

A Survey on Supervised Classification on Data Streams

Vincent Lemaire¹(✉), Christophe Salperwyck², and Alexis Bondu²

¹ Orange Labs, 2 Avenue Pierre Marzin, 22300 Lannion, France
vincent.lemaire@orange.com

² EDF R&D, 1 Avenue du Général de Gaulle, 92140 Clamart, France
{christophe.salperwyck,alexis.bondu}@edf.fr

Abstract. The last ten years were prolific in the statistical learning and data mining field and it is now easy to find learning algorithms which are fast and automatic. Historically a strong hypothesis was that all examples were available or can be loaded into memory so that learning algorithms can use them straight away. But recently new use cases generating lots of data came up as for example: monitoring of telecommunication network, user modeling in dynamic social network, web mining, etc. The volume of data increases rapidly and it is now necessary to use incremental learning algorithms on data streams. This article presents the main approaches of incremental supervised classification available in the literature. It aims to give basic knowledge to a reader novice in this subject.

1 Introduction

Significant improvements have been achieved in the fields of machine learning and data mining during the last decade. A large range of real problems and a large amount of data can be processed [1,2]. However, these steps forward improve the existing approaches while remaining in the same framework: the learning algorithms are still centralized and process a finite dataset which must be stored in the central memory. Nowadays, the amount of 'ble data drastically increases in numerous application areas. The standard machine learning framework becomes limited due to the exponential increase of the available data, which is much faster than the increase of the processing capabilities.

Processing data as a stream constitutes an alternative way of dealing with large amounts of data. New application areas are currently emerging which involve data streams [3,4] such as: (i) the management of the telecommunication networks; (ii) the detection and the tracking of user communities within social networks; (iii) the preventive maintenance of industrial facilities based on sensor measurements, etc. The main challenge is to design new algorithms able to scale on this amount of data. Two main topics of research can be identified: (i) the parallel processing consists in dividing an algorithm into independent queues in order to reduce the computing time; (ii) the incremental processing consists in implementing one-pass algorithms which update the solution after processing

each piece of data. This paper focuses on the second point: incremental learning applied on data streams.

Section 2.1 defines the required concepts and notations for understanding this article. The incremental learning is formally defined in this section and compared with other types of learning. Section 3 presents the main incremental learning algorithms classified by type of classifiers. Algorithms for data streams are then presented in Sect. 4. Such approaches adapt the incremental learning algorithms to the processing of data streams. In this case, the processed data is emitted within a data stream without controlling the frequency and the order of the emitted data. This paradigm allows to process a huge amount of data which could not be stored in the central memory, or even on the hard drive. This Section presents the main on-line learning algorithms classified by type of classifiers. Section 5 addresses the issue of processing non-stationary data streams, also called “concept drift”. Finally, Sect. 6 is dedicated to the evaluation of such approaches: the main criteria and metrics able to compare and evaluate algorithms for data streams are presented.

2 Preamble

2.1 Assumptions and Constraints

In terms of machine learning, several types of assumptions and constraints can be identified which are related to the data and to the type of concept to be learned. This section describes some of these assumptions and constraints in the context of incremental learning in order to understand the approaches that will be presented in Sects. 3 and 4 of this article.

In the remainder of this article, we focus on binary classification problems. The term “example” designates an observation x . The space of all possible examples is denoted by \mathcal{X} . Each example is described by a set of attributes. The examples are assumed to be i.i.d, namely independent and randomly chosen within a probability distribution denoted by $\mathcal{D}_{\mathcal{X}}$. Each example is provided with a label $y \in Y$. The term “model” designates a classifier (f) which aims to predict the label that must be assigned to any example x ($f : \mathcal{X} \rightarrow \mathcal{Y}$).

Availability of Learning Examples: In the context of this article, the models are learned from representative examples coming from a classification problem. In practice the availability and the access to these examples can vary: all in a database, all in memory, partially in memory, one by one in a stream, etc. Several types of algorithms for different types of availability of the examples exist in the literature.

The simplest case is to learn a model from a quantity of representative examples which can be loaded into the central memory and which can be immediately processed. In other cases, the amount of examples is too large to be loaded into the central memory. Thus, specific algorithms must be designed to learn a model in such conditions. A first way of avoiding the exhaustive storage of the examples

into the central memory is to divide the data into small parts (*also called chunks*) which are processed one after the others. In this case, parallelization techniques can be used in order to speed up the learning algorithm [5].

In the worst case, the amount of data is too large to be stored. Data are continuously emitted within a data stream. According to the “data streams” paradigm, the examples can be seen only once and in their order of arrival (*without storage*). In this case, the learning algorithm must be incremental and needs to have a very low latency due to the potentially high emission rate of the data stream.

Availability of the Model: Data mining is a two-step process: (1) learn the model, (2) deploy the model to predict on new data. In the case of regular batch learning, these two steps are carried out one after the other. But in the case of an incremental algorithm, the learning stage is triggered as soon as a new example arrives. In the cases of time-critical applications, the learning step requires a low latency algorithm (*i.e. with a low time complexity*). Others kinds of time constraints can be considered. For instance, in [6] the authors define the concept of “anytime” algorithm which is able to be stopped at any time and provide a prediction. The quality of the outcome model is expected to be better and better, while the algorithm is not stopped. The on-line classifier can be used to make predictions while it is learning.

Concept to be Learned: Let us consider a supervised classification problem where the learning algorithm observes a sequence of examples with labels. The class value of the examples follows a probability distribution denoted by P_Y . The concept to be learned is the joint probability $P(x_i, y_i) = P(x_i)P(y_i|x_i)$, with x_i a given example and y_i a given class value.

The concept to be learned is not always constant over time, sometimes it changes, this phenomenon is called “concept drift” [7]. Gama [8] identifies two categories of drift: either it is progressive and it is called “concept drift”, or it is abrupt and it is called “concept shift”. These two types of drift correspond to a change in the conditional distribution of the target values $P(Y|X)$ over time. The algorithms of the literature can be classified by their ability (or not) to support concept drifts.

The distribution of data $P(X)$ may vary over time without changing $P(Y|X)$, this phenomenon is called covariate shift [9]. The covariate shift also occurs when a selection of non-iid examples is carried such as a selection of artificially balanced training examples (*which are not balanced in the test set*) or in the case of active learning [10]. There is a debate on the notion of covariate shift, the underlying distribution of examples (D_X) can be assumed to be constant and that it is only the distribution of the observed examples which changes over time.

In the remainder of this article, in particular in Sect. 5, we will mainly focus on the drift on $P(Y|X)$, while the distribution $P(X)$ is supposed to be constant within a particular context. The interested reader can refer to [9, 11] to find elements on the covariate shift problem. Gama introduced the notion of “context” [8] defined

as a set of examples on which there is no concept drift. A data stream can be considered as a sequence of contexts. The processing of a stream consists in detecting concept drifts and/or simultaneously working with multiple contexts (see Sect. 5).

