Hindawi
Complexity
Volume 2018, Article ID 6794067, 12 pages
https://doi.org/10.1155/2018/6794067

Research Article

Incremental Gene Expression Programming Classifier with Metagenes and Data Reduction

Joanna Jedrzejowicz [1] **and Piotr Jedrzejowicz** [2]

[1] *Institute of Informatics, Faculty of Mathematics, Physics and Informatics, University of Gdansk, 80-308 Gdansk, Poland*
[2] *Department of Information Systems, Gdynia Maritime University, 81-225 Gdynia, Poland*

Correspondence should be addressed to Joanna Jedrzejowicz; jj@inf.ug.edu.pl

Received 27 March 2018; Revised 8 October 2018; Accepted 24 October 2018; Published 7 November 2018

Academic Editor: Vincent Labatut

The paper proposes an incremental Gene Expression Programming classifier. Its main features include using two-level ensemble consisting of base classifiers in form of genes and the upper-level classifier in the form of metagene. The approach enables us to deal with big datasets through controlling computation time using data reduction mechanisms. The user can control the number of attributes used to induce base classifiers as well as the number of base classifiers used to induce metagenes. To optimize the parameter setting phase, an approach based on the Orthogonal Experiment Design principles is proposed, allowing for statistical evaluation of the influence of different factors on the classifier performance. In addition, the algorithm is equipped with a simple mechanism for drift detection. A detailed description of the algorithm is followed by the extensive computational experiment. Its results validate the approach. Computational experiment results show that the proposed approach compares favourably with several state-of-the-art incremental classifiers.

1. Introduction

Learning from the environment through data mining remains an important research challenge. Numerous approaches, algorithms, and techniques have been proposed during recent years to deal with the data mining tasks. An important part of these efforts focuses on mining big datasets and data streams. Barriers posed by a sheer size of the real-life datasets, on one side, and constraints on the resources available for performing the data mining task, including time and computational resources, on the other, are not easy to overcome. Additional complications, apart from the above-mentioned complexity issues, are often encountered due to the nonstationary environments.

One of the most effective approaches to mining big datasets and data streams is using online or incremental learners. Online learning assumes dealing strictly with data streams. Online learners should have the following properties [1]:

(i) Single-pass through the data.

(ii) Each example is processed very fast and in a constant period of time.

(iii) Any-time learning: the classifier should provide the best answer at every moment of time.

The incremental learning is understood as a slightly wider concept, as compared with the online learning one. Incremental learners can deal not only with data streams but also with big datasets stored in databases for which using the "one-by-one" or "chunk-by-chunk" approach could be more effective than using the traditional "batch" learners, even if no concept drift has been detected. An important feature of the incremental learners is their ability to update the currently used model using only newly available individual data instances, without having to reprocess all of the past instances.

In fact, using incremental learners is, quite often, the only possible way to extract any meaningful knowledge. Usual for the contemporary databases is a constant inflow of new data instances. Hence, the knowledge discovered in databases needs to be constantly updated, which is usually an infeasible task for classic learners. Data streams, and even stored datasets, may be affected by the so-called concept drift. In the above cases, online or incremental learners are needed.

In the paper, we propose a new version of the incremental classifier based on Gene Expression Programming (GEP) with data reduction and a metagene as the final, upper-level, classifier. Classifiers using the GEP-induced expression trees are known to produce satisfactory or very good results in terms of the classification accuracy. Our approach uses GEP-induced expression trees to construct learners with the ability to deal with large datasets environment and with a concept drift phenomenon. The rest of the paper is organized as follows. In Section 2 a brief survey of the related results is offered. In Section 3 we describe a new version of the proposed approach. Section 4 contains a detailed description of the validating computational experiment and a discussion of its results including suggestions on how to deal with the real-life datasets through the Orthogonal Experiment Design technique. Section 5 includes conclusions and ideas for future research.

2. Related Work

To meet the required properties of the online learners several approaches and techniques have been proposed in the literature. The most successful ones include sampling, windowing, and drift detecting. Sampling assumes using only some data instances or some part of instances out of the available dataset. In [14] random sampling strategy with a probabilistic removal of some instances from the training set was proposed. Later on, the idea was extended in [15]. Some more advanced sampling strategies were proposed in [16]. Effects of sampling strategy on classification accuracy were investigated in [17].

As it has been observed in the review of [18], data sampling methods for machine learning have been investigated for decades. According to the above paper, in recent years progress has been made in methods that can be broadly categorized into random sampling including density-biased and nonuniform sampling methods, active learning methods, which are the type of semisupervised learning, and progressive sampling methods, which can be viewed as a combination of the above two approaches.

Closely related to sampling is the sliding window model. Sliding window can be seen a subset that runs over an underlying collection. Several versions of the approach can be found in [19–21]. The idea is that analysis of the data stream is based on recent instances only and a limited number of the data instances, usually equal to the window size, are used to induce a classifier. In machine learning, the concept can be used for incremental mining of association rules [22]. Another interesting application of the sliding window technique is known as the high utility pattern mining [23].

For noisy environments or environments with a concept drift the key question is when and how the current model should be adopted. Possible solutions include explicit drift detection models (see the survey by Ditzler et al. [24]) or explicit partitioning approaches (see, for example, [25]).

One of the most successful approaches to incremental mining of data streams is using the drift detection techniques. The aim of the drift detection is to identify changes in statistical properties of data distribution over time. Such changes are often referred to as the concept drift. To minimize deterioration of learners accuracy caused by the concept drift, one can apply change detection tests and modify or replace a learner upon discovering the drift (see, for example, [26, 27]). The above-described approach is known as an active solution as opposed to a passive one, where the model is constantly retrained based on the most recent sample. More recently several Extreme Learning Machine (ELM) approaches to incremental learning have been discussed. For example, [28] proposed a forgetting parameters concept named FP-ELM. Recent surveys on data stream mining can be found in [24, 29].

Among incremental models, there are also those based on exploiting a power of the ensemble classifiers. Ensemble learners involve a combination of several models. Their predictions can be combined in some manner like, for example, averaging or voting to arrive at the final prediction. Ensemble learners for the data stream mining were proposed, among others, in [30–34].

One of techniques used to construct incremental classifiers is Gene Expression Programming (GEP). Gene Expression Programming was introduced in [35]. In GEP programs are represented as linear character strings of a fixed length called chromosomes which, in the subsequent fitness evaluation, evolve into expression trees without any user intervention. This feature makes GEP-induced expression trees a convenient model for constructing classifiers [36].

An improvement of the basic GEP classifiers can be achieved by combining GEP-induced weak classifiers into a classifier ensemble. In [37] two well-known ensemble techniques, bagging and boosting, were used to enhance the generalization ability of GEP classifiers. Yet another approach to building GEP-based classifier ensembles was proposed in [38]. The idea was to construct weak (base) classifiers from different subsets of attributes controlling the diversity among these subsets through applying a variant of niching technique. Further extensions and variants of GEP-induced ensemble classifiers were discussed in [39] where ideas of incremental learning and cluster-based learning were proposed. Approaches to constructing ensemble classifiers from GEP-induced weak classifiers were also studied in [40].

