Applied Soft Computing Journal 75 (2019) 284–297

Contents lists available at ScienceDirect

Applied Soft Computing Journal

journal homepage: www.elsevier.com/locate/asoc

Exploiting fractal dimension and a distributed evolutionary approach to classify data streams with concept drifts

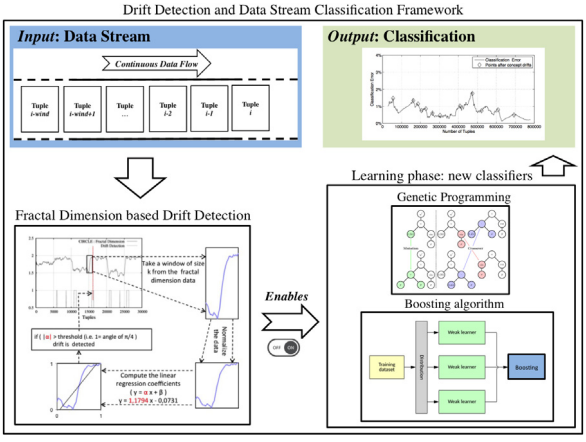
Gianluigi Folino *, Massimo Guarascio, Giuseppe Papuzzo

ICAR-CNR, Via P.Bucci 7/C, Univ. della Calabria, 87036 Rende (CS), Italy

HIGHLIGHTS

- Proposes a framework based on a distributed GP ensemble algorithm for coping with different types of concept drift.
- Experiments to assess the capacity of the framework to detect the drift and quickly respond to it.
- The fractal-based drift detection strategy proposed is comparable in terms of accuracy with well-recognized drift detection algorithms.

GRAPHICAL ABSTRACT



ARTICLE INFO

Article history:
Received 28 February 2018
Received in revised form 2 August 2018
Accepted 2 November 2018
Available online 15 November 2018

ABSTRACT

Evolutionary algorithms, i.e., Genetic Programming (GP), have been successfully used for the task of classification, mainly because they are less likely to get stuck in the local optimum, can operate on chunks of data and allow to compute more solutions in parallel. Ensemble techniques are usually more accurate than component learners constituting the ensemble and can be built in an incremental way, improving flexibility, adapting to changes and maintaining part of the history present in the data. This paper proposes a framework based on a distributed GP ensemble algorithm for coping with different types of concept drift for the task of classification of large data streams. The framework is able to detect changes in a very efficient way using only a detection function based on the fractal dimension, which can also works on new incoming unclassified data. Thus, a distributed GP algorithm is performed only when a change is detected in order to improve classification accuracy and this, together with the exploitation of an adaptive procedure, permits to answer in short time to these changes. Experiments are conducted on a real and on some artificial datasets in order to assess the capacity of the framework to detect the drift and quickly respond to it.

1. Introduction

Nowadays, people and systems overload data centers and storage systems with an exponential generation of large data streams. This rapid growth of data is mainly due to advances in digital sensors, computation, communications, and storage technologies

* Corresponding author.
E-mail addresses: gianluigi.folino@icar.cnr.it (G. Folino), massimo.guarascio@icar.cnr.it (M. Guarascio), giuseppe.papuzzo@icar.cnr.it (G. Papuzzo).

https://doi.org/10.1016/j.asoc.2018.11.009

that have created huge collections of data and, specifically, the term *Big Data* is used to define this phenomenon [1].

Nevertheless, this data can be generated by several and heterogeneous source types. In literature, three types of providers for Big Data and Data Streams are identified [2]. A typical example of *Human-Sourced* information is represented by Social Networks, where human experiences and interactions by means of the actions that they perform on social media daily are recorded. The traditional business systems and websites are common providers for *Process-Mediated Sourced* data (e.g. stock and foreign currency exchange markets). These systems record and monitor business events of interest (e.g. medical records and commercial transactions). Finally, the *machine-generated sourced* data are produced by automatic systems such as sensors and networked devices. The aim of these systems is to measure and to record events in the physical world (e.g. weather, cyber security and surveillance videos/images, mobile tracking, energy saving systems and sensor networks).

Although semantically different, all these data exhibit an interesting common feature: they have an ever-changing nature that makes them highly difficult to analyze. For example, the preferences and the interests of a person can evolve over time, the data recorded by atmospheric sensors can quickly change during a natural disaster and, in the same way, during a cyber attack, the number and the types of the connections can vary just as quickly.

In all these scenarios, we believe that the capability of identifying (*as soon as possible*) abrupt changes in the data, in order to determine when updating the classification/prediction models, is a relevant and challenging problem. To this aim, we defined a framework for coping with different types of *concept drift* and able to handle large amounts of data.

1.1. Literature overview

Generally, the traditional data mining algorithms assume that data is static, i.e., the concept to learn, described by a set of predefined features, is not affected by modifications due to the external environment changes. On the contrary, in the applications mentioned above, a concept may drift due to several factors, i.e., when the underlying distribution of the data changes. It can cause a serious degradation of the model performance; therefore its detection enables the model to revise itself and promptly restore its classification accuracy.

Several approaches have been proposed in literature to tackle the concept drift problem [3], for example, the incremental (online) systems [4,5] are able to update the underlying model as new data instances arrive. In particular, these methods build a model that represents the entire data stream and continuously refine their model as data flows. On the contrary, maintaining a unique and up-to-date model could be not a good choice, as previously trained classifiers would be discarded and an important part of any information could be lost. However, updating the model, as soon as new data arrive, might not be practicable as usually the stream of data is very fast and in many cases, it is not possible to store all the necessary information in memory. Moreover, in many real application scenarios, detecting and labeling new instances is a very expensive and time-consuming task because usually it must be performed by a human operator. A common approach to overcome this problem is to detect the changes and to update the model only if a concept drift happens.

Some research lines focused on ensemble learning algorithms as an example of incremental systems and were successfully applied for classifying data streams in many difficult environments [6–8]. The interest for these approaches is due to several reasons: (i) they allow to improve the predictive accuracy combining several base classifiers, (ii) they can be learnt and updated incrementally and (iii) they can cope with large datasets and big data environments by using parallel and distributed architecture and by extracting small portions of the original dataset for training the components of the ensemble. For instance, in [9], an approach exploiting support vector regressors (SVR) models as predictors is proposed for the classification of large datasets. The authors pointed out that the SVRs may be affected from over-fitting when they are learnt on non-uniform or imbalanced data and, to overcome this issue, an interesting strategy to extract small training subsets (TS), representative of the whole dataset, is presented. This strategy could be usefully adopted in an ensemble-based framework.

1.2. Our approach

In this work, we propose a scalable and effective approach, based on the fractal dimension, for identifying concept drifts. Specifically, the technique exploits the fractal dimension to provide an indicator able to measure how much the data distribution is changed, thus to enable the drift detection method (more details will be provided in Section 5.3). Finally, when a drift is detected, the updating phase of the classification models takes place.

Our framework deals with the problem of the large data streams classification where concept drifts may occur (both rarely as well as frequently). In particular, the proposed approach employs an *ensemble of GP-Based classifiers* in order to mitigate the loss of accuracy due to the concept drift. Specifically, the ensemble of classifiers is incrementally built by using a distributed GP-based algorithm, BoostCGPC, introduced in [10]. Then, a weighting scheme (detailed in Section 5.4) and a pruning method are used for reducing the accuracy degradation in the transitory phase after the drift detection. The former allows to weight the base models according to their accuracy on the new data, while the second permits the removal of the obsolete or inaccurate ones.

A very preliminary version of the paper has been presented in [11]; the main idea of using fractal dimension and GP is already present, but only preliminary experiments on the capacity of fractal dimension to detect drift in artificial datasets have been conducted. Here, we describe in detail the framework and the software architecture and an extensive experimentation to prove the effectiveness and the efficiency of the proposed approach is also presented.

More in detail, the architecture of the framework consists of two main modules: a fractal dimension-based function for detecting concept drifts in the *non-stationary* phase and an ensemble of GP-based classifiers used in the training (*stationary*) phase. Indeed, the distributed GP algorithm is executed only when a change is detected with a considerable gain in terms of efficiency. The proposed solution allows to identify several kinds of concept drift in a small amount of time, even for real-time data streams. To this aim, the authors of [12] define a really efficient heuristic to compute the fractal dimension on streaming data by using a constant amount of memory, which could further speedup the detection phase.

Predicting when a drift can happen is a very difficult task, as the drifts can be both really rare (e.g. the intrusion detection problem) or very frequent (e.g. updates detection in the news flow during unexpected events such as natural disasters). As labeling is a costly and time-consuming operation [13], in our approach, changes are detected only on the basis of the fractal dimension, which can be computed directly on the new chunks of unlabeled data. The main advantage of our approach is that labeled instances are required only when a new drift is discovered, which can be quite infrequent.

The use of fractal dimension to detect changes in the distribution of the points of a dataset have been already studied in machine learning tasks [14], i.e., clustering [15] and association rules [16], because there are many efficient algorithms for computing it and for its capacity to identify the concept drifts. However, to the best of our knowledge, it has not been applied to the field of classification.

To summarize, the main contributions of this work are:

- We defined an architecture based on two main modules: a fractal dimension-based function for detecting concept drifts in the *non-stationary* phase and an ensemble of GP-based classifiers used in the training (*stationary*) phase.
- Drifts can be detected quickly, while the computationally intensive training phase is executed rarely (i.e., when a change is detected) and for a limited number of windows.
- The transitory phase is handled in a clever way: the algorithm first responds to the drift readjusting the weights, so giving a first quick response, then generates new models for improving the accuracy.
- A fast detection function, based on the fractal dimension, able to quickly detect different types of drifts, is defined.
- The drift detection function is robust to noise and permits to detect different types of drift in high-dimensional datastreams.

1.3. Organization of the paper

The paper is organized as follows: Section 2 reviews some related works; Section 3 provides some background information about distributed computing, GP and ensemble; in Section 4, the different types of drift are analyzed and the strategies for detecting changes based on the fractal dimension are discussed in detail; Section 5 introduces the architecture of the system and the main components for training the classifiers and detecting the changes; Section 6 presents the experiments performed on a real and some artificial datasets; in Section 7, the main advantages and drawbacks of the fractal detection function and the complexity of the re-training phase of our approach are discussed; finally, Section 8 concludes the paper and presents some interesting future developments.

2. Related works

Ensemble-based classification algorithms have been successfully exploited for the task of classifying data streams as an example of incremental systems [17]. In fact, many papers concerning the classification of data streams adopt the ensemble paradigm. This choice is mainly due to the ease with which the ensemble can be updated in order to add/remove classifiers for improving flexibility and maintaining part of the history present in the data. A detailed review and comparison of the main approaches for the ensemble-based classification of data streams can be found in [18], while in [19] the main detection drift techniques are analyzed and compared, by remarking their advantages and drawbacks. As for the evolutionary algorithms, they are largely used for the classification task [20]; however, a few papers [13] also adopts the ensemble paradigm for the data streaming task.