Questions to Consider: The paragraphs above indicate the main questions to consider before implementing a classifier:

- Can examples be stored in the central memory?
- What is the availability of the training examples? all available? within a data stream? visible once?
- Is the concept stationary?
- What are the time constraints for the learning algorithm?

The answers to these questions allow one to select the most appropriate algorithms. We must determine whether an incremental algorithm, or an algorithm specifically designed for data streams, is required.

2.2 Different Kinds of Learning Algorithms

This section defines several types of learning algorithms regarding the possible time constraints and the availability of examples.

Batch Mode Learning Algorithms: The batch mode consists in learning a model from a representative dataset which is fully available at the beginning of the learning stage. This kind of algorithm can process relatively small amounts of data (up to several GB). Beyond this limit, multiple access/reading/processing might be required and become prohibitive. It becomes difficult to achieve learning within a few hours or days. This type of algorithm shows its limits when (i) the data cannot be fully loaded into the central memory or continuously arrives as a stream; (ii) the time complexity of the learning algorithm is higher than a quasi-linear complexity. The incremental learning is often a viable alternative to this kind of problem.

Incremental Learning: The incremental learning consists in receiving and integrating new examples without the need to perform a full learning phase from scratch. A learning algorithm is incremental if for any examples x_1, \dots, x_n it is able to generate the hypotheses f_1, \dots, f_n such that f_{i+1} depends only on f_i and x_i the current example. The notion of “current example” can be extended to a summary of the latest processed examples. This summary is exploited by the learning algorithm instead of the original training examples. The incremental algorithms must learn from data much faster than the batch learning algorithms. Such algorithms must have a very low time complexity. To reach this objective, most of the incremental algorithms read the examples just once so that they can efficiently process large amounts of data.

Online Learning: The main difference between the incremental and the online learning is that the examples continuously arrive from a data stream. The classifier is expected to learn a new hypothesis by integrating the new examples with a very low latency. The requirements in terms of time complexity are stronger than for the incremental learning. Ideally, an online classifier implemented on a data stream must have a constant time complexity ($O(1)$). The objective is to learn and predict at least as fast as the data arrives from the stream. These algorithms must be implemented by taking into account that the available RAM is limited and constant over time. Furthermore, concept drift must be managed by these algorithms.

Anytime Learning: An anytime learning algorithm is able to maximize the quality of the learned model (with respect to a particular evaluation criterion) until an interruption (which may be the arrival of a new example). The algorithms by contract are relatively close to the anytime algorithms. In [12], the authors propose an algorithm which takes into account the available resources (time/CPU/RAM).

2.3 Discussion

From the previous points above the reader may understand there are 3 cases:

- the simplest case is to learn a model from a quantity of representative examples which can be loaded into the central memory and which can be immediately processed;
- when the amount of examples is too important to be loaded into the central memory, the first possibility is to parallelize if the data can be split in independent chunks or to have algorithm able to deal the data in one pass;
- the last case, which is dealt in this paper, concerns the data that are continuously emitted. Two subcases have to take into account if the data stream is stationary or not.

3 Incremental Learning

3.1 Introduction

This section focuses on incremental learning approaches. This kind of algorithms must satisfy the following properties:

- read examples just once;
- produce a model similar to the one that would have been generated by a batch algorithm.

Data stream algorithms satisfy additional constraints which are detailed in Sect. 4.1. In the literature, many learning algorithms dedicated to the problem of supervised classification exist such as the support vector machine (SVM), the

neural networks, the k-nearest neighbors (KNN), the decision trees, the logistic regression, etc. The choice of the algorithm highly depends on the task and the need to interpret the model. This article does not describe all the existing approaches since this would take too much place. The following section focuses on the most widely used algorithms in incremental learning.

3.2 Decision Trees

A decision tree [13,14] is a classifier constructed as an acyclic graph. The end of each branch of the tree is a “leaf” which provides the result obtained depending on the decisions taken from the root of the tree to the leaf. Each intermediate node in the tree contains a test on a particular attribute that distributes data in the different sub-trees. During the learning phase of a decision tree, a purity criterion such as the entropy (used in C4.5) or the Gini criterion (used in CART) is exploited to transform a leaf into a node by selecting an attribute and a cut value. The objective is to identify groups of examples as homogeneous as possible with respect to the target variable [15]. During the test phase, a new example to be classified “goes down” within the tree from the root to a single leaf. His path is determined by the values of its attributes. The example is then assigned to the majority class of the leaf, with a score corresponding to the proportion of training examples in the leaf that belong to this class. Decision trees have the following advantages: (i) a good interpretability of the model, (ii) the ability to find the discriminating variables within a large amount of data. The most commonly used algorithms are ID3, C4.5, CART but they are not incremental.

Incremental versions of decision trees have emerged in the 80s. ID4 is proposed in [16] and ID5R in [17]. Both approaches are based on ID3 and propose an incremental learning algorithm. The algorithm ID5R ensures to build a decision tree similar to ID3, while ID4 may not converge or may have a poor prediction in some cases. More recently, Utgoff [18] proposes the algorithm ITI which maintains statistics within each leaf in order to reconstruct the tree with the arrival of new examples. The good interpretability of the decision trees and their ability to predict with a low latency make them a suitable choice for processing large amounts of data. However, decision trees are not adapted to manage concept drift. In this case, significant parts of the decision tree must be pruned and relearned.

3.3 Support Vector Machines

The Support Vector Machines (SVM) has been proposed by Vapnik in 1963. The first publications based on this classification method appeared only in the 90s [19,20]. The main idea is to maximize the distance between the separating hyperplane and the closest training examples.

Incremental versions of the SVM have been proposed, among these, [21] proposes to partition the data and identifies four incremental learning techniques:

- ED - Error Driven: when new examples arrive, misclassified ones are exploited to update the SVM.

- FP - Fixed Partition: a learning stage is independently performed on each partition and the resulting support vectors are aggregated [22].
- EM - Exceeding-Margin: when new examples arrive, those which are located in the margin of the SVM are retained. The SVM is updated when enough examples are collected.
- EM+E - Exceeding-margin+errors: use “ED” and “EM”, the examples which are located in the margin and which are misclassified are exploited to update the SVM.

Reference [23] proposes a Proximal SVM (PSVM) which considers the decision boundary as a space (*i.e. a collection of hyperplanes*). The support vectors and some examples which are located close to the decision boundary are stored. This set of maintained examples change over time, some old examples are removed and others new examples are added which makes the SVM incremental.

The algorithm LASVM [24–26] was proposed recently. This online approach incrementally selects a set of examples from which the SVM is learned. In practice LASVM gives very satisfactory results. This learning algorithm can be interrupted at any time with a finalization step which removes the obsolete support vectors. This step is iteratively carried out during the online learning which allows the algorithm to provide a result close to the optimal solution. The amount of RAM used can be parameterized in order to set the compromise between the amount of used memory and the computing time (*i.e. the latency*). A comparative study of different incremental SVM algorithms is available in [21].