3. The Proposed Incremental GEP-Based Classifier

In this paper, we extend and improve the incremental GEP-based classifier proposed in [41]. In the above paper, GEP was used to induce base classifiers. Base classifiers serve to construct an ensemble of classifiers. Such an ensemble requires the application of some integration techniques like for instance majority voting, bagging or boosting. Review of the ensemble construction methods for the online learning can be found in [42]. Alternatively, a metaclassifier can be constructed following the idea of the stacked generalization [43]. In our case, such a metaclassifier is called a metagene.

Our approach follows steps proposed in [41] as far as the construction of base classifiers and respective metagenes are concerned. The algorithm for learning the best classifier using GEP works as follows. Suppose that a training dataset is given and each vector in the dataset has a correct label representing

0	1	2	3	4
OR	(>, 1, 0.57)	NOT	(≤, 10, 0.16)	···

Figure 1: Single gene example.

the class. In the initial step, the minimal and maximal values of each attribute are calculated and a random population of chromosomes is generated. Each chromosome is composed of a single gene divided into two parts as in the original head-tail method [35]. The size of the head (h) is determined by the user with the suggested size not less than the number of attributes in the dataset. The size of the tail (t) is computed as $t = h + 1$. The size of the chromosome is $h + t = 2h + 1$. For each gene, the symbols in the head part are randomly selected from the set of functions AND, OR, NOT, XOR, and NOR and the set of terminals of type (op; $attrib$; $const$), where the value of $const$ is in the range of attribute $attrib$ and op is a relational operator. The symbols in the tail part are all terminals. In Figure 1 an example of a gene is given. The start position (position 0) in the chromosome corresponds to the root of the expression tree (OR, in the example). Then, below each function branches are attached and there are as many of them as the arity of the function, 2 in our case. The following symbols in the chromosome are attached to the branches on a given level. The process is complete when each branch is completed with a terminal. The number of symbols from the chromosome to form the expression tree is denoted as the termination point. For the discussed example, the termination point is 4; therefore further symbols are not meaningful and are denoted by $\cdots$ in Figure 1. The rule corresponding to the chromosome from Figure 1 is

IF ($attribute1 > 0.57$) OR NOT ($attribute10 \leq 0.16$) THEN Class 1.

To introduce variation in the population the following genetic operators are used:

(i) mutation,

(ii) transposition of insertion sequence elements (IS transposition),

(iii) root transposition (RIS transposition),

(iv) one-point recombination,

(v) two-point recombination.

Mutation can occur anywhere in the chromosome. We consider one-point mutation which means that with a probability, called mutation rate, one symbol in a chromosome is changed. In case of a functional symbol it is replaced by another randomly selected function; otherwise for $g = (op, attrib, const)$ a random relational operator op', an attribute $attrib'$, and a constant $const'$ in the range of $attrib'$ are selected. Note that mutation can change the respective expression tree since a function of one argument may be mutated into a function of two arguments or vice versa.

Transposition stands for moving part of a chromosome to another location. Here we consider two kinds of transposable elements. In the case of transposition of insertion sequence (IS) three values are randomly chosen: a position in the chromosome (start of IS), the length of the sequence and the target site in the head, a bond between two positions. Then a cut is made in the bond defined by the target site and the insertion sequence is copied into the site of the insertion. The sequence downstream from the copied IS element loses, at the end of the head, as many symbols as the length of the transposon. Observe that since the target site is in the head, the newly created individual is always syntactically correct though it can reshape the tree quite dramatically. In the case of root transposition, a position in the head is randomly selected, the first function following this position is chosen; it is the start of the RIS element. If no function is found, then no change is performed. The length of the insertion sequence is chosen. The insertion sequence is copied at the root position and at the same time the last symbols of the head (as many as RIS length) are deleted.

For both kinds of recombination two parent chromosomes P_1, P_2 are randomly chosen and two new child chromosomes C_1, C_2 are formed. In the case of one-point recombination, one position is randomly generated and both parent chromosomes are split by this position into two parts. Child chromosomes C_1 (respectively, C_2) is formed as containing the first part from P_1 (respectively, P_2) and the second part from P_2 (and P_1). In two-point recombination two positions are randomly chosen and the symbols between recombination positions are exchanged between two parent chromosomes forming two new child chromosomes. Observe that again, in both cases, the newly formed chromosomes are syntactically correct no matter whether the recombination positions were taken from the head or tail.

During GEP learning, the individuals are selected and copied into the next generation based on their fitness and the roulette wheel sampling with elitism which guarantees the survival and cloning of the best chromosome in the next generation.

Further details on GEP operators and GEP learning can be found in [39, 40, 44].

For a fixed training set TR and fixed gene g the fitness function counts the proportion of vectors from TR classified correctly:

$$fit_{TR}(g) = \frac{\sum_{rw \in TR,\, g(rw) \text{ is true}} sg\,(\text{rw is from class 1})}{|TR|} \quad (1)$$

where

$$sg(\varphi) = \begin{cases} 1 & \text{if } \varphi \text{ is true} \\ 0 & \text{otherwise} \end{cases} \quad (2)$$

Having generated a population of genes it is possible to create a population of metagenes which corresponds to creating an ensemble classifier. The idea is as follows. Let pop be a population of genes, with each gene identified by its id. To create metagenes from pop we define the set of functions again as Boolean ones as above and set terminals equal to identifiers

0	1	2	3	4
OR	AND	$g1$	$g2$	$g3$

Figure 2: Single metagene example.

of genes. For example, the metagene mg shown in Figure 2 makes use of three genes g1, g2, and g3.

$$\begin{array}{c} g1: \\ \\ g2: \\ \\ g3: \end{array} \begin{array}{c|c|c|c} 0 & 1 & 2 & 3 \\ \hline \text{AND} & (=,2,0.8) & (=,3,2.5) & \\ \hline 0 & 1 & 2 & 3 \\ \hline \text{AND} & \text{NOT} & (=,3,0) & (<,2,0) \\ \hline 0 & 1 & & \\ \hline (=,1,0) & \ldots & & \end{array} \tag{3}$$

For a fixed attribute vector rw each terminal (i.e., gene) has a Boolean value and thus the value of metagene can be computed. For the metagene mg from Figure 2 and $rw = (1.2, 0.8, 2.5)$ we have

$$\begin{aligned} g1\,(rw) &= true, \\ g2\,(rw) &= false, \\ g3\,(rw) &= false, \\ mg\,(rw) &= true \end{aligned} \tag{4}$$