Here, we first review some approaches based on the ensemble paradigm used to cope with data streams, which include a drift detection method and present similarities with our framework, then, we analyze the papers based on evolutionary algorithms and finally, we analyze a few papers adopting the fractal dimension for the task of mining data streams.

In [21], the authors propose a proactive approach specialized for detecting abrupt drifts in large dataset, called DetectA. This technique labels the patterns from the test set using an unsupervised method; then, it compares some statistics computed on the training and on the test set and, on the basis of the result of some multivariate hypothesis tests, a decision is taken concerning the presence or not of a drift. An experimentation conducted on real and artificial datasets, prove the effectiveness of the approach to cope with unbalanced and high-dimensional data, and to detect abrupt drifts. The algorithm has the advantage of detecting abrupt drifts in advance, however it is hardly applicable to real time problems, as the detection function is computationally expensive.

Wang et al. [6] built an ensemble of classifiers maintaining the top classifiers, i.e., the classifiers obtaining the best expected prediction accuracy on the current chunk of data. In addition, an instance-based pruning technique is adopted in order to reduce the number of classifiers necessary to classify the data. Experiments conducted on synthetic and real data sets showed that the ensemble outperforms the single classifier approach in terms of accuracy and the pruning technique reduces the number of classifiers without effecting accuracy. A main drawback of the method is that it continuously updates the model and it does affect the performance of the algorithm in terms of efficiency.

The approach proposed by Chu and Zaniolo [22] boosts fast and light models for classifying data streams. In addition, the algorithm actively detects changes and discards the old ensemble only when a change is raised. In order to detect both abrupt and gradual changes, a two-step statistical based technique is used. Experiments have been conducted on synthetic datasets and on a real life dataset demonstrating the validity of the approach in detecting drifts.

The work described in [23] has the merit of introducing time constraints while collecting labeled data and is also able to detect novel classes. In fact, their realistic model states that the class (new or old) of a tuple appears a time delay after the tuple itself. A new class is detected when a sufficient number of instances are well-separated from the training data and present strong similarities among them. A clustering algorithm is used to detect the candidate "novel class" and a unified measure of cohesion and separation establishes whether a new class is discovered. It is worth mentioning that this technique is orthogonal to our method and it can be included in our algorithm.

The paper [24] describes an interesting approach for classifying data streams, mainly designed to handle the case in which labeled tuples have random frequencies and sizes in the streams. The authors use a random forest algorithm for classifying the stream and a detection function comparing the entropy between the current and a reference window for detecting concept drifts. An interesting criterion is introduced to establish when a forest is ready for deployment, based on the margin function defined by Breiman [25]. The experiments conducted on artificial and real datasets confirm the goodness of the overall algorithm. Differently from our approach, the system updates the forest as soon as new labeled data arrives and not only when a new drift happens; furthermore, its implementation does not exploit the power of distributed architectures. In spite of it being very efficient, it would not be suitable to cope with very fast streams due to these limitations.

Alippi et al. [26] extend a previous system, Just-In-Time (JIT) classifier, with a strategy to cope with recurrent concept drifts. A number of Change-Detection Tests (CDTs) monitor the distribution of input data in order to detect changes. When a drift is detected, a new concept representation is created and compared with the stored representations of concepts and whether it is equivalent to a previously encountered concept its supervised samples are used to reconfigure the classifier. Differently, from our method, this system is particularly apt to handle abrupt shifts, but it is not particularly efficient in handling other kinds of drift. Anyway, this strategy could also be included in our framework to improve its capacity to handle recurrent drifts.

The correlation between the diversity in the ensemble and the concept drift is the main theme of the paper by Minku et al. [27]. The authors study the influence of diversity on the ensemble before and after the drift. The experimental analysis shows that ensembles with less diversity obtain a lower error before the drift, while

a high diversity is required shortly after the drift in order to attain a better accuracy. An important point of the work is that diversity alone is not sufficient to recover from the drift, but new models must be built.

In the second part of this section, we review the papers using evolutionary algorithms for the task of classification of data streams.

The authors of [28] use a genetic algorithm to improve concept drift detection methods, by automatically setting their parameters. Experiments were conducted on nine artificial and three real datasets by using four different concept drift detection techniques. In most of the cases, the accuracy is improved.

The work in [29] aims to verify whether GP can adapt to abrupt concept drifts, without using any drift detection function, on the basis of the consideration that population-based methods can easily adapt to changes. The experiments, performed on synthetic datasets and on one real dataset, show that the rate of crossover and mutation and the size of population affects the speed of adaptation and that a population previously trained on previous data, is more effective in answering to a concept drift.

In [30], the authors define an active learning framework using some policies for sampling and archiving data and adopting GP for classifying the stream. In practice, the GP algorithm is applied only to a sample of the data in order to save time in the computationally expensive phase of the fitness evaluation. The experimental results, conducted on some datasets with class imbalance, demonstrate the goodness of the approach in detecting the minority class.

The problem of evolving GP classifiers for classifying data streams with label budget is coped in [31]. As the above-described work, policies of archiving and sampling are adopted. In addition, a cost is associated to the procedure of labeling data for training the classifiers. The evaluation on artificial and real datasets shows that the framework is sensible to the usage of multiple generations for data subset and to the coevolution of programs.

In [32], the authors adopt an evolutionary approach for the construction of ensembles with different levels of diversity, after a drift is detected and use standard ensemble algorithms in the stationary phase of the algorithm. More in detail, a Kernel Density Estimation (KDE) method is used to generate synthetic datasets, subsequently labeled by means of a multi-objective optimization method, which allows to train each model of the ensemble with a different subset of synthetic samples, in order to promote diversity. Experiments conducted on artificial and real datasets aim to understand for which type or shape of drifts the approach performs better. This method is potentially usable by each ensemble-based algorithm, as it introduces diversity in the ensemble, after a drift is detected. However, the literature is contrasting on the real benefits of promoting diversity in an ensemble.

To summarize, some works are devoted to recognize only a particular type of drift, typically abrupt (as the work in [21], which has the advantage of handling well high-dimensional and imbalanced dataset) or recurrent (as the works in [26,29]), but they cannot efficiently handle other types of drift.

Many works are limited from having an expensive detection function [21], or continuously re-train the model [6], or update the models when new labeled tuples arrive independently from the drifts [22,24]; all these issues limit their applicability to data streams having real-time constraints. Some other works aim to generate an ensemble with a higher diversity (see [27] and [32]), however the improvements in terms of accuracy derived from the introduced diversity in the ensemble are not really evident.

Some strategies are orthogonal to our method, and can be included in our approach to further extend the applicability as [26], which would permit to handle recurrent drifts or [23], which would permit to handle time constraints and detect new classes. In addition, the approach in [30], which aims to understand the performance of a GP-based ensemble when only a sample of the data is used, and the method in [31], which introduces the usage of a label budget, can be also useful to improve our framework.

As mentioned in the previous section, FD has been yet used to tackle different types of data mining tasks (e.g. clustering and mining association rules). Therefore, in this last part of the related work section, we review these papers. In [15], the authors introduce GDFStream (Grid Fractal Dimension-Based Data Stream Clustering), an effective clustering approach for large data streams. The approach incorporates a grid method, and the combined Fractal Dimension-Clustering methodology is employed to efficiently discover the different clusters. The algorithm is composed of an on-line and off-line phase. First, a summary of some statistics are computed and stored (*on-line step*); then, the final clusters are generated using the computed statistics (*off-line step*). The fractal dimension is used to improve the clustering process, so that data points in the same cluster are more self-affine among themselves than to points in other clusters. The approach allows to learn clusters with arbitrary shapes in a very efficient way, and it is capable of dealing with points of high dimensionality.

In [16], the authors define an approach to mine association rules in evolving data streams, allowing the classification of events with rules, which could vary depending on the actual state of the data. The proposed technique aims to identify the most relevant features characterizing each class on the basis of the general properties exhibited by the data stream. The framework combines a statistical rule mine algorithm, named StARMiner (Statistical Association Rule Miner) with a monitoring process, which allows to avoid false alarms, guaranteeing that the set of association rules are updated only when a change in the data occurs. This monitoring technique is based on the computation of the fractal dimension used to estimate change in data streams. Experimental results on synthetic data show that the approach is scalable on both the number of events and the number of attributes.

To the best of our knowledge, GP ensembles and fractal dimension have been applied only in the work in [33] for classifying continuous data streams and handling concept drifts. The main strategy used in the above-cited paper to detect changes is based on the computing of the fractal dimension of the fitness of incoming data. The experimental results are promising, however, even this approach has the limit to need a continuous phase of training and, in addition, computing the fitness of new data requires that at least a significant sample of these data are pre-classified. In real data streams, this can be very costly, as in many cases it would require a work done by experts or a heavy method for computing real classes of the examples. Furthermore, it is hard to decide when changes may happen. In some cases, drifts are quite frequent, while in other cases they can require hours, days or months (i.e., the intrusion detection problem). On the contrary, in our work, changes are detected on the basis of the fractal dimension computed directly on the new chunks of data, without the need to pre-classify data at this phase. So, labeled training data are necessary only when new drifts are discovered, which can be very rare.

3. Background

In this section, we describe some background information useful to fully understand the way in which our classification framework works. First, we illustrate how the ensemble techniques, and in particular, the boosting approach can be used for the task of classification; then, we describe an efficient distributed boosting algorithm proposed in [10] and used as base classification tool in our framework.

3.1. Ensemble approaches for data classification

The ensemble methods [34,35] are learning techniques where several weak base classifiers are trained for the same task by one or more learning algorithms and their predictions are combined according to a specific strategy for classifying new unseen instances. Formally, let $S = \{(x_i, y_i)|i = 1, \ldots, N\}$ be a training set where x_i, called instance, is a features vector with m features and y_i is the class label associated with the instance x_i. A classifier, given a new instance, allows to predict the unknown class label for it.

The base assumption is that the combined decision of many single classifiers is usually more accurate than that given by a single component as shown in [35,36].

Several strategies are defined in literature for combining the classifier predictions.

The boosting algorithms adaptively change the weights of the training instances according to the misclassification error. Given the number T of trials (rounds) to execute, T weighted training sets $S_1, S_2, \ldots, S_T$ are sequentially generated and T classifiers $C^1, \ldots, C^T$ are built to compute a weak hypothesis h_t. Let w_i^t denote the weight of the example x_i at trial t. At the beginning $w_i^1 = 1/n$ for each x_i. At each round $t = 1, \ldots, T$, a weak learner C^t, whose error ϵ^t is bounded to a value strictly less than 1/2, is built and the weights of the next trial are obtained by multiplying the weight of the correctly classified examples by $\beta^t = \epsilon^t/(1-\epsilon^t)$ and renormalizing the weights so that $\Sigma_i w_i^{t+1} = 1$. Therefore, the algorithm aims to assign higher weights to the examples difficult to classify and a lower value to the others.

