and test subsets - so that the comparison is made on exactly the same data and the results are not influenced by different sets. We used two-tailed Wilcoxon Signed-Rank Test and in each case the difference was statistically significant at $p < 0.05$. The results presented in the second and third table are the average $rmse$ values obtained on the test sets in the crossvalidation and the corresponding average percentage of selected instances of the training set (retention rate).

The final prediction model and the inner evaluation model was k-NN with optimal $k$ for each dataset. However, in the experiments, we limited the maximum $k$ to 11. The rationale behind this was that only the most noisy datasets had optimal $k$ value above 11 and after performing instance selection, which reduced the noise, the optimal $k$ dropped to lower values, not exceeding 11. The number of clusters **Nc** used for each dataset is shown in Table 1. As our experiments showed the optimal number of instances in one cluster was between 500 and 1500 (Table 3).

If we used more instances, then the effect of the genetic algorithm not finding the optimal solution before over-fitting started to be visible. If we decreased the number of instances below the lower limit, the border effect started to decreasing the result quality. The precise optimization of the cluster size is not required and even not possible, as the differences between the algorithm performance with the cluster size of e.g. 400 and 700 are below the variability caused by the stochastic character of genetic algorithms.

**Table 1.** Datasets used in the experiments (obtained from the Keel repository) and their properties: **inst** - number of instances, **attr** - number of attributes, **oK** - the optimal $k$ (the $k$ in k-NN that gives the lowest $rmse$), **uK** - the $k$ used the experiments in place of optimal $k$, **Nc** - the number of clusters.

| Dataset | inst | attr | oK | uK | Nc | Dataset | inst | attr | oK | uK | Nc |
|---|---|---|---|---|---|---|---|---|---|---|---|
| wankara | 1609 | 9 | 9 | 9 | 5 | delta-elv | 9516 | 6 | 35 | 11 | 20 |
| plastic | 1650 | 2 | 30 | 11 | 5 | tic | 9822 | 85 | 90 | 11 | 20 |
| quake | 2178 | 3 | 50 | 1 | 5 | ailerons | 13750 | 40 | 10 | 10 | 20 |
| anacalt | 4052 | 7 | 2 | 2 | 10 | pole | 14998 | 26 | 4 | 4 | 20 |
| abalone | 4177 | 8 | 13 | 11 | 10 | elevators | 16598 | 18 | 8 | 8 | 20 |
| delta-ail | 7128 | 5 | 17 | 11 | 15 | california | 20640 | 8 | 9 | 9 | 20 |
| puma32h | 8191 | 32 | 21 | 11 | 15 | house | 22784 | 16 | 11 | 11 | 20 |
| compactiv | 8192 | 21 | 2 | 2 | 15 | mv | 40767 | 10 | 9 | 9 | 40 |

**Table 2.** Experimental results: inner evaluation algorithm: 1-NN, final prediction algorithm: 1-NN, r0 - $rmse$ without instance selection, r1, r2 - $rmse$ with instance selection without data partitioning for $\alpha = 0.9$ and 0.8, r1p, r2p - $rmse$ with data partitioning for $\alpha = 0.9$ and 0.8 (smaller is better), c1, c2, c1p, c2p - corresponding retention rates (smaller is better)

| Dataset | r0 | r1 | r1p | c1 | c1p | r2 | r2p | c2 | c2p |
|---|---|---|---|---|---|---|---|---|---|
| wankara | 0.225 | 0.222 | 0.209 | 0.679 | 0.623 | 0.290 | 0.266 | 0.147 | 0.136 |
| plastic | 0.617 | 0.542 | 0.513 | 0.718 | 0.674 | 0.607 | 0.589 | 0.202 | 0.192 |
| quake | 1.344 | 1.153 | 1.134 | 0.672 | 0.607 | 1.342 | 1.296 | 0.148 | 0.133 |
| anacalt | 0.227 | 0.232 | 0.211 | 0.482 | 0.470 | 0.282 | 0.280 | 0.194 | 0.171 |
| abalone | 0.915 | 0.768 | 0.747 | 0.697 | 0.655 | 0.872 | 0.844 | 0.163 | 0.154 |
| delta-ail | 0.716 | 0.630 | 0.629 | 0.618 | 0.593 | 0.744 | 0.740 | 0.141 | 0.135 |
| puma32h | 1.212 | 1.025 | 1.004 | 0.673 | 0.644 | 1.202 | 1.207 | 0.165 | 0.154 |
| compactiv | 0.254 | 0.295 | 0.295 | 0.434 | 0.414 | 0.307 | 0.307 | 0.236 | 0.216 |
| delta-elv | 0.828 | 0.708 | 0.710 | 0.663 | 0.623 | 0.838 | 0.819 | 0.136 | 0.128 |
| tic | 1.366 | 1.130 | 1.118 | 0.704 | 0.681 | 1.330 | 1.312 | 0.148 | 0.142 |
| ailerons | 0.657 | 0.585 | 0.580 | 0.596 | 0.583 | 0.701 | 0.688 | 0.142 | 0.139 |
| pole | 0.244 | 0.258 | 0.258 | 0.662 | 0.648 | 0.349 | 0.335 | 0.137 | 0.131 |
| elevators | 0.686 | 0.641 | 0.635 | 0.669 | 0.651 | 0.763 | 0.730 | 0.173 | 0.151 |
| california | 0.654 | 0.596 | 0.582 | 0.676 | 0.655 | 0.711 | 0.703 | 0.143 | 0.135 |
| house | 0.872 | 0.775 | 0.766 | 0.645 | 0.625 | 0.929 | 0.924 | 0.141 | 0.137 |
| mv | 0.210 | 0.199 | 0.199 | 0.728 | 0.721 | 0.302 | 0.300 | 0.164 | 0.160 |
| average | 0.689 | 0.610 | 0.599 | 0.645 | 0.617 | 0.723 | 0.709 | 0.161 | 0.151 |
| Wilcox. p-value | | 0.00236 | | 0.00044 | | 0.00180 | | 0.00044 | |