Similarly as in (1), for a fixed training set TR and fixed metagene mg the fitness function counts the proportion of vectors from the testing set classified correctly:

$$FIT_{TR}\,(mg) = \frac{\sum_{rw \in TR,\, mg(rw)\ is\ true} sg\,(\text{rw is from class 1})}{|TR|} \tag{5}$$

The incremental GEP classifier with metagenes works in rounds. In each round, a chunk of data is used to induce genes and another chunk to induce metagenes. Chunk size is one of the incremental classifier parameters. Its role is to control the frequency with which the model is updated with a view to adapt to a possible concept drift. Main assumptions for such an approach are as follows:

(i) Class labels of instances belonging to the first and second chunks are known at the outset

(ii) Class labels of instances belonging to the chunk number 3, and to all the following chunks, are immediately revealed after the class of each instance has been predicted

(iii) All instances except those belonging to the first two chunks are classified one by one in the "natural" order

Based on the above assumptions, in [2], the following procedure was implemented. In each round a chunk of training data c_1 is used to create a population of genes, next chunk of data c_2 is used to create the population of metagenes and to choose one best-fitted metagene denoted mg, and the following chunk c_3 is tested by metagene mg. In the next round, $c_1 := c_2$, $c_2 := c_3$ and next chunk is used as c_3. For further comparisons, the incremental classifier from [2] is denoted as Inc-GEP1.

Computational experiments confirmed that Inc-GEP1 performs quite well. Comparison with the state-of-the-art incremental classifiers showed that the approach outperforms, in the majority of cases, the existing solutions in terms of the classification accuracy. Unfortunately, Inc-GEP1 suffers from a high demand on computational resources which, in many situations, might prevent it from mining data streams and datasets from the big data environment. One of the reasons behind the above situation is that Inc-GEP1 has not been equipped with any adaptation mechanism providing for updating the model only upon detecting a concept drift. Instead, the model is induced anew each time after classifying a chunk of instances.

To offer more flexibility and to shorten the computation time as compared with Inc-GEP1 we propose two measures. The first is an extensive data reduction option, and the second is providing some adaptation mechanism with a view to decreasing the number of required learner updates during computations. Following the idea of the random sampling proposed for the classic (nonincremental) learners [41], in the proposed incremental learner, the user has an option to set values of the following main parameters:

(i) Chunk size (ch)

(ii) Number of the base classifiers (NB)

(iii) Number of attributes used to induce the base genes (NA)

(iv) Percent of instances used to induce the base genes (RB)

(v) Percent of instances used to induce metagenes (RM)

Each of the above options can be used to control and effectively decrease or increase the computation time of the whole process, including learning models and predicting class labels of the incoming instances. Setting value of the chunk size determines how often the learner is updated. Smaller size results in increasing the number of updates. In our case, this number can be decreased through the proposed adaptation mechanism described later in this section. The number of base classifiers used to induce metagenes influences computation time needed to perform the job. A smaller number of the base classifiers may, however, decrease the accuracy of the resulting metagenes. The number of attributes used to induce base genes should be smaller than the number of original attributes in each instance of the considered dataset. Once set, it results in selecting randomly as many attributes as required from the set of all data attributes. The random draw of attributes takes place each time when one of the base classifiers is induced. This means that for inducing each base classifier a combination of attributes is repeatedly randomly drawn. Setting percent of instances used to induce the base genes and metagenes results in randomly sampling chunks used to induce the base genes and metagenes, respectively.

```
Input: chunk C, number of base classifiers NB, number of attributes NA, percent of instances RB
Output: the population of base classifiers BC
BC ⟵ ∅
for i ⟵ 1 to NB do
    /* prepare chunk for learning                                    */
    CF ⟵ C filtered onto NA attributes chosen randomly
    NI ⟵ size(CF)×RB
    CN ⟵ ∅
    for i ⟵to NI do
        select random row r from CF
        add row r to CN
    apply GEP learning to CN ([2])
    add the best gene to base classifiers BC
return BC
```

Algorithm 1: Inducing base classifiers.

```
Input: chunk C, base classifiers BC, percent of instances RM
Output: best metagene mg
/* prepare chunk for learning metagene                              */
NI ⟵ size(C)×RM
CN ⟵ ∅
for i ⟵ 0 to NI do
    select random row r from C
    add row r to CN
apply metagene learning to CN and base classifiers BC ([2])
select best metagene mg
return mg
```

Algorithm 2: Inducing metagene.

Such filtering results in diminishing the number of instances used to induce each of the base classifiers and each of metagenes, by a given percentage.

Apart from the data reduction measures, we also propose to introduce a simple adaptation mechanism reducing unnecessary learner updates. After having used the first two data chunks to induce the initial set of base classifiers and the current metagene (mg), the following scheme is used. Class labels of instances belonging to the third chunk c_3 are predicted using mg and the average accuracy of class prediction for that chunk (av_3) is recorded. In the next step, mg is used to predict class labels of the fourth chunk c_4 and the average accuracy of prediction av_4 is calculated and recorded. If $av_4 < av_3$, then the learner is updated using c_3 and c_4 producing new current mg. Else, the current metagene is used to predict class labels of instances belonging to the next incoming chunk. The procedure is repeated until instances in all chunks have been classified. Wherever the inequality $av_i < av_{i-1}$ holds, the current metagene is replaced by a new one induced using chunks c_i and c_{i-1}. The above adaptation mechanism is denoted as ADAPT1. Alternatively, the second version of the adaptation mechanism, denoted as ADAPT2, can be used. Under ADAPT2 the current metagene is replaced by a newly induced one only after the average classification accuracy for two consecutive chunks is worse than the accuracy produced by the metagene induced for their predecessor chunk. The procedure using ADAPT1 is shown as Algorithm 3 and the case for ADAPT2 is omitted, as being similar. The incremental classifier with data reduction and ADAPT1 mechanism is further on referred to as Inc-GEP2. Such classifier equipped with ADAPT2 mechanism is further on referred to as Inc-GEP3.

Procedures for inducing base classifiers and metagenes are shown as Algorithms 1 and 2, respectively. In both cases, the fitness function is an accuracy of the class label prediction calculated over the respective chunk of data.

4. Computational Experiment Results

To evaluate the performance of the proposed approach we have carried out the computational experiment over a representative group of the publicly available 2-classes benchmark datasets including large datasets and datasets often used to test incremental learning algorithms. Datasets used in the experiment are shown in Table 1.