In addition to the boosting algorithm, stacking-based methods and random forest are among the most used in literature.

The Stacking [37], a.k.a. *stacked generalization*, is an effective ensemble-based approach able to combine the predictions of several classifiers by means a meta-classifier but, differently from the boosting approach, the whole training set is used to learn each single base model. In detail, the algorithm exploits these base models for building a *"stacked-view"* where the features of each instance are the predictions of each classifier on a training set instance. Finally, this view is used to learn the meta-classifier. The main problem of this technique is the need of choosing what kind and how many base classifiers to employ.

The Random Forests [38] is an efficient and particularly accurate ensemble-based technique. This algorithm combines simultaneously two different strategies: a bagging algorithm and a random selection of the features. Specifically, the bagging approach is applied on a set of tree-based classifiers. The main difference with respect to a simple bagging procedure relies on the building of the trees, which are trained and evaluated only on a random subset of the features. The main drawback of this approach is the model size: a forest could require hundreds of megabytes of memory to be applied.

3.2. The distributed GP tool

The GP paradigm is often used to build decision trees in classification problems [39], also exploiting the tree-based representation of this paradigm. In fact, the problem of building a tree by using the GP representation can be easily defined by defining the function set, which includes all the attributes to test and the terminal set including all the classes to predict. Specifically, the function set can be obtained by converting each attribute of the training set into an attribute-test function. Thus, there are as many functions as attributes. Let $A_1, \ldots, A_m$ be the set of m attributes of the training set. Let $Dom(A_i) = \{a_1, \ldots, a_n\}$ be the domain of the possible values that A_i can assume. Then, the corresponding attribute-test function f_{A_i} has arity n, and if the value of A_i is a_j, then $f_{A_i}(a_1, \ldots, a_n) = a_j$. First, when a tuple has to be evaluated, the function contained in the root node of the tree is applied to test the corresponding attribute and consequently to execute the argument that emerges from the test. If the argument is a terminal, then the class name for that tuple is returned, otherwise the new function is executed.

Given a network made up of N nodes,
each having a data set S_j
For j = 1, 2, …, N (for each island in parallel)
  Initialize the weights associated with each tuple
  Initialize the population Q_j with random individuals
end parallel for
For t = 1,2,3, …, T (boosting rounds)
  For j = 1, 2, …, N (for each island in parallel)
    Train cGP on S_j using a weighted fitness according to the weight distribution
    Compute a weak hypothesis
    Exchange the hypotheses among the N islands
    Update the weights
  end parallel for
end for t

Output the hypothesis

Fig. 1. The BoostCGPC algorithm based on AdaBoost.M2.

Indeed, the GP paradigm have been widely applied yet in solving challenging problems coming from several application domains [40–43]. In particular, for the task of classification of data streams, GP-based algorithms are less likely to get stuck in the local optimum, permit to evolve more solutions (and therefore they can be used to build adaptive ensembles better modeling the solution of the problem), and furthermore, they perform a process of implicit feature extraction [13].

In addition, no changes are required in a training set to generate different models for the stochastic nature of the evolutionary algorithms and finally they can be easily implemented on distributed architectures. Therefore, in this work, we use the distributed GP-based algorithm, BoostCGPC, proposed in [10]. It is a GP ensemble method, which performs a hybrid variation of the well-known island model and it demonstrates a superior efficiency and an improved accuracy in comparison with the classical GP algorithm. The pseudo-code of this algorithm is presented in Fig. 1.

In practice, a cGP (cellular Genetic Programming) algorithm improved with the boosting technique AdaBoost.M2 and a population initialized with random individuals weighted according to a uniform distribution are embedded into each node. In detail, the cGP algorithm defines the strategies for selecting and replacing the trees of the population and for the asynchronous migration among the nodes. Each node generates a number of GP classifiers (one for each round of the boosting algorithm) by running for a fixed number of generations and for a given number of boosting rounds. After the end of each boosting round, each classifier updates the local vector of the weights that measures the current prediction accuracy. Finally, for each boosting round, the hypotheses, generated by each classifier, are exchanged among all the processors in order to produce the overall ensemble of predictors. In this way, the whole ensemble is embedded in each node and it can be used for recomputing the new vector of weights.

A detailed description of the BoostCGPC framework can be found in [10].

4. Concept drifts and fractal dimension

In this section, we illustrate the main types of concept drift presented in the literature and show how a function of the family of the fractal dimensions can be used to detect them efficiently.

4.1. Concept drifts and detection function

A concept drift is defined as a change of the distribution generating the data [44,45] and it is usually defined as virtual drift. A different case happens when the conditional distribution of the target variable changes, while the data distribution could not change. Also in this case, we have a type of drift, named real concept drift [46]. Techniques for coping with the real concept drift must be based on the knowledge of the predicted class and, therefore, data including also the class must be available. On the contrary, techniques for handling virtual drift can also act without using any training set with obvious advantages in terms of efficiency.

This work mainly handles drifts that can be detected from the incoming data distribution, so that the drift can be detected as soon as possible and also without having information on the real class of the data, which it is often available later. However, the weighting strategy (see Section 5.4), used to reduce the degradation in the accuracy in the transition phase, can work also in the case of real drifts.

In our paper, we consider the scenario in which we have a fast real data stream and an incremental classification algorithm trying to classify the stream and to answer in real time to changes, which can happen at any time and can be unpredictable. In some cases, the drifts can be recurrent, or in consequence of some event [19]. In other cases, they may happen suddenly/abruptly by switching from one concept to another. Therefore, drifts can be classified on the basis of the speed of the change in terms of gradual or abrupt drifts, or they can be considered recurrent when they are periodic. They are predictable if we can predict when it is likely they occur. In [27], a more detailed description of the most-known types of drift is given.

Designing an efficient detection function for coping with drifts is not an easy task. Among the desired properties of the function, it is worth mentioning the handling of different types of concept drift and the velocity necessary to reduce the delay between the time the drift is detected and the appropriate measures are undertaken. Usually, training data presents an amount of noise, hard to separate; so the detection function must be quite robust to that noise. In addition, typically, the class (label) of the data is not available at the time in which new data arrive and therefore, it would be better that the function was able to work on unlabeled data.

All these considerations brought us to choose the detection function described in the next subsection.

4.2. Exploiting fractal dimension as detection function

Fractals [47] are specific structures that present *self-similarity*, i.e., an invariance with respect to the scale used. A family of functions, named *fractal dimension* (FD) [48], can be usefully adopted to characterize changes in the data. Among the properties of FD, it is worth noticing that the presence of a noise smaller than the signal does not affect it [49].

We can compute the fractal dimension of a dataset by embedding it in a d-dimensional grid with cells of size r and computing the frequency p_i with which data points fall in the ith cell. The fractal dimension is given by the formula $FD = \frac{1}{q-1}\frac{log \sum_i p_i^q}{log\ r}$. Among the fractal dimensions, the *correlation dimension*, obtained when $q = 2$ measures the probability that two points chosen at random will be within a certain distance of each other. Changes in the correlation dimension mean changes in the distribution of the data; thus it can be used as an indicator of concept drift. There are fast algorithms to compute the fractal dimension. We used the *FD*3 algorithm of [50], downloadable from here,[1] that efficiently implements the *box counting method* [51].

[1] https://www.cs.csustan.edu/~john/Post/FD3/.

Owing to these interesting properties, the above-described algorithm is chosen as detection function in our framework. In practice, the fractal dimension function is directly computed on the labeled/unlabeled data coming from the stream and it is applied in an adaptive way to permit the detection of the changes. More details are supplied in Section 5.3.

5. The proposed framework for detecting concept drifts and classifying data streams

In this section, we provide a detailed description of the our framework for the data streams classification task. First, we detail the software architecture and the distributed GP algorithm used to generate the ensemble used to classify the data stream, then we illustrate the phase of the drift detection and discuss the issues concerning the delay in detecting the drift itself.

5.1. The architecture of the framework

In this section, we illustrate the general architecture adopted for classifying data streams, detecting drifts and consequently updating the ensemble.

At the bottom in Fig. 2, an infinite stream of tuples is supposed to flow as input to the framework and only part of them are labeled. After the startup phase, in which the ensemble is built, each newly arrived tuple, i.e., T_i, is passed to the classifier that assigns a class to it. Concurrently, a window of tuples ($T_{i-1}, T_{i-2}, \ldots, T_{i-wind}$, where wind is the number of tuples forming the window) is analyzed by the module computing the fractal dimension in order to generate a new calculation of the fractal dimension Fd_i.

Then, the drift detection module evaluates whether a drift is detected on the basis of the last k values of the fractal dimension computed. In practice, if the drift is detected then the BoostCGPC module is activated, the switch in figure is turned on and a number of labeled tuples are passed to this module. In case the number of labeled tuples is not sufficient as a training set for the classification, the algorithm waits until a sufficient number of tuples is reached. Obviously, this introduces a delay in response to the concept drift.

The base learner of the ensemble is a GP-based decision tree, better detailed in Section 3.2. The ensemble of GP decision trees ($C_1, C_2, \ldots, C_m$) originally built by the BoostCGPC algorithm in the startup phase, is updated, by adding the new trees generated and/or removing old or inadequate trees in accordance with a strategy of pruning. The current implementation simply removes the oldest models. Note that it is one of the possible strategies; for example, a different approach shown in [6] stores each classifier with a class distribution similar to that of the current data and it deserves to be investigated in future works. Anyway, the ensemble-based methods assume that the base models should be quite different in order to obtain a good predictive accuracy, but the correlation between diversity and accuracy is even currently studied and no strong conclusion has been drawn [52].

A more formal description of the overall process can be found in the next subsection.

5.2. The distributed GP classifier algorithm

The classification streaming algorithm alternates two phases: a non-stationary phase in which a function based on the fractal dimension for detecting changes in new unlabeled data (but it can also work on labeled data) is adopted and a (usually not frequent) stationary phase in which the distributed GP algorithm is run in order to build new classifiers and recover from a drift.