**Table 3.** Experimental results: inner evaluation algorithm: optimal-k k-NN, final prediction algorithm: optimal-k k-NN, symbols are explained at Table 2

| Dataset | r0 | r1 | r1p | c1 | c1p | r2 | r2p | c2 | c2p |
|---|---|---|---|---|---|---|---|---|---|
| wankara | 0.167 | 0.183 | 0.171 | 0.447 | 0.420 | 0.217 | 0.217 | 0.154 | 0.146 |
| plastic | 0.468 | 0.452 | 0.452 | 0.427 | 0.398 | 0.466 | 0.466 | 0.219 | 0.198 |
| quake | 1.025 | 1.009 | 1.010 | 0.676 | 0.611 | 1.030 | 1.029 | 0.211 | 0.200 |
| anacalt | 0.212 | 0.212 | 0.211 | 0.480 | 0.459 | 0.274 | 0.273 | 0.167 | 0.163 |
| abalone | 0.702 | 0.708 | 0.705 | 0.681 | 0.640 | 0.756 | 0.755 | 0.141 | 0.132 |
| delta-ail | 0.560 | 0.575 | 0.572 | 0.692 | 0.585 | 0.605 | 0.607 | 0.167 | 0.148 |
| puma32h | 0.896 | 0.909 | 0.911 | 0.477 | 0.458 | 0.910 | 0.905 | 0.215 | 0.206 |
| compactiv | 0.231 | 0.277 | 0.267 | 0.507 | 0.463 | 0.288 | 0.285 | 0.258 | 0.242 |
| delta-elv | 0.610 | 0.622 | 0.625 | 0.700 | 0.613 | 0.622 | 0.624 | 0.230 | 0.202 |
| tic | 1.015 | 0.996 | 0.989 | 0.680 | 0.656 | 1.014 | 1.009 | 0.177 | 0.160 |
| ailerons | 0.504 | 0.519 | 0.516 | 0.477 | 0.469 | 0.555 | 0.551 | 0.255 | 0.231 |
| pole | 0.214 | 0.236 | 0.233 | 0.487 | 0.478 | 0.254 | 0.252 | 0.303 | 0.302 |
| elevators | 0.559 | 0.577 | 0.572 | 0.465 | 0.451 | 0.625 | 0.620 | 0.207 | 0.205 |
| california | 0.527 | 0.549 | 0.551 | 0.527 | 0.516 | 0.579 | 0.580 | 0.336 | 0.324 |
| house | 0.687 | 0.708 | 0.708 | 0.563 | 0.538 | 0.727 | 0.721 | 0.269 | 0.261 |
| mv | 0.140 | 0.160 | 0.160 | 0.488 | 0.484 | 0.187 | 0.186 | 0.251 | 0.242 |
| average | 0.532 | 0.543 | 0.541 | 0.548 | 0.515 | 0.569 | 0.567 | 0.222 | 0.210 |
| Wilcox. p-value | | 0.04236 | | 0.00044 | | 0.02382 | | 0.00044 | |

## 5 Conclusions

We discussed the improvement of instance selection for regression tasks using genetic algorithms with clustering to partition the data space. The experiments showed that in the tested configuration this allows for stronger reduction of the data size and also for some reduction of *rmse*; we were able to reduce the dataset size by on average 5.5% and at the same time to reduce the *rmse* by about 1.5%. The next step in our research will be verification of this method for other learning models than k-NN and for multi-objective evolutionary instance selection. We believe that this approach can be useful also to other problems, where the search space is big and no significant interactions occur between remote parameters, especially in the cases where the optimization algorithm cannot run too long, because of possible over-fitting.

**Acknowledgements.** This work was supported by Polish National Science Center (NCN) grant "Evolutionary Methods in Data Selection" No. 2017/01/X/ST6/00202.