In Table 2 experiment settings used in Inc-GEP2 and Inc-GEP3 are shown. There are 4 main parameters affecting the proposed classifiers performance. Chunk size refers to the number of instances classified one by one without interruption using the current metagene. The number of attributes

```
Input: dataset D, chunk size ch, number of base classifiers NB
Output: overall prediction accuracy
/* induce NB base classifiers using the first chunk and best metagene
   using the second chunk                                                  */
(1) dataTrain ⟵ first ch rows from D
(2) dataTrainM ⟵ next ch rows from D
(3) apply Algorithm 1 to dataTrain to induce NB base classifiers BC
(4) apply Algorithm 2 to dataTrainM and BC to induce metagene mg
(5) dataTest ⟵ next ch rows from D
(6) ac ⟵ accuracy of classification performed on dataTest by metagene mg
(7) acV ⟵ ac
(8) while rows in D not considered yet do
(9)     dataTestN ⟵ next ch rows from D
(10)    acN ⟵ accuracy of classification performed on dataTestN by metagene mg
(11)    acV ⟵ acV + acN
(12)    dataTrain ⟵ dataTrainM
(13)    dataTrainM ⟵ dataTest
(14)    dataTest ⟵ dataTestN
(15)    if acN < ac then
            /* metagene updated by new learning                            */
(16)        apply Algorithm 1 to dataTrain to induce base classifiers BC
(17)        apply Algorithm 2 to dataTrainM to induce metagene mg
(18)    ac ⟵ acN
(19) noC ⟵ number of chunks −2
(20) acV ⟵ acV/noC
(21) return acV
```

Algorithm 3: Incremental classifier with data reduction and ADAPT1 adaptation mechanism.

refers to the number of randomly selected attributes used to induce each gene. Reduction rate reflects the percent of both instances used to induce genes and instances used to induce metagenes. Number of classifiers refers to the number of base classifiers (genes). Method of setting values of the above parameters is explained later. Other settings including the number of iterations in GEP (set at 100) and probabilities of applying genetic operators (set as in [2]) have been the same throughout the whole experiment.

In Table 3 mean classification accuracy of Inc-GEP1, Inc-GEP2, and Inc-GEP3 is shown. Accuracy and standard deviation have been calculated as mean values obtained over 20 runs with parameter settings as shown in Table 2. For the Inc-GEP1 chunk size and the number of attributes are identical as in the case of the Inc-GEP2 and Inc-GEP3. In Inc-GEP1, however, there is no reduction with respect to the percentage of genes used to induce base classifiers and metagenes. Additionally, in Inc-GEP1 base classifiers and metagenes are induced using the full set of attributes.

Parameter values shown in Table 2 have been selected through the Orthogonal Experimental Design (OED) method. Since there are four main factors affecting classifier performance, it has been decided to use an L9 orthogonal array to identify the influence of 4 different independent variables on classifier performance. For each variable 3 level values have been set. Selection of the level values was arbitrary, albeit based on common sense.

The decision to use the OED method has been preceded by a comparison of mean classification accuracy values for each dataset and each combination of main factors out of 9 combinations under analysis. Thus, for each dataset, we had 9 groups of samples, each containing 10 classification accuracies obtained by running the considered classifier for 10 times for each combination of factors. The one-way ANOVA with the null hypotheses stating that samples in all groups are drawn from populations with the same mean values has

Table 1: Benchmark datasets used in the experiment. The table is reproduced from [2] (under the Creative Commons Attribution License/public domain).

Dataset	Source	Instances	Attributes
Airlines	[3]	539383	8
Bank M.	[4]	45211	10
Banknote Auth	[4]	1372	5
Breast Cancer	[4]	263	10
Chess	[5]	503	9
Diabetes	[4]	768	9
Electricity	[6]	45312	6
Heart	[4]	303	14
Image	[4]	2086	19
Internet Adv	[4]	3279	1559
Ionosphere	[4]	351	35
Luxemburg	[5]	1901	32
Sea	[7]	5000	4
Usenet2	[7]	1500	100

Table 2: Experiment settings for algorithms Inc-GEP2 and Inc-GEP3.

Dataset name	Chunk size	No of attrib.	Reduction rate (%)	No of classifiers
Airlines	10000	4	50	20
Bank M	400	13	10	20
Banknote Auth.	120	4	80	50
Breast cancer	30	4	20	20
Chess	50	7	50	20
Diabetes	60	3	80	30
Electricity	4000	4	50	30
Heart	30	11	90	20
Image	100	15	50	20
Internet Adv.	500	500	70	60
Ionosphere	60	30	50	20
Luxemburg	50	11	80	30
Sea	2500	2	10	30
Usenet2	20	60	50	20

Table 3: Computational experiment results (mean accuracy and standard deviation, %).

Dataset name	Inc-GEP1		Inc-GEP2		Inc-GEP3	
	Accuracy	+/-	Accuracy	+/-	Accuracy	+/-
Airlines	62.56	2.339	63.79	2.029	61.24	3.108
Banknote auth	93.07	1.436	93.01	1.356	93.41	1.199
Bank M	93.31	1.867	88.79	0.461	89.98	0.985
Breast cancer	82.13	0.720	74.38	0.654	76.37	0.536
Chess	85.94	1.676	84.16	0.897	77.34	0.549
Diabetes	85.01	0.932	62.13	0.245	65.33	0.453
Electricity	88.13	3.247	95.39	1.298	92.67	1.541
Heart	83.91	1.127	78.95	1.457	77.33	0.914
Image	86.6	2.358	79.04	1.298	75.58	1.598
Internet Adv.	91.47	0.793	95.67	0.033	94.92	0.055
Ionosphere	92.9	1.002	89.61	0.972	87.06	0.839
Luxemburg	100.00	0.000	100.00	0.000	100.00	0.000
Sea	81.12	1.393	83.28	0.356	84.56	0.541
Usenet2	78.15	2.362	74.24	1.885	69.78	1.177

shown that, for all considered datasets with the exception of the Bank Marketing dataset, null hypotheses should be rejected. This finding assures sensibility of searching for the best combination of factor values for each of the considered datasets.

The procedure of the orthogonal experiment and selection of the parameter values is shown below on the example of the Sea dataset. The similar procedure has been applied to all considered datasets.

In Table 4 factor (term) levels for the orthogonal array used in the experiment with the Sea dataset are shown. In Table 5 response values representing classification accuracy using the Inc-GEP2 classifier are displayed. The first column shows factor level numbers. Next ten columns contain response values. The last column contains the average of responses.

Response table for signal-to-noise ratio shown in Table 6 indicates that key role in maximizing the discussed ratio plays the number of attributes while data in Table 7 showing the response table for classification accuracy means indicate that key factor in maximizing accuracy plays the number of classifiers and next the number of attributes. The response table for signal-to-noise ratios contains a row for the average signal-to-noise ratio for each factor level, Delta, and rank. Delta is the difference between the maximum and minimum average response for the factor. The response table for means shows the size of the effect by taking the difference between the highest and lowest characteristic average for a factor. Ranks in a response table allow to quickly identify which factors have the largest effect. All factors, however, have statistically significant effects on response. This is confirmed by the main effect plot for means shown in Figure 3. Main effect plot is constructed by plotting the means for each value of a variable. A line connects the points for each variable. When the line is horizontal (parallel to the x-axis), there is no main effect present. The response mean is the same across

Table 4: Factor levels for the orthogonal array used in the experiment with the Sea dataset.