3.4 Rule-Based Systems

The rule-based systems are widely used to store and handle knowledge in order to provide a decision support. Typically, such a system is based on rules which come from a particular field of knowledge (*ex: medicine, chemistry, biology, etc.*) and is used to make inferences or choices. For instance, such a system can help a doctor to find the correct diagnosis based on a set of symptoms. The knowledge (*i.e. the set of rules*) can be provided by a human expert of the domain, or it can be learned from a dataset. In the case of automatically extracted knowledge, some criteria exist which are used to evaluate the quality of the learned rules. The interested readers can refer to [27] which provides a complete review of these criteria.

Several rule-based algorithms were adapted to the incremental learning:

- STAGGER [28] is the first rule-based system able to manage concept drift. This approach is based on two processes: the first one adjusts the weights of the attributes of the rules and the second one is able to add attributes to existing rules.
- FLORA, FLORA3 [29] are based on a temporal windowing and are able to manage a collection of contexts which can be disabled or enabled over time.
- AQ-PM [30] keeps only the examples which are located close to the decision boundaries of the rules. This approach is able to forget the old examples in order to learn new concepts.

3.5 Naive Bayes Classifier

This section focuses on the naive Bayes classifier [31] which assumes that the explanatory variables are independent conditionally to the target variable. This assumption drastically reduces the training time complexity of this approach which remains competitive on numerous application areas. The performance of the naive Bayes classifier depends on the quality of the estimate of the univariate conditional distributions and on an efficient selection of the informative explicative variables.

The main advantage of the approach is its low training and prediction time complexity and its low variance. As shown in [32], the naive Bayes classifier is highly competitive when few learning examples are available. This interesting property contributes to the fact that the naive Bayes classifier is widely used and combined with other learning algorithms, as decision trees in [33] (NBTree). Furthermore, the naive Bayes approach is intrinsically incremental and can be easily updated online. To do so, it is sufficient to update some counts in order to estimate the univariate conditional probabilities. These probabilities are based on an estimate of the densities; this problem can be viewed as the incremental estimation of the conditional densities [34]. Two density estimation approaches are proposed in [35].

IFFD is proposed in [36] which is an incremental discretization approach. This approach allows updating the discretization at the arrival of each new example without computing the discretization from scratch. The conditional probabilities are incrementally maintained in memory. The IFFD algorithm is only based on two possible operations: add a value to an interval, or split an interval. Others non-naive Bayesian approaches exist which are able to aggregate variables and/or values.

3.6 KNN - Lazy Learning

The lazy learning approaches [37] such as the k-nearest neighbors (KNN) do not exploit learning algorithms but only keep a set of useful examples. No global model is built. The predictions are carried out using a local model (built on the fly) which exploits only a subset of the available examples. This kind of approaches can be easily adapted to the incremental learning by updating the set of the stored examples. Some approaches avoid using the old examples during this update [38]. For instance, the KNN approach predicts the label of an example by selecting the closest training examples and by exploiting the class values of these selected examples.

Two main topics of research exists in literature: (i) the speed up of the search of the KNN [39, 40]; (ii) the learning of metrics [41–43]. A comparative study of incremental and non-incremental KNN can be found in [44].

4 Incremental Learning on Streams

4.1 Introduction

Since 2000, the volumetry of data to process has increased exponentially due to the internet and most recently to social networks. A part of these data

arrives continuously and are visible only once: these kinds of data are called data streams. Specific algorithms dedicated to data streams are needed to efficiently build classifiers. This section presents the main methods in the literature for classification on data streams.

First, let us focus on the expected properties for a good algorithm designed for incremental classification on streams. Domingos wrote a general article [45] about learning on data streams and identify additional expected properties of this kind of algorithms:

- process each example in a low and constant time;
- read examples just once and in their order of arrival;
- use of a fixed amount of memory (independently of the number of examples processed);
- able to predict at any time (can be interrupted);
- able to adapt the model in case of concept drift.

Similar properties were presented in [46] but the study was in the context of large data set and not of streaming data. More recently, [47] proposed eight properties to qualify an algorithm for streaming data. However this last article is more related to database field than to data mining. Table 1 summarizes the expected properties in these three papers.

Table 1. Expected properties for a learning algorithm on data streams: (1) = [46]; (2) = [48]; (3) = [47].

	(1)	(2)	(3)
Incremental	x	x	
Read data only once	x	x	x
Memory management	x	x	x
Anytime	x	x	x
Deal with concept drift		x	

Secondly, we focus on the criteria to compare the learning methods. Table 2 presents this comparison with the criteria generally used [49] to evaluate learning algorithm. The deployment complexity is given in this table for a two classes problem and represents the upper bound of the complexity to obtain one of the conditional probability for a test example: $P(C_k|X)$. This upper bound is rarely reached for many classifiers: for instance if we take a decision tree (and the tree is not fully unbalanced), the complexity is $O(h)$ where h represents the depth of the tree.

Finally the differences between incremental learning for batch learning or for data streams are presented in Table 3. A “post-optimization” step can be performed at the end of the learning process. This step aims to improve the model without necessarily reading again the examples.

Table 2. Comparison of the main algorithms: n number of examples; j number of attributes; a number of rules; b average number of predicates per rule, s number of support vectors.

Criterion	Decision tree	SVM	Nearest neighbour	Rules base system	Naive Bayes
Characteristics of the learning algorithm					
Learning speed	+	- -	+	-	+
Deployment complexity	$O(a)$	$O(sj)$	$O(nj)$	$O(ab)$	$O(j)$
Speed and ability to update the model	- -	-	+	+	+
CPU - memory	+	+	- -	+	+
Relevance of the classifier					
Accuracy	+	+	+	+	+
Number of parameters	-	-	+	+	+
Speed to predict	+	-	-	+	+
Model visualization	+	-	+	+	+
Noise sensibility	- -	+	-	-	+

Table 3. Properties of incremental learning vs incremental learning on streams (yes = required, no = not required).

	Incremental	Incremental on streams
Tuning of the learner settings using cross-validation	No	No
Data read just once	Yes	Yes
Post-optimization after learning	Yes	No
Complexity for learning and prediction	Low	Very low
Memory management	Yes	Yes
Handling of the trade-off between accuracy and time to learn	No	Yes
Concept drift handling	No	Yes
Anytime	No	Recommended

The next parts in this section present algorithms which are incremental and also seem to be adapted to process data streams: able to learn without slowing down the stream (lower complexity than the algorithms in the previous section), able to reach a good trade-off between processing/memory/precision, able to deal with concept drifts. They fulfill all or a part of the criteria presented in Table 3.

4.2 Decision Tree

Introduction

Limits of the existing algorithms: SLIQ [50], SPRINT [51], RAINFOREST [52] are algorithms specifically designed to deal with very large data sets. However they require reading many times the data and therefore they do not fulfill the stream constraint of reading only once the data. Few algorithms are able to see data just once as for instance ID5R [17] or ITI [18]. Nevertheless sometimes it is more expensive to update the model than to learn it from scratch, hence these algorithms cannot be considered as “online” or “anytime”.