## References


1. Tallón-Ballesteros, A.J., Riquelme, J.C.: Data cleansing meets feature selection: a supervised machine learning approach. In: Ferrández Vicente, J.M., Álvarez-Sánchez, J.R., de la Paz López, F., Toledo-Moreo, F.J., Adeli, H. (eds.) IWINAC 2015. LNCS, vol. 9108, pp. 369–378. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-18833-1_39
2. Olvera-López, A., Carrasco-Ochoa, J., Martínez-Trinidad, F., Kittler, J.: A review of instance selection methods. Artif. Intell. Rev. **34**(2), 133–143 (2010)
3. Garcia, S., Derrac, J., Cano, J.R., Herrera, F.: Prototype selection for nearest neighbor classification: taxonomy and empirical study. IEEE Trans. Pattern Anal. Mach. Intell. **34**(3), 417–435 (2012)
4. Rusiecki, A., Kordos, M., Kamiński, T., Greń, K.: Training neural networks on noisy data. In: Rutkowski, L., Korytkowski, M., Scherer, R., Tadeusiewicz, R., Zadeh, L.A., Zurada, J.M. (eds.) ICAISC 2014. LNCS (LNAI), vol. 8467, pp. 131–142. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-07173-2_13
5. Kordos, M.: Optimization of evolutionary instance selection. In: Rutkowski, L., Korytkowski, M., Scherer, R., Tadeusiewicz, R., Zadeh, L.A., Zurada, J.M. (eds.) ICAISC 2017. LNCS (LNAI), vol. 10245, pp. 359–369. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-59063-9_32
6. Kordos, M., Wydrzyński, M., Łapa, K.: Obtaining pareto front in instance selection with ensembles and populations. In: Rutkowski, L., Scherer, R., Korytkowski, M., Pedrycz, W., Tadeusiewicz, R., Zurada, J.M. (eds.) ICAISC 2018. LNCS (LNAI), vol. 10841, pp. 438–448. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-91253-0_41
7. Merelo, J.J., et. al.: There is noisy lunch: a study of noise in evolutionary optimization problems. In: 7th International Joint Conference on Computational Intelligence (IJCCI), pp. 261–268 (2015)
8. Antonelli, M., Ducange, P., Marcelloni, F.: Genetic training instance selection in multiobjective evolutionary fuzzy systems: a coevolutionary approach. IEEE Trans. Fuzzy Syst. **20**(2), 276–290 (2012)
9. Tsai, C.-F., Eberle, W., Chu, C.-Y.: Genetic algorithms in feature and instance selection. Knowl.-Based Syst. **39**, 240–247 (2013)
10. Cano, J.R., Herrera, F., Lozano, M.: Instance selection using evolutionary algorithms: an experimental study. In: Pal, N.R., Jain, L. (eds.) Advanced Techniques in Knowledge Discovery and Data Mining. Advanced Information and Knowledge Processing, pp. 127–152. Springer, London (2005). https://doi.org/10.1007/1-84628-183-0_5
11. Derrac, J., et al.: Enhancing evolutionary instance selection algorithms by means of fuzzy rough set based feature selection. Inf. Sci. **186**, 73–92 (2012)
12. Goldberg, D.: Genetic Algorithms in Search, Optimization and Machine Learning. Addison Wesley, Boston (1989)
13. Lobo, F.G., Lima, C.F., Michalewicz, Z.: Parameter Setting in Evolutionary Algorithms. Studies in Computational Intelligence, vol. 54. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-69432-8
14. Deb, K.: Multi-Objective Optimization using Evolutionary Algorithms. Wiley, Hoboken (2001)
15. Aggarwal, C.C., Reddy, C.K.: Data Clustering: Algorithms and Applications. Chapman and Hall/CRC, Boca Raton (2013)

16. Rosales-Pérez, A., García, S., Gonzalez, J.A.: An evolutionary multiobjective model and instance selection for support vector machines with pareto-based ensembles. IEEE Trans. Evol. Comput. **21**(6), 863–877 (2017)
17. Escalante, H.J., et al.: MOPG: a multi-objective evolutionary algorithm for prototype generation. Pattern Anal. Appl. **20**(1), 33–47 (2017)
18. Gong, D., Zhou, Y.: Multi-population genetic algorithms with space partition for multi-objective optimization problems. IJCSNS Int. J. Comput. Sci. Netw. Secur. **6**, 52–58 (2006)
19. Ali, F.A., Ahmed, N.N.: Differential evolution algorithm with space partitioning for large-scale optimization problems. Intell. Syst. Appl. **11**, 49–59 (2015)
20. Arnaiz-González, Á., Díez-Pastor, J.F., Rodríguez, J.J., García-Osorio, C.: Instance selection for regression: adapting DROP. Neurocomputing **201**, 66–81 (2016)
21. Alcala-Fdez, J., et al.: KEEL Data-Mining Software Tool and Data Set Repository (2017). http://sci2s.ugr.es/keel/datasets.php