Factor level	Window size	No of attrib.	Reduction rate (%)	No of classifiers
1	5000	3	30	30
2	2500	2	20	20
3	100	1	10	10

Table 5: Experiment response results (Sea dataset, Inc-GEP2 classifier).

Levels	1	2	3	4	5	6	7	8	9	10	AV
1,1,1,1	0.816	0.802	0.798	0.803	0.806	0.797	0.820	0.815	0.820	0.791	0.807
1,2,2,2	0.815	0.813	0.804	0.800	0.802	0.794	0.803	0.820	0.811	0.816	0.808
1,3,3,3	0.736	0.720	0.744	0.690	0.735	0.735	0.737	0.743	0.711	0.710	0.726
2,1,2,3	0.823	0.837	0.839	0.818	0.828	0.834	0.837	0.832	0.821	0.803	0.827
2,2,3,1	0.837	0.835	0.832	0.831	0.833	0.826	0.831	0.837	0.830	0.833	0.833
2,3,1,2	0.756	0.727	0.742	0.758	0.759	0.736	0.742	0.758	0.742	0.761	0.748
3,1,3,2	0.796	0.812	0.799	0.801	0.809	0.812	0.807	0.812	0.791	0.793	0.803
3,2,1,3	0.793	0.771	0.780	0.804	0.809	0.802	0.808	0.819	0.800	0.805	0.799
3,3,2,1	0.759	0.771	0.785	0.765	0.740	0.758	0.761	0.752	0.774	0.737	0.760

Table 6: Response Table for signal-to-noise ratios: Sea dataset.

Level	Window	Attributes	Reduction	Class.no
1	-2.080	-2.563	-2.092	-2.128
2	-1.920	-1.798	-1.962	-2.093
3	-2.166	-1.806	-2.113	-1.946
Delta	0.246	0.765	0.151	0.182
Rank	2	1	4	3

Table 7: Response table for means: Sea dataset.

Level	Window	Attributes	Reduction	Class.no
1	0.9214	0.9142	0.9242	0.9145
2	0.9402	0.9366	0.9334	0.9329
3	0.9254	0.9362	0.9294	0.9396
Delta	0.0189	0.0224	0.0092	0.0251
Rank	3	2	4	1

all factor levels. On the other hand, when the line is not horizontal, there is a main effect present and the response mean is not the same across all factor levels. The steeper the slope of the line, the greater the magnitude of the main effect.

As data in Table 5 indicate, the best combination of factor levels for the Sea dataset is window (chunk) size 2500, 2 attributes for inducing base classifiers, 10% of instances used to induce genes and, respectively, metagenes, and 30 classifiers. Similar analysis has been performed for all considered datasets with a view to find out the best combination of parameter (factor) values.

Orthogonal array analysis can be also carried out with respect to the computation times. For example, in the case of the Sea dataset the respective response table for computation times means indicates that key role in minimizing computation time plays the window size and number of classifiers used to construct the ensemble. The respective main effects plot displaying how the considered factors affect computation times for the Sea dataset is shown in Figure 4. In this Figure "means" refers to times in seconds needed to classify a single instance. In Table 8 comparison between mean computation times for all the considered dataset and for settings of parameters from Table 2 is shown. Respective values refer to times in seconds needed to classify 100 instances by the considered algorithms run on Dell Precision 3520 workstation with Xeon processor and 16 GB RAM. Columns Speed-up1 and Speed-up2 contain speed-up factors comparing Inc-GEP1 with Inc-GEP2 and Inc-GEP1 with Inc-GEP3, respectively. As can be observed from Table 8, there are significant differences in computation times needed to run algorithms under comparison. On average, the proposed Inc-GEP2 classifier is over 2 times quicker as compared with the incremental Gene Expression Programming with metagenes without data reduction (Inc-GEP1). Moreover, the proposed Inc-GEP3 classifier is, on average, over 7 times quicker than the control algorithm Inc-GEP1. To properly evaluate both Inc-GEP2 and Inc-GEP3 one has to evaluate also their performance in terms of the classification accuracy. Assuming equal variances, one-way ANOVA allows observing that null hypothesis stating that all three mean accuracies are equal under the confidence level 0.05 holds. Hence, the alternative hypothesis stating that not all the considered means are equal should be rejected. The above finding is confirmed by Fisher and Tukey tests.

In Table 9 comparison of the proposed GEP-based incremental classifiers with some literature reporting state-of-the-art incremental classifiers in terms of the mean classification accuracy is shown. The abbreviations used for incremental classifiers are as follows: FTDD, Fisher Test Drift Detection; IncSVM, Incremental SVM; EDDM, Early Drift Detection Method; IncN-B, Incremental Naïve Bayes; KFCM, Online distance based classifier with Kernel Fuzzy C-means; IncEnsemble, Incremental Ensemble; and FISH, Unified Instance Selection Algorithm.