Tree size: Decision trees are built in a way that makes them always grow as new data arrive [53] but a deeper tree usually means over-fitting than better performance ([49] - p.203). In the context of stream mining, a tree would never stop to grow if there are no mechanisms to prune it. Therefore new algorithms were proposed to build trees on a data stream that will have dedicated mechanisms to stop them from growing.

Concept drift: Data streams usually have concept drift and in that case the decision tree must be pruned or restructured to adapt to the new concept [17, 18].

Hoeffding bound: Numerous incremental learning algorithms on streams use the Hoeffding bound [54] to find the minimal number of examples needed to split a leaf into a node. For a random variable r within the range R and after n independent observations, the Hoeffding bound states that, with a probability $1 - \delta$, that the true mean of a r is not different from $\bar{r} - \epsilon$, where $\epsilon = \sqrt{\frac{R^2}{2n} \ln(\frac{1}{\delta})}$. This bound does not depend on the data distribution but only on:

- the range R
- the number of observations n
- the desired confidence δ

As a consequence, this bound is more conservative since it has no prior on the data distribution¹.

Main Algorithms in the Literature: This section presents the main incremental decision trees for data streams in the literature:

VFDT [56] is considered as a reference article for learning on data streams with millions/billions of examples. This article is widely referenced and compared to new approaches on the same problems since 2000. In VFDT, the tree is built incrementally and no examples are kept. The error rate is higher in the early stage of the learning than an algorithm as C4.5. However after processing millions

¹ This bound is not well used in many algorithms of incremental trees as explain in [55] but with not a very big influence on the results.

of examples, the error rate become lower than C4.5 which is not able to deal with millions of examples but has to use just a subset of them. Figure 1, taken from VFDT article, shows this behavior and why it is better to use VFDT in the case of massive data streams. Moreover Domingos and Hulten proved that the “Hoeffding Tree” is similar to the tree that would have been learned with an off-line algorithm. In order to suit better the use case of stream mining VFDT can be tuned. The two main parameters are: (i) the maximum amount of memory to use, (ii) the minimal number of examples seen before calculating the criterion.

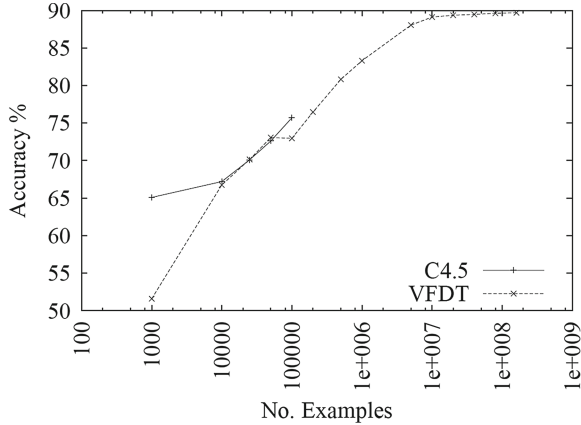


Fig. 1. Comparison of C4.5 with VFDT (taken from [56])

CVFDT [48] is an extension of VFDT to deal with concept drift. CVFDT grows an alternative subtree whenever an old one seems to be out-of-date, and replaces the old subtree when the new one becomes more accurate. This allows one to make adjustments when concept drift occurs. To limit the memory usage only the promising sub-trees are kept. On the dataset “rotating hyperplane” (see Sect. 6.3), the experiments show that the tree contains four times less nodes but the time spent to build it is five times longer comparatively to VFDT.

VFDTc [57] is an extension of VFDT able to deal with numerical attributes and not only categorical. In each leaf and for each attribute counters for numerical values are kept. These counters are stored using a binary tree so that it is efficient to find the best split value for each attribute to transform this leaf into a node. The authors of VFDTc observe that it needs 100 to 1,000 examples to transform a leaf into a node. In order to improve the performance of the tree, they propose to add a local model in the leaves. The naive Bayes classifier is known to have good performance with few data (Sect. 3.5) and therefore this classifier is used in the leaves of VFDTc. Local models improve the performance of the tree without using extra amount of memory since all the needed statistics are already available.

In **IADEM** [58] and **IADEMc** [59] the tree construction is also based on the Hoeffding bound. The tree growth is stopped using the error rate given as a parameter to the algorithm. **IADEMc** is an extension able to handle numerical attributes and which have a naive Bayes classifier in its leaves (as **VFDtc**). Their experiments on the WaveFrom and LED datasets show the following differences: a slightly lower accuracy but less deep trees.

Finally Kirkby in [60] makes a study on “Hoeffding Tree” and proposes improvements. His first improvement concerns leaves that could have either a naive Bayes or a majority class classifier. His experiments show that having a local classifier is not always the best option. He proposes to use the classifier which has the lowest error rate in its leaf. His second improvement is “Hoeffding Option Trees” (related to [61]) which have few sub-trees in their nodes. The same learning examples can update many trees at the same time. The prediction is a majority vote to predict the class value. Bagging [62] and boosting [63] techniques are also studied: bagging surely improves the accuracy but boosting does not always give a positive result.

Discussion

Limits of decision trees: Because of the tree structure its first node is very important. If a change occurs on the attribute on top nodes then it is more complicated to update the tree. In that case, it might be needed to rebuild the whole tree from scratch as stated in [64]. In that case, using decision trees for classification on non-stationary streams can be costly. The ability for an incremental tree to get restructured is for sure a key point to have good performance.

About the “anytime” aspect, [65] proposed to use a tree containing a Bayesian model in all the nodes. If there is not enough time to reach a leaf, the model in the last visited node is used.

About the Hoeffding bound: one could mention that other bounds like the McDiarmid bound exist and could be used as well. In [66] Rutkowski et al. show that the Hoeffding’s inequality is not totally appropriate to solve the underlying problem. They prove two theorems presenting the McDiarmid’s bound for both the information gain, used in ID3 algorithm, and for Gini index, used in Classification and Regression Trees (CART) algorithm.

4.3 Support Vector Machine (SVM)

In [67], a SVM version to learn on large datasets is presented. This version is based on an approximation of the optimal solution using “MEB - Minimum Enclosing Balls” resolution. Its spatial complexity is independent of the learning dataset size. A parameter tunes the trade-off between speed and precision of the classifier.

Reference [68] presents an optimization of the existing methods in order to increase their throughput. This optimization use the “divide and conquer” technique to parallelize the problem into sub problems. The goal is to rapidly find and exclude examples which are not support vectors.

We could also use LASVM [24–26] on each example of the stream: either the example is close to the actual support vector and the current solution is updated; or it is far and then the example is ignored. If the example is used, the complexity is in the best case $O(s^2)$ with s the number of support vectors. In both cases (example ignored or not), the elements from the kernel need to be computed. This pretty high complexity can explain the very few use cases of “LASVM” in data stream mining (compared to Hoeffding trees). Reference [69] presents guarantees for “LASVM” in the context of online learning. To improve the training time to build the SVM, [70] proposed to use linear SVM and parallelize computation using GPUs to reduce the computation of a factor 100.