The pseudo-code of the algorithm, after an initial phase of training, is shown in Fig. 3. The initial phase simply consists of

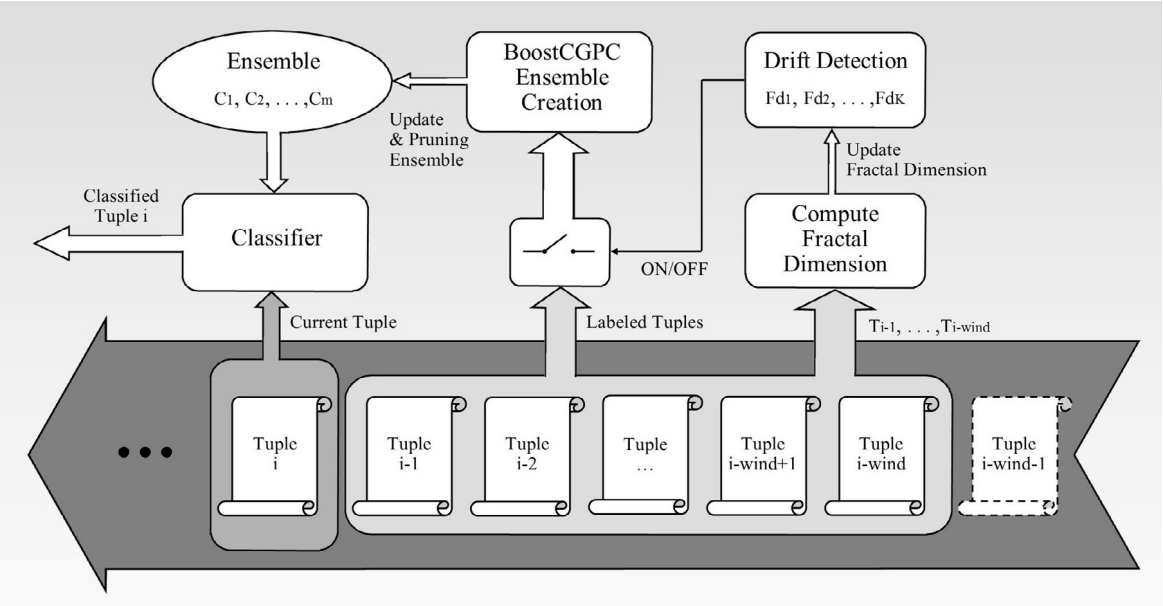


Fig. 2. Software Architecture of the Streaming GP algorithm.

the building of the ensemble by means of BoostCGPC. We consider an infinite stream of data composed by unlabeled tuples $T_1, T_2, \ldots, T_\infty$ and an ensemble $E = \{C_1, C_2, \ldots, C_m\}$ previously built by BoostCGPC. A limit is placed on the size of the ensemble, i.e., the maximum number of classifiers is set to M. In order to detect changes, different windows of data are analyzed. The size of each window is set to T_{wind} and, to avoid the overhead associated with the analysis of each tuple, the detection of the change is detected each T_{incr} tuples.

The fractal dimension computed on each window is added to the set of elements to be analyzed by the detection function and the oldest element is removed in order to maintain the same size. More details on how the fractal dimension is computed and on how the detection function acts are reported in the next subsection.

When a change is detected, a new set of labeled tuples *TL* is produced (i.e., taking the labeled tuples from the stream, if they are present or using an automatic/manual classifier). *BoostCGPC* runs for T rounds on N nodes using this set of tuples and it will produce $T \times N$ classifiers, which will be added to the ensemble removing the oldest classifiers so that the overall number of classifiers remains M. If the size of the ensemble is greater than the maximum fixed size M, the ensemble is pruned by retiring the oldest $(M - T \times N)$ predictors and adding the new generated ones.

5.3. The drift detection module

The drift detection module is based on the fractal dimension function, which is directly computed on the unlabeled data coming from the stream. The overall detection process is illustrated in Fig. 4, and explained in the following.

As explained in the previous subsection, the value of the fractal dimension function (Fd_i) is computed on a window of tuples ($T_{i-1}, T_{i-2}, \ldots, T_{i-wind}$, where wind is the number of tuples forming the window). As the stream goes, we will obtain a fractal set $FS = \{Fd_1, Fd_2, \ldots, Fd_k\}$ where k is the number of elements considered for applying the detection function; the k points will constitute the curve to be analyzed, shown in the box of the figure. Afterwards, the curve is normalized and the linear regression coefficients of the fractal set are computed; if the absolute value of the angle coefficient is greater than a predefined threshold (i.e., 1 corresponding to an angle of $\frac{\pi}{4}$), a change is detected. Choosing

Given a network made up of N nodes to train *BoostCGPC*.
Let T_{wind} and T_{incr} be respectively the size of the window
  examined and the increment considered.
Given $T_1, T_2, \ldots T_\infty$, the unlabeled tuples composing the stream.
Consider M as the maximum number of classifiers forming the
ensemble, $TS = \emptyset$ (a set of tuples to analyze),
the boosting ensemble $E = \{C_1, C_2, \ldots, C_m\}$ where $m < M$,
a fractal set $FS = \{Fd_1, Fd_2, \ldots, Fd_K\}$ where $K = \frac{T_{wind}}{T_{incr}}$
while (more_Tuples)
  $TA = \emptyset$ (new tuples to analyze)
  while ($|TA| < T_{incr}$)
    Add new tuples to TA
  end while
  Add TA to TS and remove the oldest tuples, until
  $|TS| = T_{wind}$ compute the fractal dimension F_d of the set TS
  Add F_d to the set FS and remove the oldest element
  if (detection_function (FS))
    Generate a set $TL = \{Tl_1, Tl_2, \ldots, Tl_k\}$ of labeled tuples
    Train *BoostCGPC* on TL for T rounds of boosting, using N
    nodes and it outputs $T \times N$ classifiers
    Add these classifiers to the ensemble E
  end if
  if ($|\ E\ | > M$)
    prune the ensemble E, removing the oldest $M - |E|$ classifiers
  end if
end while

Fig. 3. The overall algorithm.

angle coefficients greater (minor) permits more (less) abrupt drifts to be coped with, and, in addition, the amount of false positives to be detected can be reduced (increased), at the cost of not detecting some drifts. In other words, the sensibility of the algorithm can be controlled by this parameter.

5.4. Velocity of the stream and delay

A minimum number of labeled tuples, comprising tuples in which the drift was generated, are necessary in order to obtain significant results in the generation of classifiers. That obviously introduces a delay in detecting the drift due to the time necessary to collect this set of tuples [23].

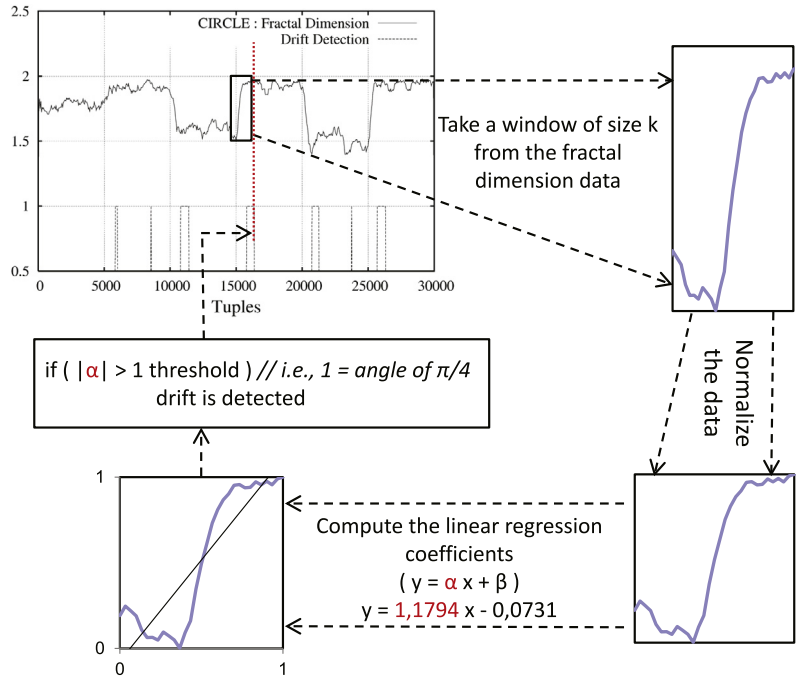


Fig. 4. Detecting a drift on the basis of the fractal dimension.

In this respect, *semi-supervised learning* techniques could be introduced for exploiting the new unlabeled instances provided by the data stream. These techniques are commonly used when a large amount of unlabeled data is available and a limited number of labeled examples is provided [53]. In particular, these approaches try to infer the class of unlabeled instances by means of a model learnt on labeled examples and then using them to enrich the training set and to build a classifier with better generalization capabilities.

Another important point to consider is the delay from when the drift first is detected to when the new classifiers are built and are ready to respond to the drift. This delay mainly depends on two factors: the velocity of the drift, i.e., how many new tuples will arrive in the unit of time, and on the time necessary to built the new classifiers by the evolutionary algorithm. The latter aspect can be reduced by employing a more powerful cluster or more resources in a cloud at the expenses of a higher cost, thanks to the distributed nature of our framework. Thus, in the transitory, it is necessary to accept a trade-off between a degradation in accuracy or the cost of paying a higher price to rent a larger number of resources.

Therefore, we introduced a mechanism to reduce the degradation in accuracy in this transitory phase. In practice, during this phase, the algorithm performs a dynamic tuning of the weights of the classifiers by using a minimum set of new "labeled" coming tuples as a validation set. The number of tuples necessary to adjust the weights is considerably lower than the set needed by the GP algorithm, so that we have a first response to the drift, before the new classifiers are generated. In addition, this strategy has some advantages in comparison to adopting only the pruning of the oldest classifiers. In fact, the historical classifiers will be preserved and those that perform well on the drift will obtain a higher weight and could be also used in the case of recurrent drifts.

Remember that, in the AdaBoost algorithm, the boosting method used in the BoostCGPC algorithm (detailed in Section 3.2), the classifiers composing the final ensemble are weighted by the factor $log(1/\beta^t)$ so that greater weight is given to classifiers with lower error, as β varies from 0 to 1 and is a function of the error (see Section 3.2). In our algorithm, the current ensemble $E = \{C_1, C_2, \ldots, C_m\}$ is validated on a minimum set of labeled tuples $T_{l1}, T_{l2}, \ldots, T_{lk}$ and β is updated using the following formula: $\beta = \beta(1 - \gamma\epsilon)$, where ϵ is the error on the new validation set and γ is a parameter varying from 0 to 1, taking into account the importance of the history of the data (1 indicates no importance to historical data).

In addition, this approach could be exploited to adopt a clever pruning strategy and supplies robustness in case of recurrent drifts.

6. Performance evaluation

In order to validate the goodness of our approach in detecting a drift and quickly responding to it, different experiments using a real and some artificial datasets are conducted. Hence, the main goals of this section are to assess the capacity of the framework in detecting and quickly reacting to the concept drifts, to evaluate the classification accuracy exhibited by the framework and to compare the effectiveness of our approach to recognize the drifts in comparison with other techniques and with the algorithm without the drift recognition part. In the next subsection, more details are supplied concerning the datasets and the parameters used.

6.1. Datasets and parameters

Here, the artificial datasets and the real dataset used in the experiments are described. The artificial datasets were generated using a modified version of the dataset generator freely downloadable,[2] supplied by the authors of the work described in [27]. It is worth noticing that the generator is able to simulate drifts with different grades of class severity.