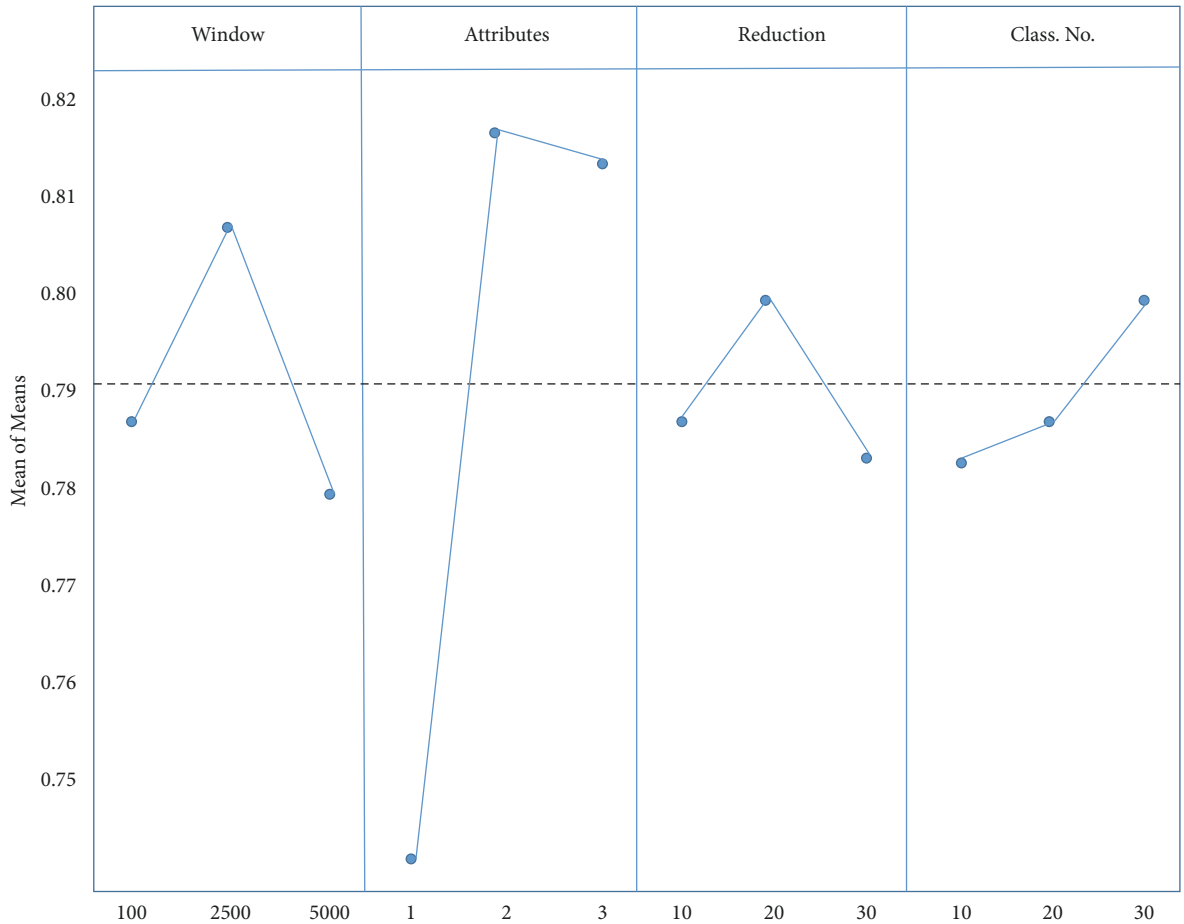

Figure 3: Main effects plot for classification accuracy means: Sea dataset.

Table 8: Mean computation times (seconds per 100 instances).

Dataset	Inc-GEP1	Inc-GEP2	Inc-GEP3	Speed-up1	Speed-up2
Airlines	1.07	0.39	0.16	2.7	6.7
Banknote auth.	6.63	4.30	1.09	1.5	6.1
Bank M.	0.75	0.39	0.20	1.9	3.8
Breast	26.24	11.03	1.14	2.4	23.0
Chess	5.37	3.58	2.39	1.5	2.3
Diabetes	4.56	3.65	2.86	1.3	1.6
Electricity	7.27	2.53	0.50	2.9	14.5
Heart	18.81	10.56	5.61	1.8	3.4
Image	14.43	2.92	0.91	4.9	15.8
Internet Ad	13.08	5.58	2.10	2.3	6.2
Ionosphere	10.54	7.69	3.70	1.4	2.8
Luxemburg	0.79	0.26	0.16	3.0	5.0
Sea	0.76	0.34	0.18	2.3	4.3
Usenet2	17.80	8.07	3.13	2.2	5.7

Table 9: Comparison of the proposed GEP-based incremental classifiers with some literature reporting incremental classifiers in terms of the mean classification accuracy.

Dataset	Inc-GEP1	Inc-GEP2	Literature reported acc	Incremental classifier	Source
Airlines	63.79	61.24	65.44	FTDD	[8]
Bank M.	88.79	89.98	86.90	IncSVM	[9]
Breast C.	74.38	76.37	72.20	IncSVM	[10]
Chess	84.16	77.34	71.80	EDDM	[11]
Diabetes	62.13	65.33	75.70	IncN-B	[12]
Electricity	95.39	92.67	90.70	KFCM	[13]
Heart	78.95	77.33	83.80	IncSVM	[10]
Ionosphere	89.61	87.06	92.40	IncEnsemble	[12]
Luxemburg	100.00	100.00	88.11	FISH2	[5]

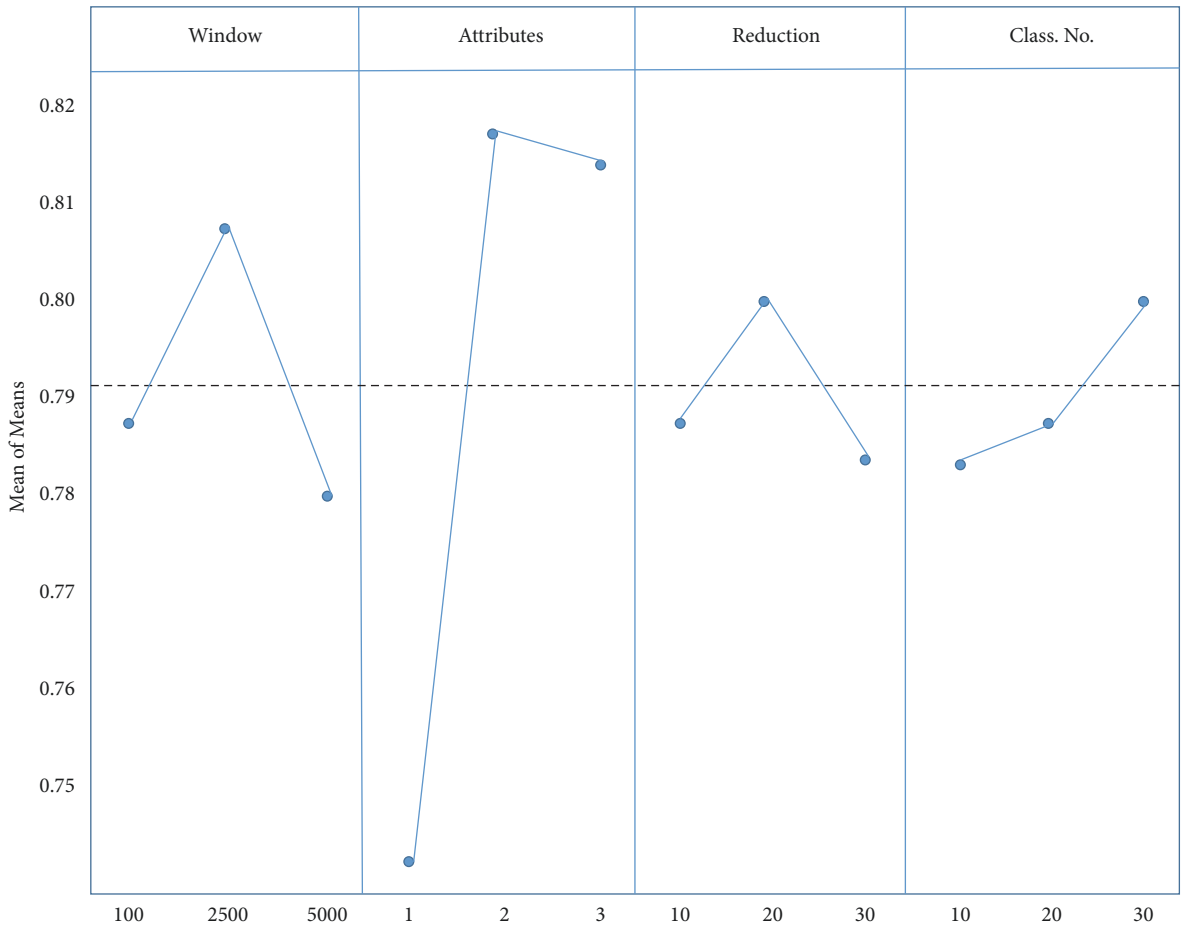

Figure 4: Main effects plot for classification time means: Sea dataset.