4.4 Rule-Based System

Few articles proposed rule-based systems on data streams. Incremental rule-based algorithms exists (Sect. 3.4) but they are unable to deal with data streams. Nevertheless three articles of Ferrer et al. [71, 72] proposed the FACIL algorithm able to deal with data streams. FACIL is based on the rule-based system AQ-PM [30].

The FACIL algorithm works in the following way:

- when a new example arrives, it looks for all the rules covering and having its label, then it increments the rule positive support.
- if the example is not covered then it looks for the rule which needs the minimal increase of its coverage space. Moreover this increase has to be limited to a given value κ to be accepted.
- if no rule with the same label as the example are found, then it looks into the whole set of rules with a different label. If a rule is found then it adds this example as a negative one and increments the rule negative support.
- if no rule covers this example, then a new rule can be created.
- each rule has its own forgetting window. This window has a variable size.

More recently Gama and Kosina [73] proposed a new algorithm based on the Hoeffding bound to build rules. A rule is expanded if, for an attribute, the gain to split on an attribute is better than not to split (the notion of gain is the same as in VFDT). Moreover another condition is added: the value of the chosen attribute has to be seen in more than 10 % of the examples. Similarly to VFDTc, they add a naive Bayes classifier in each rule to improve their accuracy.

4.5 Naive Bayes

The naive Bayes classifier uses conditional probabilities estimates (Sect. 3.5). These estimates are usually done after a discretization of the explicative variables. In the stream mining context this first step needs dedicated methods as most off-line methods usually need to load all the data in memory. In the literature, two kinds of publication related to incremental discretization can be found. The articles related to data-mining are not numerous but the literature related to database provides much more articles because the database systems (DBMS) need to have good estimates of the data distribution.

Discretization for Data Mining: Gama and Pinto in [74], propose PiD: “Partition Incremental Discretization” which performs an incremental discretization in order to build a naive Bayes classifier for streams. This discretization is based on two layers:

- first layer: it aims to lower the spatial and processor complexity of the streams. It stores per interval the number of examples per class. These intervals evolve with the stream using the following algorithms: if new values arrive and are outside the range of data already arrived then a new interval is inserted (\sim EqualWidth); if an interval contains too many examples then this interval is split into two intervals (\sim EqualFreq). The update is incrementally done. This first layer needs to contain much more intervals than the second layer. The number of intervals in the first layer is one of the parameters of the algorithm.
- second layer: it aims to have a more qualitative discretization to build a good model. It takes as an input the discretization of the first layer and builds the final discretization which can be: EqualWidth, EqualFreq, K-means, Proportional discretization, Recursive entropy discretization, MDL.

One issue that could arise with this method is due to the first layer that can use more memory than expected. This could happen if the data distribution is much skewed or in the presence of numerous outliers. In that case lots of new intervals will be created.

Discretization for DBMS: Database systems (DBMS) also used discretization for estimating data distribution. These estimations are used to have statistics on the data in order to build accurate execution plans and know which optimization/join to use. For DBMS it is necessary that these algorithms are able to deal with insertion and deletion but for data-mining on data streams this is not always mandatory.

Gibbons et al. [75] propose new incremental algorithms to update the histograms for database systems. Their approach is based on two structures:

- a reservoir to store representative examples (based on “Reservoir sampling” [76]);
- “Equal Frequency” histograms to summarize the data.

Weighted Naive Bayes: After the computation of the conditional densities ($\log(P(X|C))$) a weight on each explanatory variable could be computed online as in [77]. These kinds of method allow the naive bayes to reach better results.

4.6 KNN - Lazy Learning

Nearest neighbors algorithms are easily incremental (Sect. 3.6). In the context of stream meaning, most of approaches focus on keeping the most representative examples so that the memory consumption does not increase. This can

be achieved by (i) forgetting the oldest examples, (ii) aggregating the nearest neighbors.

Different methods based on the nearest neighbors are proposed in the literature for stream mining:

- [78] used a discretization technique to lower the input space and therefore limit the memory needed to store the examples.
- [79] keeps the most recent examples and used a forgetting window to update them.
- [80] considers that the key problem for learning on data stream is to maintain an implicit concept description in the form of a case base (memory).

4.7 Other Methods

The main approaches in the literature for incremental classification presented above do not cover all the literature of online learning. A general scheme for others classifiers could be found in [81]. We give below a list of other online classifiers (which do not necessarily have all the properties given in Table 1):

Examples inspired on The Immortal Perceptron [82, 83]: Second-Order Perceptron [84], Ultraconservative Algorithms [85], Passive-Aggressive Learning [86].

Examples based on Stochastic Optimization: Pegasos [87], Stochastic Gradient Descent [88].

Examples based on Recursive Least Squares/Optimal Control: Online Kernel Recursive Least Squares [89], Sparse Online Gaussian Processes [90].

Example based on Bandits²: Thompson Sampling [91], UCB, UCT and variants [92], Exp3, and variants [93].

Examples (Online Learning with Kernels) dedicated to deal with concept drift (see Section 5): Kernel Perceptron [94], Passive-Aggressive Learning [86] (implemented in Mahout), Pegasos [87], Budget online learning (Budget Perceptron [95], Forgetron [96], Projectron [97]).

² Multi-armed bandits explore and exploit online set of decisions, while minimizing the cumulated regret between the chosen decisions and the optimal decision. Originally, multi-armed bandits have been used in pharmacology to choose the best drug while minimizing the number of tests. Today, they tend to replace A/B testing for web site optimization (Google analytics), they are used for ad-serving optimization. They are well designed when the true class to predict is not known: for instance, in some domains the learning algorithm receives only partial feedback upon its prediction, i.e. a single bit of right-or-wrong, rather than the true label.

5 Concept Drift Management

Concept drift consists of changes in statistical properties of incoming examples. It causes a decrease in classifiers accuracy as time passes (non stationary data stream).

In supervised classification, the concept to learn $P(C|X)$ is the conditional probability of the class of C knowing the data observations X . A data stream could be non-stationary if the process which generates the data evolves over the time. In this case the classifier has to adapt gradually as the concept changes.

The literature on concept drift detection or concept drift management is abundant [98,99]. The methods on drift management can be split into several groups: drift detection, ensemble of classifiers, samples weighting, etc.

In this section we consider that the concept to learn may change over time but it is persistent and consistent (see [100]) between two changes. The period which exists between two changes is named context (see Sect. 2.1 [8]). Concept drift appears through the examples: the old observations are obsolete when considering the actual process which generates the actual observations. The old observations do not belong to the actual context.

We assume that between two context changes there exists a concept sufficiently persistent and consistent to be able to collect training examples. In that case concept drift management is mainly based on concept drift detection.

If a concept drift detection method is available, then we could:

- either retrain the decision model from scratch;
- or adapt the current decision model;
- or adapt a summary of the data stream on which the model is based;
- or work on a sequence of decision models which are learned over the time.