In addition, a real dataset, hereafter named PhotoObject, was also used. The dataset, a view of the DR6 dataset, extracted from the Sloan Digital Sky Survey (SDSS[3]) is the same used in [24], also for the sake of comparison. This view contains all the attributes (7) of each photometric object, i.e., five attributes representing the original color measurement, and the other two respectively

[2] http://pages.bangor.ac.uk/~mas00a/EPSRC_simulation_framework/changing_environments_stage1a.htm.

[3] http://www.sdss.org/dr6/.

indicating the right ascension (RA) and the declination (DEC). So, the dataset will comprise 500,000 records of class 1 (galaxies), and 41,116 records of class 2 (dwarf-stars). In order to balance the number of tuples of the different classes, as in the above-cited paper, the tuples of the two classes were merged (keeping their original order) by selecting records, from class 1 with probability 0.9 and records from class 2 with probability 0.1. In practice, if the result of the random extraction is 1, the next tuple of class 1 is inserted into the dataset; on the contrary, the next tuple of class 2 is added to the dataset. The resulting dataset contains 782,730 tuples, quite balanced between the two classes.

Differently from the artificial datasets, there is no information on the position and the presence of drifts. However, in the original DSS dataset and similarly also in the PhotoObject view of this dataset, the correct classification of stars and galaxies presents different levels of difficulty and a static model cannot adequately handle all these differences for different reasons. First, dimmer objects close to the galactic plane are obscured by dust and the higher the declination, the fewer dim objects will be obscured, since the angle of capturing the images is moving towards the vertical access of our solar system. In addition, the ratio between the stars and galaxies switches gradually as the declination increases, so that the area that is close to the vertical access of our solar system has less stars and more galaxies. Since the two classes (starts and galaxies) do not have clear boundaries to separate them, changing the ratio of the classes might effect the boundaries between the two classes, and so the classification task becomes more difficult.

The artificial datasets comprise the circle function, the sine function, and the moving hyperplane [7]. The hyperplane generator can also be used to generate lines and planes. All the data sets contain a noise level of 10% obtained by randomly flipping the class of the tuples from 0 to 1 and vice versa, with probability 0.1. Furthermore, the drifts are ordered by increasing class severity (i.e., the percentage of input space having its target changed after the drift).

The circle data set is a 2-dimensional unit hypercube, thus an example x is a vector of 2 features $x_i \in [0, 1]$. The class boundary is a circle of radius r and center c of coordinates (a, b). If an example x is inside the circle then it is labeled class 1, class 0 otherwise. Drift is simulated moving the radius of the circle. We fixed the center to (0.5, 0.5) and varied the radius from 0.2 to 0.3 (16%), from 0.2 to 0.4(38%) and from 0.2 to 0.5(66%). This simulates drifts more and more abrupt (the class severity is reported in brackets).

The sine function is defined by the formula $a \cdot sin(bx + c) + d$ where $a = b = 1, c = 0$ and d varying from -2 to 1(15%), from -5 to 4(45%), from -8 to 7(75%).

The moving hyperplane in d dimensions is defined by the class boundary $\sum_{i=1}^{d} a_i x_i < a_0$ where $a_1 = a_2 = \cdots = a_d = 0.1$ and a_0 varies from -2 to -2.7(14%), from -1 to -3.2(44%) and from -0.7 to -4.4(74%).

All the experiments of the next subsections were averaged over 30 runs and, in order to validate the comparison results statistically, we performed a Wilcoxon signed-ranked test with the confidence level of 0.95 ($\alpha = 0.05$). As for the BoostCGPC classification algorithm, we used the same parameters as in [10] and no tuning phase was conducted as the classification algorithm is quite stable [10] and, in addition, we are not interested in obtaining improvements in terms of accuracy, by choosing the optimal set of parameters for each dataset/application. Therefore, we fixed a probability of 0.1 for reproduction, 0.8 for crossover and 0.1 for mutation. The maximum depth of the new generated sub-trees is 4 for the step of population initialization, and 2 for mutation, while the maximum size after crossover is fixed to 17. During the training phase, BoostCGPC is executed for 5 rounds and for 100 generations for each round, using a population of 1280 individuals. A window of 500 tuples is chosen for the evaluation of the drift, when not differently indicated. In order to evaluate the classification accuracy (Sections 6.3 and 6.6) an interleaved Test-Then-Train evaluation is conducted, i.e., each tuple is used to test the model before it is used for training. Therefore, the model is always being tested on examples it has not seen. In addition, the training phase is conducted only in the initial phase of the algorithm and when a drift is detected.

6.2. Concept drift detection analysis

A first set of experiments aims to analyze the capacity of the detection module to quickly recognize the different types of drift. To this aim, the algorithm is run on the artificial datasets described in the previous subsection. For all the datasets, we generated 30,000 tuples and the concept drifts were simulated each 5000 steps for a total of 5 drifts. We increased the drift from mild to severe every 5000 steps. In practice, at step 5000, the less severe drift is performed, at 10,000 a drift brings the function again to the initial situation, at 15,000 a medium drift is performed and finally, at 25,000 a severe drift is performed. Then, the detection module is applied computing the fractal dimension on a window of 500 tuples with an increment $FD_{incr} = 50$. The drift is detected when the absolute value of the angle coefficient is greater than a threshold fixed to 1, corresponding to an angle of $\frac{\pi}{4}$, which, according to our experiments, permits to handle many types of drift.

In Fig. 5, we show simultaneously the fractal dimension for the analyzed datasets and the identified drifts as vertical lines. The width of the drift is proportional to the number of windows, containing a drift detected by the fractal function. Although the datasets exhibit different behaviors, the proposed function is very effective and allows to identify a drift a few windows after it happens. Moreover, the function maintains good performances even though the presence of noise. Our approach detected only two false positives for the circle and of the sine dataset and only one in the case of the line and of the hyperplane dataset. Nevertheless, they are mainly due to the presence of noise, are very narrow and could be suitably filtered.

The experiments described here only aims at understanding whether our detection function is able to detect the drifts independently of many issues concerning a real stream, such as the minimum number of labeled tuples necessary to train the classifiers and the consequent delay necessary to collect this set of tuples, the delay from when the drift first happens to when the new classifiers are built and ready to respond to the drift, etc. These problems will be analyzed in the experiments of the next subsection.

6.3. Classification accuracy: the effect of delay

As remarked in Section 5.4, the accuracy in classification is limited by the delay in detecting the drift mainly due to two factors: the time necessary to collect the tuples necessary to train the classifiers and the time necessary to the framework in order to build the classifiers. To this aim, in this subsection, we study the effect of the delay after a drift happens and before the new classifiers are trained and used. As the delay is determined by a number of not easily computable factors, we simulate different delays in term of percentage of tuples (5%, 10% 15% and 20%) between two consecutive drifts (5000 tuples for all the artificial datasets and 10,000 tuples for the real dataset).

In Table 1, the error is reported for different percentage of delay (5%, 10%, 15%, 20%).

In order to evaluate whether the differences between the different percentages of delay are significant, a Wilcoxon signed-ranked test with the confidence level of 0.95 ($\alpha = 0.05$) was used. The Wilcoxon test is a nonparametric pairwise test that can

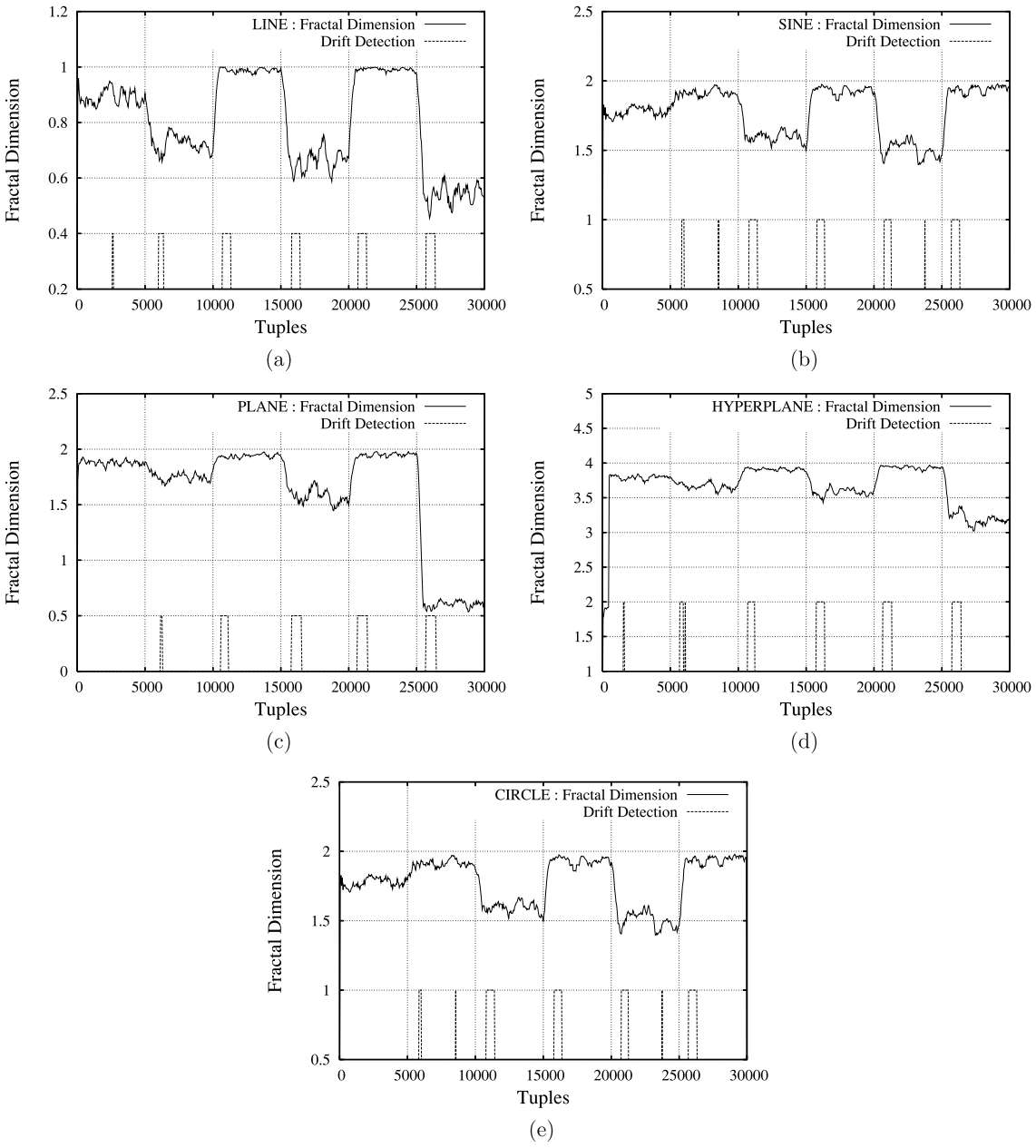


Fig. 5. Fractal Dimension (and Drift Detection) results for (a) Line (b) Sine (c) Plane (d) Hyperplane (e) Circle using 30,000 tuples, presenting a concept drift each 5000 tuples for a total of 6 concept drifts. The width of the drift shown in the figure is proportional to the number of windows in which the detection function identifies the change.