From Table 9 it can be seen that the proposed classifiers perform well and are competitive to several other approaches. In several cases, GEP-based incremental classifiers outperform earlier available solutions.

5. Conclusions

The main contribution of the paper is to propose the incremental Gene Expression Programming classifier with metagenes and data reduction. The concept of metagenes increases the classification accuracy while data reduction allows controlling computation time. The proposed approach extends earlier incremental GEP-based classifier [2]. Additionally, the extended version contains a simple drift detection mechanism allowing dealing more effectively with data streams.

Another important novelty introduced in the paper is using the Orthogonal Experimental Design principles to set up classifier parameters values. The approach allows us to easily evaluate the statistical importance of main parameters (factors) showing through main effects plots and the

respective response tables key factors and their influence on classifier performance and signal-to-noise ratios.

An extensive computational experiment confirms that the proposed classifier offers better performance in respect to the required computation times as compared with its earlier version. At the same time, it provides similar results in terms of classification accuracy. The algorithm offers also scalability through the possibility of adjusting computation times to the user needs, which might be a useful feature even at a cost of possibly a bit lower classification accuracy.

Comparison of the proposed GEP-based incremental classifiers with some literature reporting state-of-the-art incremental classifiers in terms of the mean classification accuracy proves that our approach offers quite satisfactory solutions, outperforming in many cases the existing methods. The proposed approach can be useful in data analytics and big data processing where single-pass limited-memory models enabling a treatment of big data within a streaming setting are increasingly needed [45].

Future research would concentrate on incorporating more sophisticated drift detection mechanisms and to further improve efficiency by implementing the algorithm in a parallel environment.

Acronyms and Abbreviations

GEP:	Gene Expression Programming
IS:	Sequence insertion
RIS:	Root transposition
TR:	Training set
pop:	Population of genes
mg:	Metagene
fit:	Fitness function for genes
FIT:	Fitness function for metagenes
ch:	Chunk size
NB:	Number of base classifiers
NA:	Number of base classifiers
RB:	Percent of instances used to induce base classifiers
RM:	Percent of instances used to induce metagenes
BC:	Base classifiers
ADAPT1, ADAPT2:	Adaptation procedure in two versions
Inc-GEP1:	Incremental classifier
Inc-GEP2:	Incremental classifier with adaptation ADAPT1
Inc-GEP3:	Incremental classifier with adaptation ADAPT2.

Data Availability

Previously reported datasets data were used to support this study and are publically available at UCI Machine Learning Repository (see [36]) with respect to Bank Marketing, Banknote Authentication, Breast Cancer, Diabetes, Heart, Image, Internet Advertisement, and Ionosphere. Airlines dataset is publically available at Open Machine Learning site (https://www.openml.org/). Chess and Luxemburg datasets are available from Indre Zliobaite (see [4]). Electricity dataset is publically available from UCI Repository–Massive Online Analysis (see [15]). SEA and Usenet2 datasets are publically available from Joaquin Vanschoren et al. (see [41]). These datasets are cited at relevant places within the text.

Conflicts of Interest

The authors declare that they have no conflicts of interest.

References

[1] L. I. Kuncheva, "Classifier ensembles for changing environments," in *Proceedings of the 5th International Workshop Multiple Classifier Systems MCS '04*, F. Roli, J. Kittler, and T. Windeatt, Eds., vol. 3077 of *Lecture Notes in Computer Science*, pp. 1–15, Springer, Berlin, Germany, 2004.

[2] J. Jedrzejowicz and P. Jedrzejowicz, "Incremetal GEP-Based Ensemble Classifier," in *Intelligent Decision Technologies 2017*, vol. 72 of *Smart Innovation, Systems and Technologies*, pp. 61–70, Springer International Publishing, Cham, 2018.

[3] Airlines dataset, 2017.

[4] M. Lichman, "Uci machine learning repository," 2013.

[5] I. Žliobait, "Combining similarity in time and space for training set formation under concept drift," *Intelligent Data Analysis*, vol. 15, no. 4, pp. 589–611, 2011.

[6] Massive Online Analysis, Uci machine learning repository, 2013.

[7] J. Vanschoren, J. N. van Rijn, B. Bischl, and L. Torgo, Openml: networked science in machine learning, 2014.

[8] D. R. Cabral and R. S. Barros, "Concept drift detection based on Fisher's exact test," *Information Sciences*, vol. 442/443, pp. 220–234, 2018.

[9] K. Wisaeng, "A comparison of different classification techniques for bank direct marketing," *International Journal of Soft Computing and Engineering*, vol. 3, no. 4, pp. 116–119, 2013.

[10] L. Wang, H.-B. Ji, and Y. Jin, "Fuzzy Passive-Aggressive classification: A robust and efficient algorithm for online classification problems," *Information Sciences*, vol. 220, pp. 46–63, 2013.

[11] I. Žliobaitė, "Controlled Permutations for Testing Adaptive Classifiers," in *Discovery Science*, vol. 6926 of *Lecture Notes in Computer Science*, pp. 365–379, Springer Berlin Heidelberg, Berlin, Heidelberg, 2011.

[12] S. B. Kotsiantis, "An incremental ensemble of classifiers," *Artificial Intelligence Review*, vol. 36, no. 4, pp. 249–266, 2011.

[13] J. Jędrzejowicz and P. Jędrzejowicz, "Distance-based online classifiers," *Expert Systems with Applications*, vol. 60, pp. 249–257, 2016.

[14] J. S. Vitter, "Random sampling with a reservoir," *ACM Transactions on Mathematical Software*, vol. 11, no. 1, pp. 37–57, 1985.

[15] S. Chaudhuri, R. Motwani, and V. Narasayya, "On Random Sampling over Joins," *SIGMOD Record*, vol. 28, no. 2, pp. 263–273, 1999.

[16] J. Xu, Y. Y. Tang, B. Zou, Z. Xu, L. Li, and Y. Lu, "The generalization ability of online svm classification based on markov sampling," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 26, no. 3, pp. 628–639, 2015.

[17] M. S. Esfahani and E. R. Dougherty, "Effect of separate sampling on classification accuracy," *Bioinformatics*, vol. 30, no. 2, pp. 242–250, 2014.

[18] A. ElRafey and J. Wojtusiak, "Recent advances in scaling-down sampling methods in machine learning," *Wiley Interdisciplinary Reviews. Computational Statistics (WIREs)*, vol. 9, no. 6, e1414, 13 pages, 2017.

[19] S. K. Tanbeer, C. F. Ahmed, B.-S. Jeong, and Y.-K. Lee, "Sliding window-based frequent pattern mining over data streams," *Information Sciences*, vol. 179, no. 22, pp. 3843–3865, 2009.