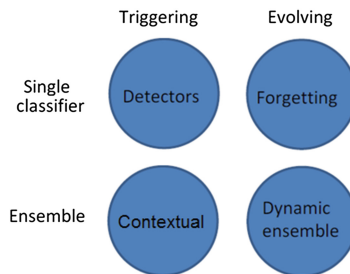


Fig. 2. Adaptive learning strategies [101].

In [101] another taxonomy is given by considering two types of method: (i) Triggering/Evolving and (ii) Single classifier/Ensemble of classifiers (see Fig. 2). In this figure, in case of:

- Single classifier + Triggering: detect a change and cut;
- Single classifier + Evolving: forget old data and retrain;

- Ensemble of classifiers + Triggering: build many models, switch models according to the observed incoming data;
- Ensemble of classifiers + Evolving: build many models, dynamically combine them.

The literature indicates that concept drift may have different faces. Assuming that the concept of a sequence of instances changes from S_i to S_j , we call a drift a sudden drift, if at time t , S_i is suddenly replaced by S_j . Instead of a sudden change which means that the source is switched during a very short period of time after the change point t , gradual drift refers to a change with a period both sources S_i and S_j are active. As time passes, the probability of sampling from source S_i decreases, probability of sampling from source S_j increases until S_i is completely replaced by S_j . There is another type of gradual drift called incremental or stepwise drift. With more than two sources, the difference between the sources is very small, thus the drift is noticed only when looking at a longer time period. Reoccurring context means that previously active concept reappears after some time. It is not certainly periodic. So there is not any predictive information on when the source will reappear. In Fig. 3, we can see how these changes happen in a data stream.

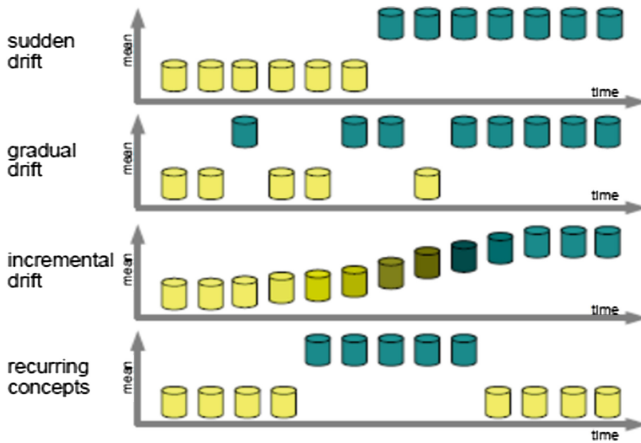


Fig. 3. Types of concept drift [99]

The goal of the supervised classification on data streams is to keep the performance of the classifier over the time, while the concept to learn is changing. In this case, the literature contains three families:

- methods without a classifier: authors mainly use the distributions of the explanatory variables of the data stream: $P(X)$, $P(C)$, $P(X|C)$.
- methods using a classifier: authors mainly use the performance of the classifier related to the estimation of $P(C|X)$.
- adaptive methods with or without using a classifier.

From a Bayesian viewpoint, all these methods try to detect the variation of a part of:

$$P(C|X) = P(C)P(X|C)/P(X) \quad (1)$$

with

1. $P(C)$ the priors of the classes in the data stream
2. $P(X)$ the probability (distribution) of the data
3. $P(X|C)$ the conditional distribution of the data X knowing the class C .

The following section is organized in two parts: the first part describes methods to perform concept drift detection; the second part describes methods to adapt the model (single classifier or ensemble of classifiers). In this section we do not consider the novelty detection problem [102]. Novelty detection is a useful ability for learning systems, especially in data stream scenarios, where new concepts can appear, known concepts can disappear and concepts can evolve over time. There are several studies in the literature investigating the use of machine learning classification techniques for novelty detection in data streams. However, there is no consensus regarding how to evaluate the performance of these techniques, particular for multiclass problems. The reader may be interested by a new evaluation approach for multiclass data streams novelty detection problems described in [103].

5.1 Concept Drift Detection

An extensive literature on concept drift exists. The interested reader may find in [99] an overview of the existing methods. In the following subsections we present only a part of the literature related to drift detection methods applied on one of these three terms: $P(C)$, $P(X)$ and $P(X|C)$.

Concept drift detection is mainly realized by (i) monitoring the distributions of the explanatory variables of the data stream: $P(X)$, $P(C)$, $P(X|C)$; (ii) monitoring the performance of the classifier related to the estimation of $P(C|X)$.

Methods Using a Classifier: Methods using a classifier monitor the performance of the classifier and detect a concept drift when the performance varies significantly. These methods assume that the classifier is a stationary process and data are independent and identically distributed (iid). Though, these hypothesis are not valid in case of a data stream [55], these methods proved their interest on diverse application [98, 104, 105]. The three most popular methods using a classifier of the literature are described below.

- **Widmer and Kubat** [29] proposed to detect the change in the concept by analyzing the misclassification rates and the changes that occur in the structures of the learning algorithm (adding new definitions in the rules in his case). From some variation of these indicators, the size of the training window is reduced otherwise this window grows in order to have more examples to achieve better learning. This article was one of the first works on

this subject. Many other authors, like Gama in the paragraph below, based their methods on it.

- **DDM** This method of Gama et al. [104], detects a concept drift by monitoring the classifier accuracy. Their algorithm assumes that the binary variable which indicates that the classifier has correctly classified the last example follows a binomial distribution law. This law can be approximated as a normal law when the number of examples is higher than 30. The method estimates after the observation of each sample of the data stream the probability of misclassification p_i (p_i corresponds also to the error rate) and the corresponding standard deviation $s_i = \sqrt{p_i(1-p_i)/i}$. A significant increase of the error rate is considered by the method as the presence of a concept drift.

The method uses two decision levels: a “warning” level when $p_i + s_i \geq p_{min} + 2 \cdot s_{min}$ and a “detection level” when $p_i + s_i \geq p_{min} + 3 \cdot s_{min}$ (after the last detection, every time a new example i is processed p_{min} and s_{min} are updated simultaneously according to $(p_{min} + s_{min}) = \min_i(p_i + s_i)$). The time spent between the warning level and the detection level is used to train a new classifier which will replace the current classifier if the detection level is reached. This mechanism allows not starting from scratch when the decision level is reached (if the concept drift is not too sudden).

- **EDDM** This method [105] uses the same algorithm but with another criterion to set the warning and detection levels. This method uses the distance between the classification errors rather the error rate. This distance corresponds to the number of right predictions between two wrong predictions. EDDM computes the mean distance between the errors p'_i and the corresponding standard deviation s'_i . As for DDM a warning level and a detection level are defined, respectively of $(p'_i + 2 \cdot s'_i)/(p'_{max} + 2 \cdot s'_{max}) < \alpha$ and $(p'_i + 2 \cdot s'_i)/(p'_{max} + 2 \cdot s'_{max}) < \beta$. In the experimental part the authors of EDDM use $\alpha = 90\%$ and $\beta = 95\%$. On synthetic datasets EDDM detects faster than DDM the gradual concept drift.