Table 1
Error (%) and standard deviation for the five artificial datasets, for different percentage of delay. The values significantly better of the successive value of delay, are reported in bold.

Dataset	5%	10%	15%	20%
Circle	**13.68 ± 0.20**	**14.22 ± 0.20**	**15.03 ± 0.19**	16.62 ± 0.21
Sine	**12.83 ± 0.25**	**13.81 ± 0.22**	**14.79 ± 0.24**	15.71 ± 0.22
Line	**12.31 ± 0.09**	**13.01 ± 0.10**	**13.58 ± 0.12**	14.09 ± 0.12
Plane	**12.56 ± 0.20**	**13.22 ± 0.19**	**13.81 ± 0.19**	14.30 ± 0.17
Hyperplane	**17.65 ± 0.42**	**17.91 ± 0.40**	18.23 ± 0.38	18.49 ± 0.34

be used to detect significant differences between two algorithms. It is analogous to the paired t-test, however, it is largely used in literature as it does not assume normal distributions and the outliers (exceptionally good/bad performances) have less effect on it. In practice, for each dataset, the test is conducted between two adjacent columns (two different values of delay) and the values, which are significantly better of the successive value of delay, are reported in bold.

In any case, also thanks to the adaptive adjustment of the weights of the classifiers, considering the same dataset, the difference between two consecutive columns (i.e., 5% of difference in term of delay) is not dramatic, even if it is statistically significant in most of the cases. However, passing form a delay of 5% to a delay of 20%, in many cases, the difference is considerable (i.e. for the Sine dataset, we pass from an error of about 13% to an error of about 16%).

6.4. Classification accuracy: the effect of drift detection

In this subsection, we want to analyze the improvement due to the detection of the drifts and the consequent re-training of the classifiers.

Table 2
Error (%) and standard deviation for the five artificial datasets and for the real dataset, detecting drift using the fractal dimension (left) and using only the classification algorithm (right). The values significantly better than the other case are reported in bold.

Dataset	Drift detection function	No drift detection function
Circle	**13.68 ± 0.20**	19.34 ± 0.49
Sine	**12.83 ± 0.25**	18.21 ± 0.45
Line	**12.31 ± 0.09**	18.68 ± 0.15
Plane	**12.56 ± 0.20**	18.94 ± 0.36
HyperPlane	**17.65 ± 0.42**	22.42 ± 0.58
PhotoObject	**0.704 ± 0.034**	0.943 ± 0.043

Table 3
Error (%) and standard deviation for the five artificial datasets and for the real dataset (PhotoObject), for different drift detection strategies (Fractal, ADWIN, STEPD and DDM).

Dataset	Fractal	ADWIN	STEPD	DDM
Circle	**13.382 ± 0.201**	13.783 ± 0.208	**13.258 ± 0.193**	13.822 ± 0.220
Sine	*12.227 ± 0.246*	12.572 ± 0.121	**11.960 ± 0.108**	13.195 ± 0.234
Line	**12.013 ± 0.094**	12.703 ± 0.227	*12.318 ± 0.128*	13.407 ± 0.196
Plane	**12.160 ± 0.197**	12.980 ± 0.091	*12.347 ± 0.139*	13.055 ± 0.116
Hyperplane	**16.052 ± 0.223**	17.068 ± 0.216	*16.665 ± 0.284*	17.042 ± 0.208
PhotoObject	**0.704 ± 0.034**	*0.787 ± 0.044*	0.888 ± 0.067	0.845 ± 0.062

In Table 2, we reported the error and the standard deviation for the two cases (using only the classification algorithms and retraining after a drift is detected). In bold are reported the differences, which are significant on the basis of a Wilcoxon signed-ranked test with the confidence level of 0.95 ($\alpha = 0.05$).

To allow a fair comparison between the two cases, in the table, the drift detection column reports the values considering a delay of 5% in detecting the drift. As it can be observed from the table, for all the datasets, the improvement in accuracy is considerable and it is also meaningful with respect to the above-mentioned statistical test. The differences in accuracy are less evident for the real dataset, in which the drifts are not artificially introduced.

6.5. Classification accuracy: comparing with different drift detection strategies

In order to understand how our fractal-based strategy for detecting the drifts behaves in comparison with other well-recognized techniques, here we compare the accuracy of our strategy in comparison with three drift detection strategies (i.e., ADWIN, DDM and STEPD).

We used the implementations of these techniques included in the MOA tool and the choice of these strategies is based on the work in [54], which analyzes the performance of the most important drift detection strategies.

Statistical Test of Equal Proportions (STEPD) [55] is a drift detection algorithm based on the accuracy. It computes two statistics: the overall accuracy from the beginning of the stream and the accuracy of the model computed on a testing window W. If the difference between the accuracy computed on W and the overall accuracy is greater than a threshold, then a drift is detected and the model must be rebuilt/updated. STEPD uses three parameters: the window value to detect recent changes (we are using 20 instances, as in original paper) and the significance levels α_w and α_d. In practice, STEPD stores the instances in its memory when $P < \alpha_w$ and it resets all the variables (i.e., clear its memory, reset the window accuracy, etc.) when $P < \alpha_d$. As suggested in the original paper, we use a value of 0.03 for α_d and 0.08 for α_w.

The ADaptative WINDdowing method (ADWIN) [56] keeps a sliding window W with the most recent examples and compares the distribution on two sub-windows of W. When the difference of the average value of the two sub-windows is greater than a threshold, then the older sub-window is dropped and a change in the distribution of examples is assigned. A confidence parameter δ with value of 0.002 is used to control the false positive rate, i.e. if the expected value of distribution remains constant within W, the probability that ADWIN shrinks the window at this step is at most δ.

Finally, DDM (drift detection method) [57] is based on a binomial distribution, which represents the probability for the random variable considering the number of errors in a sample of n examples and its empirical standard deviation. As this technique is based on the overall error rate, its effectiveness depends on the changes in the sum of false positive and false negatives and therefore, it has some problems when the changes are gradual. Also for this function, we use standard parameters.

Table 3 reports the percentage of error and the standard deviation for the five artificial datasets and for the real dataset (PhotoObject), respectively for our strategy (Fractal) and for ADWIN, STEPD and DDM. To verify whether the differences are statistically significant we used the Friedman test. The critical value of the Friedman test [58] is obtained from a chi-square distribution with two degree of freedom and a significance level of 5%. In addition, when the test report significant differences, a pair-wise Wilcoxon signed-ranks test with the Bonferroni adjustment was applied to each couple of algorithms in order to rank the algorithms. The algorithm/s performing better is reported in bold and the second one/s in italic.

As for the Circle and for the Line and Plane dataset, STEPD and Fractal dimension are the most effective techniques and no big differences among the two strategies is observed. STEPD outperforms all the other strategies for the Sine dataset, even if differences with our algorithm are not so relevant. Finally, the fractal dimension performs significantly better then the others for the Hyperplane and for the real dataset.

6.6. Comparison with the random forest algorithm

We compared our approach with the random forest-based method described in the related work section, as it is the most related to our approach and uses a real-world large dataset, PhotoObject, described in Section 6.1. We recall that the algorithm adopts a detection function comparing the entropy between the current and a reference window for coping with concept drifts. In addition, it uses a margin function (mg) in order to establish when the forest is ready for deployment, i.e., when the margin function is greater than a defined threshold. Two different threshold definitions are used, one ($mg_{threshold}$) representing a linear relation and the other one ($mg'_{threshold}$) representing a monotonic relation. In practice, when using $mg'_{threshold}$ to decide to deploy the forest the algorithm obtains large classification errors, but fewer unlabeled records are wasted, since the forest is deployed more often than when basing the decision on $mg_{threshold}$.

Fig. 6, taken from [24], shows the results concerning the application of the streaming random forest algorithm to the PhotoObject dataset using the two different thresholds described above (the difference is visible only after the point marked by **F** (record 483,395)). In the figure, the asterisks surrounded by circles and by diamonds respectively represent the points at which the forest has passed through a test phase after it was evaluated and the points of testing for the forest after detecting concept drifts.

The two plots reach a maximum 1.8% classification error. The error reaches its maximum at the point marked by **F**, and decreases until it reaches its minimum towards the end of the stream.

Fig. 8 reports the fractal dimension for the PhotoObject dataset and the identified drifts as vertical lines. The width of the drift is

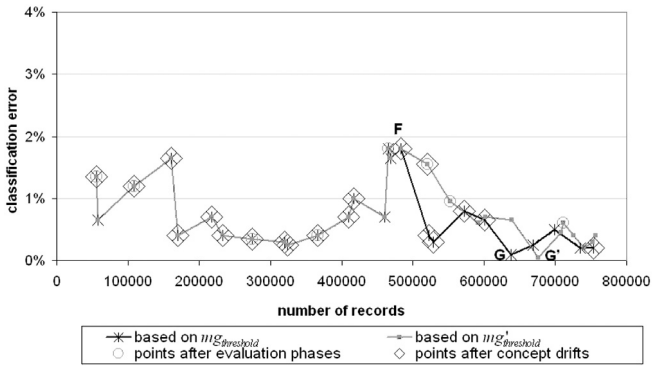


Fig. 6. Classification errors for PhotoObject using the Random Forest algorithm.

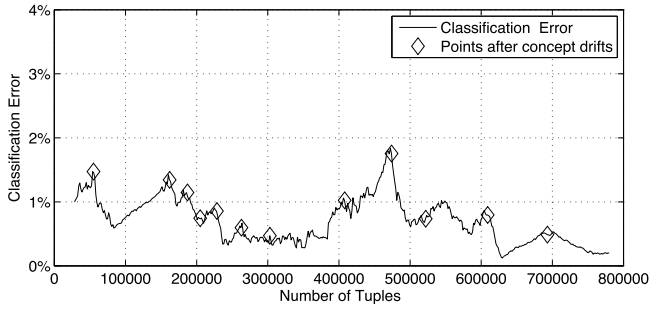


Fig. 7. Classification errors for PhotoObject using our algorithm.