[20] M. Deypir, M. H. Sadreddini, and S. Hashemi, "Towards a variable size sliding window model for frequent itemset mining over data streams," *Computers & Industrial Engineering*, vol. 63, no. 1, pp. 161–172, 2012.

[21] H. Chen, L. Shu, J. Xia, and Q. Deng, "Mining frequent patterns in a varying-size sliding window of online transactional data streams," *Information Sciences*, vol. 215, pp. 15–36, 2012.

[22] C. Lee, C. Lin, and M. Chen, "Sliding-window filtering: an efficient algorithm for incremental mining," in *Proceedings of the 10th International Conference on Information and Knowledge Management, CIKM '01*, pp. 263–270, ACM, New York, NY, USA, 2001.

[23] H. Ryang and U. Yun, "High utility pattern mining over data streams with sliding window technique," *Expert Systems with Applications*, vol. 57, pp. 214–231, 2016.

[24] G. Ditzler, M. Roveri, C. Alippi, and R. Polikar, "Learning in nonstationary environments: a survey," *IEEE Computational Intelligence Magazine*, vol. 10, no. 4, pp. 12–25, 2015.

[25] J. P. Fan, J. Zhang, K. Z. Mei, J. Y. Peng, and L. Gao, "Cost-sensitive learning of hierarchical tree classifiers for large-scale image classification and novel category detection," *Pattern Recognition*, vol. 48, no. 5, pp. 1673–1687, 2015.

[26] C. Alippi and M. Roveri, "Just-in-time adaptive classifiers - Part I: Detecting nonstationary changes," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 19, no. 7, pp. 1145–1153, 2008.

[27] J. Gama, P. Medas, G. Castillo, and P. Rodrigues, "Learning with drift detection," in *Advances in Artificial Intelligence—SBIA 2004: Proceedings of the 17th Brazilian Symposium on Artificial Intelligence, Sao Luis, Maranhao, Brazil, September 29–Ocotber 1, 2004*, vol. 3171 of *Lecture Notes in Computer Science*, pp. 286–295, Springer, Berlin, Germany, 2004.

[28] D. Liu, Y. Wu, and H. Jiang, "FP-ELM: An online sequential learning algorithm for dealing with concept drift," *Neurocomputing*, vol. 207, pp. 322–334, 2016.

[29] B. Krawczyk, L. L. Minku, J. Gama, J. Stefanowski, and M. Woźniak, "Ensemble learning for data stream analysis: A survey," *Information Fusion*, vol. 37, pp. 132–156, 2017.

[30] A. Bifet, G. Holmes, B. Pfahringer, R. Kirkby, and R. Gavaldà, "New ensemble methods for evolving data streams," in *Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '09)*, pp. 139–147, Paris, France, July 2009.

[31] P. Zhang, X. Zhu, J. Tan, and L. Guo, "Classifier and cluster ensembles for mining concept drifting data streams," in *Proceedings of the IEEE International Conference on Data Mining*, pp. 1175–1180, 2010.

[32] I. Czarnowski and P. Jędrzejowicz, "Ensemble classifier for mining data streams," in *Proceedings of the 18th International Conference in Knowledge Based and Intelligent Information and Engineering Systems, KES '14*, P. Jedrzejowicz, L. C. Jain, R. J. Howlett, and I. Czarnowski, Eds., vol. 35 of *Procedia Computer Science*, pp. 397–406, Elsevier, Gdynia, Poland, 2014.

[33] X.-C. Yin, K. Huang, and H.-W. Hao, "De2: Dynamic ensemble of ensembles for learning nonstationary data," *Neurocomputing*, vol. 165, pp. 14–22, 2015.

[34] D. Mejri, R. Khanchel, and M. Limam, "An ensemble method for concept drift in nonstationary environment," *Journal of Statistical Computation and Simulation*, vol. 83, no. 6, pp. 1115–1128, 2013.

[35] C. Ferreira, "Gene expression programming: a new adaptive algorithm for solving problems," *Complex Systems*, vol. 13, no. 2, pp. 87–129, 2001.

[36] C. Ferreira, *Gene Expression Programming: Mathematical Modeling by an Artificial Intelligence*, vol. 21 of *Studies in Computational Intelligence*, Springer, Berlin, Germany, 2006.

[37] Q. Li, W. Wang, S. Han, and J. Li, "Evolving classifier ensemble with gene expression programming," in *Proceedings of the 3rd International Conference on Natural Computation, ICNC '07*, vol. 3, pp. 546–550, China, 2007.

[38] W. Jiang, T. Changjie, Z. Jun et al., "An attribute-oriented ensemble classifier based on niche gene expression programming," in *Proceedings of the 3rd International Conference on Natural Computation, ICNC '07*, vol. 3, pp. 525–529, China, August 2007.

[39] J. Jędrzejowicz and P. Jędrzejowicz, "A family of gep-induced ensemble classifiers," in *Proceedings of the 1st International Conference Computational Collective Intelligence, Semantic Web, Social Networks and Multiagent Systems, ICCCI '09*, N. T. Nguyen, R. Kowalczyk, and S.-M. Chen, Eds., vol. 5796 of *Lecture Notes in Computer Science*, pp. 641–652, Springer Berlin Heidelberg, 2009.

[40] J. Jędrzejowicz and P. Jędrzejowicz, "Experimental evaluation of two new GEP-based ensemble classifiers," *Expert Systems with Applications*, vol. 38, no. 9, pp. 10932–10939, 2011.

[41] J. Jedrzejowicz and P. Jedrzejowicz, "Gene expression programming ensemble for classifying big datasets," in *Proceedings of the Computational Collective Intelligence - 9th International Conference, ICCCI '17*, T. N. Ngoc, A. George, P. Jedrzejowicz, B. Trawinski, and G. Vossen, Eds., vol. volume 10449 of *Lecture Notes in Computer Science*, pp. 3–12, Springer, Berlin, Germany, 2017.

[42] A. Fern and R. Givan, "Online ensemble learning: an empirical study," *Machine Learning*, vol. 53, no. 1-2, pp. 71–109, 2003.

[43] D. H. Wolpert, "Stacked generalization," *Neural Networks*, vol. 5, no. 2, pp. 241–259, 1992.

[44] J. Jedrzejowicz and P. Jedrzejowicz, "Gep-induced expression trees as weak classifiers," in *Proceedings of the 8th Industrial Conference Advances in Data Mining, Medical Applications, E-Commerce, Marketing, and Theoretical Aspects, ICDM '08*, P. Perner, Ed., vol. 5077 of *Lecture Notes in Computer Science*, pp. 129–141, Springer, Berlin, Germany, 2008.

[45] B. Hammer, H. He, and T. Martinetz, "Learning and modeling big data," in *Proceedings of the 22th European Symposium on Artificial Neural Networks, ESANN '14*, 2014.