Methods Without a Classifier: The methods without a classifier are mainly methods based on statistical tests applied to detect a change between two observation windows. These methods are often parametric and need to know the kind of change to detect: change in the variance, change in the quantiles, change in the mean, etc. A parametric method is a priori the best when the change type exactly corresponds to its setting but might not perform well when the problem does not fit its parameters. In general, it is a compromise between the particular bias and the effectiveness of the method. If one seeks to detect any type of change it seems difficult to find a parametric method. The list below is not exhaustive but gives different ideas on how to perform the detection.

- **Welch’s t test:** this test applies on two samples of data of size N_1 and N_2 and is an adaptation of the Student’s t test. This test is used to statistically

test the null hypothesis that the means of two population \bar{X}_1 and \bar{X}_2 , with unequal variances (s_1^2 and s_2^2), are equals. The formula of this test is:

$$p\text{-value} = (\bar{X}_1 - \bar{X}_2) / \left(\sqrt{(s_1^2/N_1) + (s_2^2/N_2)} \right) \quad (2)$$

The null hypothesis can be rejected depending on the *p-value*.

- **Kolmogorov-Smirnov’s test:** this test is often used to determine if a sample follow (or not) a given law or if two samples follow the same law. This test is based on the properties of their empirical cumulative distribution function. We will use this test to check if two samples follow the same law. Given two samples of size N_1 and N_2 having respectively cumulative distribution function $F_1(x)$ and $F_2(x)$, the Kolmogorov-Smirnov distance is defined as: $D = \max_x |F_1(x) - F_2(x)|$. The null hypothesis, assuming that the two samples follow the same law, is rejected with a confidence of α if: $\left(\sqrt{(N_1 N_2) / (N_1 + N_2)} \right) D > K_\alpha$. K_α can be found in the Kolmogorov-Smirnov table.
- **Page-Hinkley test:** Gama in [106] suggests to use the Page-Hinkley test [107]. Gama modifies this test using fading factor. The Page-Hinkley test (PHT) is a sequential analysis technique typically used for monitoring change detection. It allows efficient detection of changes in the normal behavior of a process which is established by a model. The PHT was designed to detect a change in the average of a Gaussian signal [108]. This test considers a cumulative variable defined as the cumulated difference between the observed values and their mean till the current moment.
- **MODL $P(W|X_i)$** this method has been proposed by [109] to detect the change while observing a numerical variable X_i in an unsupervised setting. This method addresses this problem as a binary classification problem. The method uses two windows to detect the change: a reference window W_{ref} which contains the distribution of X_i at the beginning and a current (sliding or jumping) window W_{cur} which contains the current distribution. The examples belonging to W_{ref} are labeled with the class “*ref*” and the ones belonging to W_{cur} are labeled with the class “*cur*”. This two labels constitute the target variable $W \in \{W_{ref}, W_{cur}\}$ of the classification problem. All these examples are merged into a training set and the supervised MODL discretization [110] is applied to see if the variable X_i could be split in more than a single interval. If the discretization gives at least two intervals then there are at least two significantly different distributions for X_i conditionally to the window W . In this case the method detects that there is a change between W_{ref} and W_{cur} .

5.2 Adaptive Methods

The online methods can be divided into two categories [111] depending on whether or not they use an explicit concept drift detector in the process of adaptation to evolving environments.

In case of explicit detection, after a concept drift was detected we have to:

- (i) either retrain the model from scratch; (ii) or adapt the current decision model;

(iii) or adapt a summary of the data stream on which the model is based; (iv) or work on a sequence of models which are learned over the time.

If we decide to retrain the classifier from scratch the learning algorithm uses:

- either a partial memory of the examples: (i) a finite number of examples; (ii) an adaptive number of examples; (iii) a summary of the examples. The size of the window of the last used examples cannot exceed the last detected concept drift [112].
- or statistics on all the examples under the constraint of memory and/or the precision on the statistics computed [113]. In this case, the idea is to keep only in memory the “statistics” is useful to train the classifier.

In case of implicit adaptation, the learning system adapts implicitly to a concept change by holding an ensemble of decision models. Two cases exist:

- either the predictions are based on all the classifiers of the ensemble using simple voting, weighted voting, etc. [114]:
 - SEA (Streaming Ensemble Algorithm) [115]: the data stream fills a buffer of a predefined size. When the buffer is full a decision tree is built by the C4.5 algorithm on all the examples in the buffer. Then new buffers are created to repeat the process and these classifiers are put in a pool. When the pool is full: if the new inserted classifier improves the prediction of the pool then it is kept, otherwise it is rejected. In [64] the same technique is used, but different types of classifiers are used in the pool to incorporate more diversity: Bayesian naive, C4.5, RIPPER, etc.
 - In [116], a weight is given to each classifier. The weights are updated only when a wrong prediction of the ensemble occurs. In this case a new classifier is added to the ensemble and the weights of the others are decreased. The classifiers with too small weights are pruned.
 - ADWIN (ADaptative WINdowing) proposed by [98,117] detects a concept change using an adaptive sliding window model. This method uses a reservoir (a sliding window) of size W which grows up when there is no concept drift and decrease when there is a concept drift. The memory used is $\log(W)$. ADWIN looks at all possible sub-windows partitions in the window of training data. Whenever two large enough sub-windows have distinct enough averages, a concept change is detected and the oldest partition of the window is dropped. Then the worst classifier is dropped and a new classifier is added to the ensemble. ADWINs only parameter is a confidence bound, indicating how confident we want to be on the detection.
 - ADACC (Anticipative Dynamic Adaptation to Concept Changes) [118] is a method suggested to deal with both the challenge of optimizing the stability/plasticity dilemma and with the anticipation and recognition of incoming concepts. This is accomplished through an ensemble method that controls an ensemble of incremental classifiers. The management of the ensemble of classifiers naturally adapts to the dynamics of the concept changes with very few parameters to set, while a learning mechanism managing the changes in the ensemble provides means for anticipation and quick adaptation to the underlying modification of the concept.

- or the predictions is based on a single classifier of the ensemble: they are used one by one and other classifiers are considered as a pool of potential candidates [119, 120].

Another way is to weight along the time the examples used to train the decision model. Since the 80's several methods have been proposed as the fading factors [28, 29, 112, 121]. This subject remains relevant for online algorithms. We can cite for example [121]:

- TWF (Time-Weighted Forgetting): the older the example is, the lower is its weight;
- LWF (Locally-Weighted Forgetting): at the arrival of a new example the weights of its closest examples are increased and the weights of the other examples are decreased. The region having recent examples are then kept or created if they did not exist. Regions with no or few examples are deleted. This kind of method is used also for unsupervised learning as in [122];
- PECS (Prediction Error Context Switching): the idea is the same as LWF but all examples are stored in memory and a checkup of the labels of the new examples over the old ones is done. A probability is calculated using the number of examples having or not the same labels. Only the best examples are used to achieve learning.