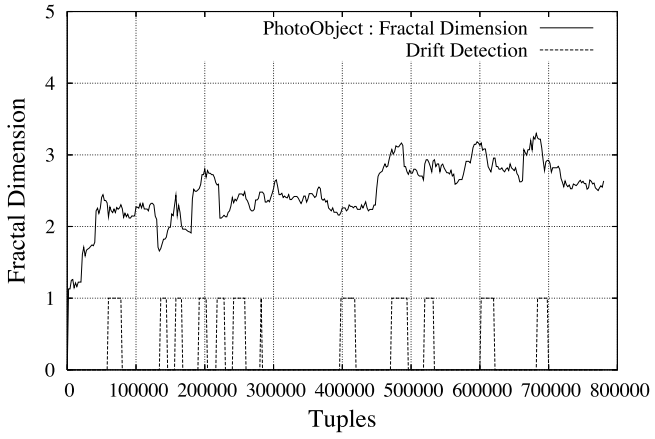


Fig. 8. Fractal Dimension (and Drift Detection) results for the PhotoObject dataset. The width of the drift shown in the figure is proportional to the number of windows in which the detection function identifies the change.

proportional to the number of windows, containing a drift detected by the fractal function.

In Fig. 7, the classification error for our algorithm is shown together with the points (diamonds) in which a concept drift is detected and consequently, it is needed to build new classifiers. The behavior of the two algorithms in terms of classification error is quite similar, except of a region just before 200,000 tuples, in which the random forest algorithm obtains a better accuracy. However, our algorithm (see also Fig. 8) detects only 12 concept drifts (while the random forest needs to recover from 16 concept drifts) and that permits a considerable saving regarding the computationally expensive phase, in which it is necessary to build the classifiers.

7. Discussion

In this section, we want to evaluate the complexity of the re-training phase (the GP classification algorithm) of our approach and to discuss the main advantages and drawbacks of the fractal detection function in comparison with other well-known detection functions, for the task of detecting different types of drift and handling noisy high-dimensional datasets.

Among the main advantages of our algorithm we recall that the detection function can be evaluated quickly and that the computationally expensive re-training phase must be performed only when a drift is detected, which usually is a rare event. However, particularly in the case of frequent drifts, it is important to evaluate the computational cost of this re-training phase, also in comparison with other different approaches. To this aim, on the following we evaluate and compare the time complexity of our method with other two approaches that try to address our same task [24,32]; the first one is based on a population approach as our technique, while the last one adopt a less computationally expensive random forest algorithm.

The GP distributed algorithm used in this work for re-training the ensemble is dominated by the evaluation phase of the evolutionary algorithm while the other operations (crossover, mutation, etc.) are negligible. To summarize the way in which the evolutionary algorithm works, a population of *Pop* classifiers (solutions of the problem) are randomly generated and evolved for a fixed number N of generations, driven by an evolutionary process. In practice, different operations are performed (e.g. crossover, mutation, etc.) in order to generate a new population. In addition, to a classical GP algorithm, our boosted version, evolves the population for K rounds of boosting. At the end of the process, the best solutions form the ensemble used to classify the data. Therefore, the time complexity of our classification method is $O(Pop * N * K)$, as we must evaluate all the members of the population for N generations and for K round of boosting. Typically, a small number of rounds are required to obtain good results (i.e., 5 or 10). Thus, the complexity is dominated by the number of generations and by the dimension of the population. However, the distributed nature of our algorithm would permit to reduce the time by a factor close to m (as GP-based algorithms are highly scalable), where m is the number of processors of a parallel/distributed machine, in which we perform the computation.

The population-based approach proposed in [32] exploits the algorithm NSGA-II (Nondominated Sorting Genetic Algorithm), whose complexity heavily depends on the non-dominated sorting part of the algorithm. The complexity of this operation for each iteration is equal to $O(F*Pop^2)$, where F is the number of objectives and *Pop* is the population size. The number of models in the ensemble K and the number of synthetic samples for training N are two important elements in the complexity analysis. On the contrary, the number of class labels L does not impact on the complexity of the NSGA-II solver per iteration.

In the random forest approach in [24], the training phase depends on three components: *(i)* the time to build each tree, *(ii)* the time to evaluate the forest, and *(i)* the time to test the forest. Assuming the trees are balanced, the time to build a tree is $O(U * tree_{min} * log_2\ tree_{size})$ where U is the number of the trees. The evaluation time depends on reading the records of the evaluation set and passing them down all the trees of the current forest. Anyway, the evaluation set contains a maximum of 1% of the number of records read; therefore the complexity of this step is $O(U * tree_{min} * log_2\ tree_{size})$. The last component requires $O(U * log_2\ tree_{size})$ for evaluating the whole forest. Therefore, the overall complexity of this approach is $O(U * tree_{min} * log_2\ tree_{size})$.

Obviously, with respect the two above mentioned algorithms, while our approach is comparable and also better of the other

population-based approach (if we do not use a large number of generations), it is slower than the random forest algorithm and it could be considerably slower than other approaches based on a single solution. However, by considering that the re-training phase is executed not frequently and that the distributed nature of the framework can alleviate this aspect, nowadays, it can be hardly considered a problem. Indeed, the re-training phase of the experiments performed for this paper for the real world dataset took about 5 s on 16 nodes of a cluster having two Xeon E5520 2.26 GHz CPUs and 16 GB of RAM for node. Consider that this cluster is not so recent (it was acquired at the end of 2009) and, in addition, GPU-based implementations would permit to run GP algorithm up to hundreds of times faster in comparison with running only on CPUs [59], when a sufficient number of evaluation is needed. That would allow to handle really fast streams of data and frequent drifts.

As for the capacity of the drift detection function to detect different types of drifts, in comparison with other approaches it is hard to establish which algorithm is the best, because it depends on the metrics of evaluation considered (among the most significant ones, we recall the evaluation time, the false alarm and the miss detection rate) and on the type and the frequency of the drifts, on the dimensionality of the data, on the presence of noise, etc. According to [3], drift detection methods can be divided in four families, which share common drawbacks and advantages: Hypothesis Tests (assessing the validity of a hypothesis according to a predetermined confidence), Change-point Methods (evaluating when the process changes its statistical behavior or not by analyzing all possible partitions of the data sequence), Sequential Hypothesis Tests (sequentially inspecting incoming samples up to when a decision to accept or refuse the no-change hypothesis can be taken), and Change Detection Tests (sequentially analyzing the statistical behavior of streams of data/features, in order to detect a change).

Hypothesis Tests (as also Change-point methods) works on fixed sequence of data, and therefore, are not particularly apt to work with online datastreams. In addition, change-point methods are usually computationally expensive and hardly can be used for working with fast data streams. Change Detection Tests (CDTs) are usually fast and are apt to works on data streams as they sequentially analyze the statistical behavior of the stream. Sequential Hypothesis Tests can work on sequences of data, but have the problem that, once they take a decision on the drift, no additional data are analyzed and this limits their applicability to a continuous stream of data with frequent changes. Many CDT-based algorithms (i.e., ADWIN, PHT and CUSUM) works on the mean or the standard deviation of the data and therefore, their behavior is more appropriate for characterizing data, which have linear behavior, but they do not behave well when data has nonlinear and chaotic behavior, while fractal dimension can handle also non-linear cases. Other algorithms are based on the error rate (DDM), on the distance-error-rate (EDDM), and directly on the accuracy as STEPD, which compares the accuracy of the base learner in the W most recent instances with the overall accuracy. All these techniques based on the error/accuracy rate, present problems when the changes are gradual or are only present in the features of the data and typically detect a higher number of false alarms. A detailed experimental comparison of the above-mentioned drift detection techniques for different metrics can be found in [54].

As for the drift detection function considered in this paper, i.e., the fractal dimension, it can be evaluated very quickly, is particularly able to filter the noise and to reduce the dimensionality of the data and, typically, it works well even in the case of nonlinear data. In addition, it needs to store in memory only a few values, one for each window of data. In addition, it is less prone to false alarms, as if changes in data are present only for a few windows, they are not considered by our function as a drift. However, our function can hardly recognize really slow drifts, as the fractal dimension curve would have a slope, which the algorithm cannot detect.

8. Conclusions

Classifying large data streams with concept drifts is an important and challenging issue for traditional machine learning approaches both as the drifts can be of different types (abrupt, gradual, etc.) and because the detection phase must be conducted quickly in order to not degradate the accuracy of the model. To cope with this problem, we propose a framework consisting of a distributed GP ensemble classifier used in the training (stationary) phase and a fractal dimension-based function for detecting different types of concept drift in the non-stationary phase. Experiments results show that the framework is able to detect different types of drift quickly, is robust to the noise and has a good behavior when the delay in collecting tuple/recognizing the drift is not excessive. In addition, the fractal-based drift detection strategy proposed is comparable in terms of accuracy with well-recognized drift detection algorithms.

As future works, we plan to study how the distributed nature of the framework permits the choice of a tradeoff between the velocity of the response to a drift and the cost of requesting an adequate number of resources of a Cloud/Distributed environment, i.e., a public cloud environment as the Amazon EC2 platform.

References

[1] Xindong Wu, Xingquan Zhu, Gong-Qing Wu, Wei Ding, Data Mining with Big Data, IEEE Trans. Knowl. Data Eng. 26 (1) (2014) 97–107.
[2] Pethuru Raj, Ganesh Chandra Deka, Handbook of Research on Cloud Infrastructures for Big Data Analytics, first ed., IGI Global, Hershey, PA, USA, 2014.
[3] Gregory Ditzler, Manuel Roveri, Cesare Alippi, Robi Polikar, Learning in non-stationary environments: A survey, IEEE Comp. Int. Mag. 10 (4) (2015) 12–25.
[4] Denis Moreira dos Reis, Peter Flach, Stan Matwin, Gustavo Batista, Fast unsupervised online drift detection using incremental Kolmogorov-Smirnov Test, in: Proceedings of the 22Nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, in: KDD '16, ACM, 2016, pp. 1545–1554.
[5] Dong Liu, Youxi Wu, He Jiang, FP-ELM: An online sequential learning algorithm for dealing with concept drift, Neurocomputing 207 (2016) 322–334.
[6] H. Wang, Wei Fan, P.S. Yu, J. Han, Mining concept-drifting data streams using ensemble classifiers, in: Proceedings of the nineth ACM SIGKDD International conference on Knowledge discovery and data mining (KDD'03), ACM, Washington, DC, USA, 2003, pp. 226–235.
[7] W. Nick Street, YongSeog Kim, A streaming ensemble algorithm (SEA) for large-scale classification, in: Proceedings of the seventh ACM SIGKDD International conference on Knowledge discovery and data mining (KDD'01),, ACM, San Francisco, CA, USA, 2001, pp. 377–382.
[8] J. Zico Kolter, Marcus A. Maloof, Dynamic weighted majority: An ensemble method for drifting concepts, J. Mach. Learn. Res. 8 (2007) 2755–2790.
[9] JinXing Che, YouLong Yang, Li Li, YanYing Li, SuLing Zhu, A modified support vector regression: Integrated selection of training subset and model, Appl. Soft Comput. 53 (2017) 308 – 322.
[10] G. Folino, C. Pizzuti, G. Spezzano, Ensembles for large scale data classification, IEEE Trans. Evol. Comput. 10 (5) (2006) 604–616.
[11] Gianluigi Folino, Giuseppe Papuzzo, Handling different categories of concept drifts in data streams using distributed GP, in: Anna Isabel Esparcia-Alcázar, Anikó Ekárt, Sara Silva, Stephen Dignum, A. Şima Uyar (Eds.), Genetic Programming, in: Lecture Notes in Computer Science, vol. 6021, Springer, 2010, pp. 74–85.
[12] Angeline Wong, Leejay Wu, Phillip B. Gibbons, Christos Faloutsos, Fast estimation of fractal dimension and correlation integral on stream data, Inf. Process. Lett. 93 (2) (2005) 91–97.
[13] Malcolm I. Heywood, Evolutionary model building under streaming data for classification tasks: opportunities and challenges, Genetic Programm. Evolvable Mach. (2014) 1–44.
[14] Daniel Barbará, Chaotic Mining: Knowledge Discovery Using the Fractal Dimension, in: 1999 ACM SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery, 1999.
[15] Guopin Lin, Leisong Chen, A grid and fractal dimension-based data stream clustering algorithm, Inf. Sci. Eng. Int. Symp. 1 (2008) 66–70.
[16] Elaine Parros Machado Sousa, Marcela Xavier Ribeiro, Agma Juci Machado Traina, Caetano Traina Jr., Tracking the Intrinsic Dimension of Evolving Data Streams to Update Association Rules, in: 3rd International Workshop on Knowledge Discovery from Data Streams, part of the 23th International Conference on Machine Learning (ICML06), 2006.

[17] Bartosz Krawczyk, Leandro L. Minku, Joo Gama, Jerzy Stefanowski, Micha Woniak, Ensemble learning for data stream analysis: A survey, Inf. Fusion 37 (C) (2017) 132–156.
[18] Heitor Murilo Gomes, Jean Paul Barddal, Fabrício Enembreck, Albert Bifet, A survey on ensemble learning for data stream classification, ACM Comput. Surv. 50 (2) (2017) 23:1–23:36.
[19] João Gama, Indrė Žliobaitė, Albert Bifet, Mykola Pechenizkiy, Abdelhamid Bouchachia, A survey on concept drift adaptation, ACM Comput. Surv. 46 (4) (2014) 44:1–44:37.
[20] Rodrigo Coelho Barros, Márcio Porto Basgalupp, André Carlos Ponce Leon Ferreira de Carvalho, Alex Alves Freitas, A survey of evolutionary algorithms for decision-tree induction, IEEE Trans. Syst. Man, Cybern. Part C 42 (3) (2012) 291–312.
[21] Tatiana Escovedo, Adriano Koshiyama, Andre Abs da Cruz, Marley Vellasco, DetectA: abrupt concept drift detection in non-stationary environments, Appl. Soft Comput. 62 (2018) 119 – 133.
[22] Fang Chu, Carlo Zaniolo, Fast and light boosting for adaptive mining of data streams, in: Honghua Dai, Ramakrishnan Srikant, Chengqi Zhang (Eds.), Proceedings of the 8th Pacific-Asia Conference (PAKDD 2004), May 26-28, 2004, Proceedings, in: LNAI, 3056, Springer Verlag, Sydney, Australia, 2004, pp. 282–292.
[23] Mohammad M. Masud, Jing Gao, Latifur Khan, Jiawei Han, Bhavani M. Thuraisingham, Classification and novel class detection in concept-drifting data streams under time constraints, IEEE Trans. Knowl. Data Eng. 23 (6) (2011) 859–874.
[24] Hanady Abdulsalam, David B. Skillicorn, Patrick Martin, Classification using streaming random forests, IEEE Trans. Knowl. Data Eng. 23 (1) (2011) 22–36.
[25] Leo Breiman Statistics, Leo Breiman, Random forests, in: Machine Learning, 2001, pp. 5–32.
[26] C. Alippi, G. Boracchi, M. Roveri, Just-in-time classifiers for recurrent concepts, IEEE Trans. Neural Netw. Learn. Syst. 24 (4) (2013) 620–634.
[27] Leandro L. Minku, Allan P. White, Xin Yao, The impact of diversity on online ensemble learning in the presence of concept drift, IEEE Trans. Knowl. Data Eng. 22 (5) (2010) 730–742.
[28] Silas Garrido Teixeira de Carvalho Santos, Roberto Souto Maior de Barros, Paulo Mauricio Gonçalves Júnior, Optimizing the parameters of drift detection methods using a genetic algorithm, in: 27th IEEE International Conference on Tools with Artificial Intelligence, ICTAI 2015, Vietri sul Mare, Italy, November 9-11, 2015, 2015, pp. 1077–1084.
[29] Murray Smith, Vic Ciesielski, Adapting to concept drift with genetic programming for classifying streaming data, in: IEEE Congress on Evolutionary Computation, CEC 2016, Vancouver, BC, Canada, July 24-29, 2016, 2016, pp. 5026–5033.
[30] Sara Khanchi, Malcolm I. Heywood, A. Nur Zincir-Heywood, Properties of a GP active learning framework for streaming data with class imbalance, in: Proceedings of the Genetic and Evolutionary Computation Conference, GECCO 2017, Berlin, Germany, July 15-19, 2017, 2017, pp. 945–952.
[31] Ali Vahdat, Jillian Morgan, Andrew R. McIntyre, Malcolm I. Heywood, A. Nur Zincir-Heywood, Tapped delay lines for GP streaming data classification with label budgets, in: Genetic Programming - 18th European Conference, EuroGP 2015, Copenhagen, Denmark, April 8-10, 2015, Proceedings, 2015, pp. 126–138.
[32] Jesus L. Lobo, Javier Del Ser, Miren Nekane Bilbao, Cristina Perfecto, Sancho Salcedo-Sanz, DRED: An evolutionary diversity generation method for concept drift adaptation in online learning environments, Appl. Soft Comput. 68 (2018) 693 – 709.
[33] Gianluigi Folino, Clara Pizzuti, Giandomenico Spezzano, Mining distributed evolving data streams using fractal GP ensembles, in: EuroGP, 2007, pp. 160–169.
[34] Y. Freund, R. Scapire, Experiments with a new boosting algorithm, in: Proceedings of the 13th Int. Conference on Machine Learning, 1996, pp. 148–156.
[35] Leo Breiman, Bagging predictors, Mach. Learn. 24 (2) (1996) 123–140.
[36] J. Ross Quinlan, Bagging, Boosting, and C4.5, in: Proceedings of the 13th National Conference on Artificial Intelligence AAAI96, Mit Press, 1996, pp. 725–730.
[37] David H. Wolpert, Stacked generalization, Neural Netw. 5 (1992) 241–259.
[38] Tin Kam Ho, The random subspace method for constructing decision forests, IEEE Trans. Pattern Anal. Mach. Intell. 20 (8) (1998) 832–844.
[39] John R. Koza, Genetic Programming - on the Programming of Computers by Means of Natural Selection, in: Complex adaptive systems, MIT Press, 1993, pp. I–XVIII, 1–419.
[40] Dilip P. Ahalpara, Jitendra C. Parikh, Genetic Programming based approach for Modeling Time Series data of real systems, Int. J. Mod. Phys. C 19 (1) (2008) 63–91.
[41] E. Benkhelifa, G. Dragffy, A.G. Pipe, M. Nibouche, Design innovation for real world applications, using evolutionary algorithms, in: Andy Tyrrell (Ed.), 2009 IEEE Congress on Evolutionary Computation, in: IEEE Computational Intelligence Society, IEEE Press, Trondheim, Norway, 2009.
[42] Gregory. S. Hornby, Jason D. Lohn, Derek S. Linden, Computer-automated evolution of an X-band antenna for NASA's space technology 5 mission, Evolutionary Computation 19 (1) (2011) 1–23.
[43] Yi-Shian Lee, Lee-Ing Tong, Forecasting energy consumption using a grey model improved by incorporating genetic programming, Energy Convers. Manage. 52 (1) (2011) 147–152.
[44] Jing Gao, Wei Fan, Jiawei Han, On appropriate assumptions to mine data streams: analysis and practice, in: Proceedings of the 2007 Seventh IEEE International Conference on Data Mining, IEEE Computer Society, Washington, DC, USA, 2007, pp. 143–152.
[45] Gerhard Widmer, Miroslav Kubat, Learning in the presence of concept drift and hidden contexts, Mach. Learn. 23 (1) (1996) 69–101.
[46] T. Ryan Hoens, Robi Polikar, Nitesh V. Chawla, Learning from streaming data with concept drift and imbalance: an overview, Prog. Artif. Intell. 1 (1) (2012) 89–101.
[47] B. Mandelbrot, The Fractal Geometry of Nature, W.H Freeman, New York, 1983.
[48] P. Grassberger, Generalized dimensions of strange attractors, Phys. Lett. 97A (1983) 227–230.
[49] M. Tykierko, Using invariants to change detection in dynamical system with chaos, Physica D 237 (2008) 6–13.
[50] J. Sarraille, Lin S. Myers, FD3: A program for measuring fractal dimension, Educ. Psychol. Meas. 54 (1) (1994) 94–97.
[51] L. Liebovitch, T. Toth, A fast algorithm to determine fractal dimensions by box counting, Phys. Lett. 141A (8) (1989).
[52] Ludmila I. Kuncheva, Diversity in multiple classifier systems, Inf. Fusion 6 (1) (2005) 3–4.
[53] Xiaojin Zhu, Andrew B. Goldberg, Ronald Brachman, Thomas Dietterich, Introduction to Semi-Supervised Learning, Morgan and Claypool Publishers, 2009.
[54] Paulo Mauricio Gonçalves Jr., Silas Garrido Teixeira de Carvalho Santos, Roberto Souto Maior de Barros, Davi Carnauba De Lima Vieira, A comparative study on concept drift detectors, Expert Syst. Appl. 41 (18) (2014) 8144–8156.
[55] Kyosuke Nishida, Koichiro Yamauchi, Detecting concept drift using statistical testing, in: Discovery Science, Springer, 2007, pp. 264–269.
[56] Albert Bifet, Ricard Gavalda, Learning from time-changing data with adaptive windowing, in: SDM, Vol. 7, SIAM, 2007, p. 2007.
[57] Joao Gama, Pedro Medas, Gladys Castillo, Pedro Rodrigues, Learning with drift detection, in: In SBIA Brazilian Symposium on Artificial Intelligence, Springer Verlag, 2004, pp. 286–295.
[58] Janez Demsar, Statistical comparisons of classifiers over multiple data sets, J. Mach. Learn. Res. 7 (2006) 1–30.
[59] Simon Harding, Wolfgang Banzhaf, Fast genetic programming on GPUs, in: EuroGP'07, Springer-Verlag, 2007, pp. 90–101.