A recent survey on adaptive methods may be found in [118] and the reader may also be interested in the book [123], the tutorial given during the PAKDD conference [101] or the tutorial given at ECML 2012 [124].

6 Evaluation

In the last years, many algorithms have been published on the topic of supervised classification for data streams. Although Gama in [106] and then in [8] (Chap. 3) suggests an experimental method to evaluate and to compare algorithms for data streams. The comparison between algorithms is not easy since authors do not always use the same evaluation method and/or the same datasets. Therefore the aim of this section is to give, in a first part, references about the criteria used in the literature and in a second part references on the datasets used to perform comparisons. Finally, in a third part we discuss other possible points of comparison.

6.1 Evaluation Criteria

The evaluation criteria are the same as those used in the context of off-line learning, we can cite the main ones:

- Accuracy (ACC): the accuracy is the proportion of true results (both true positives and true negatives) in the population. An accuracy of 100 % means that the measured values are exactly the same as the given values.
- Balanced Accuracy (BER): the balanced accuracy avoids inflated performance estimates on imbalanced datasets. It is defined as the arithmetic mean of sensitivity and specificity, or the average accuracy obtained on either class.

- Area Under the ROC curve (AUC): Accuracy is measured by the area under the ROC curve [125]. An area of 1 represents a perfect test; an area of 0.5 results from a classifier which provides random prediction.
- Kappa Statistic (K) where $K = \frac{(p_0 - p_c)}{(1 - p_c)}$ with p_0 : the prequential accuracy of the classifier and p_c : the probability that a random classifier makes a correct prediction. $K = 1$ if the classifier is always correct and $K = 0$ if the predictions coincide with the correct ones as often as those of the random classifier.
- Kappa Plus Statistic (K^+): a modification a of the Kappa Statistic has recently been presented in [126, 127] where the random classifier is replaced by the “persistent classifier” (which predict always the label of the latest example received).

6.2 Evaluation Methods

In the context of off-line learning the most used evaluation method is the cross validation named k-fold cross validation. This technique is a model validation technique for assessing how the results of a statistical analysis will generalize to an independent data set. The goal of a cross validation is to define a dataset to “test” the model in the training phase (i.e., the validation dataset), in order to limit overfitting, give an insight on how the model will generalize to an independent data set (i.e., an unknown dataset, for instance from a real problem), etc.

One round of cross-validation involves partitioning a sample of data into disjoint subsets, learning the classifier on one subset (called the training set), and validating the classifier on the other subset (called the validation set or testing set). To reduce variability, multiple rounds of cross-validation (k) are performed using different partitions, and the validation results are averaged over the rounds.

But the context is different in the case of online learning on data streams. Depending on whether the stream is stationary or not (presence of concept drift or not) two main techniques exist:

- “Holdout Evaluation” requires the use of two sets of data: the train dataset and the holdout dataset. The train dataset is used to find the right parameters of the classifier trained online. The holdout dataset is then used to evaluate the current decision model, at regular time intervals (or set of examples). The loss estimated in the holdout is an unbiased estimator. The constitution of the independent test set is realized a the beginning of the data stream or regularly (see Fig. 4).
- “Prequential Evaluation” [106]: the error of the current model is computed from the sequence of examples [128]. For each example in the stream, the actual model makes a prediction based only on the example attribute-values. The prequential-error is the sum of a loss function between the prediction and observed values (see Fig. 5). This method is also called: “Interleave Test-Then-Train”. One advantage of this method is that the model is always being tested on unseen examples and no holdout set is needed so that available data are fully used. It also produces smooth plot accuracy over time. One may find prequential evaluation of the AUC in [129].

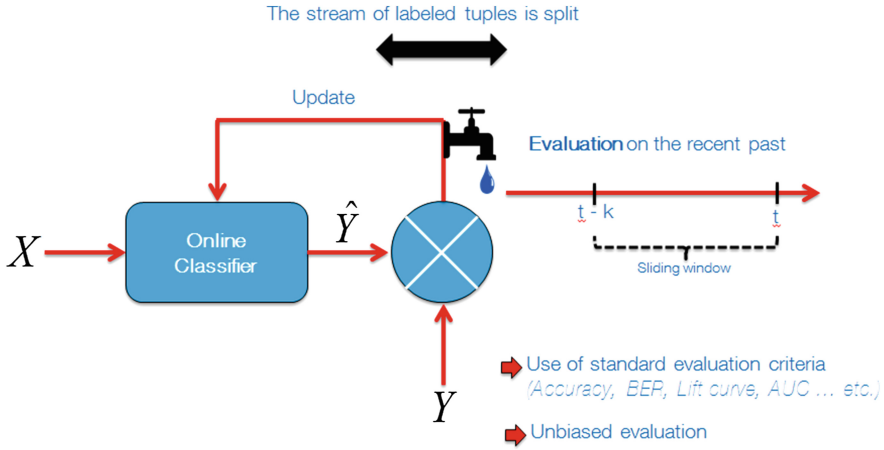


Fig. 4. Holdout Evaluation

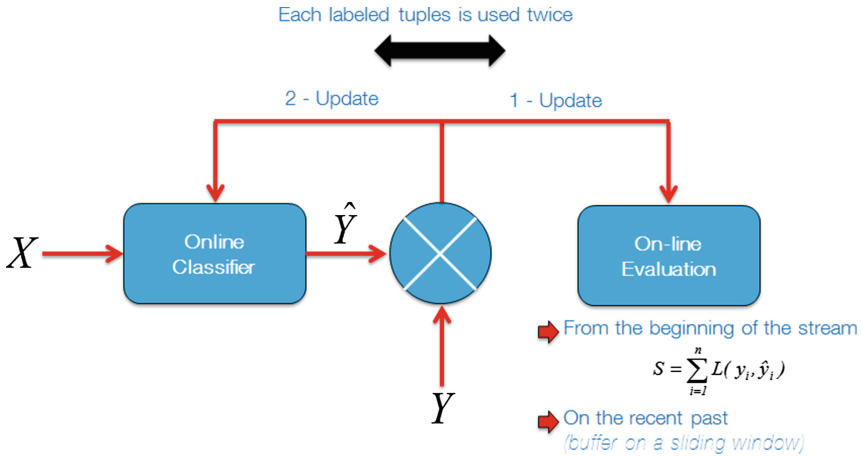


Fig. 5. Prequential Evaluation

Figure 6 compares the errors rates ($1 - Accuracy$) for the algorithm VFDT on the WaveForm dataset (for which the error rate is bounded by an optimal Bayes error of 0.14) using (i) the holdout evaluation and the prequential evaluation [106].

The use of fading factors allows a less pessimistic evaluation as Gama suggested in [106] and as illustrated in Fig. 7. The use of a fading factor decreases the weights of predictions from the past.

In case of:

- stationary data streams: the Holdout or Prequential evaluation can be used, though the Holdout Evaluation is the most used in the literature;
- presence of concept drifts: the Holdout Evaluation cannot be used since the test and train dataset could incorporate examples from distinct concepts.

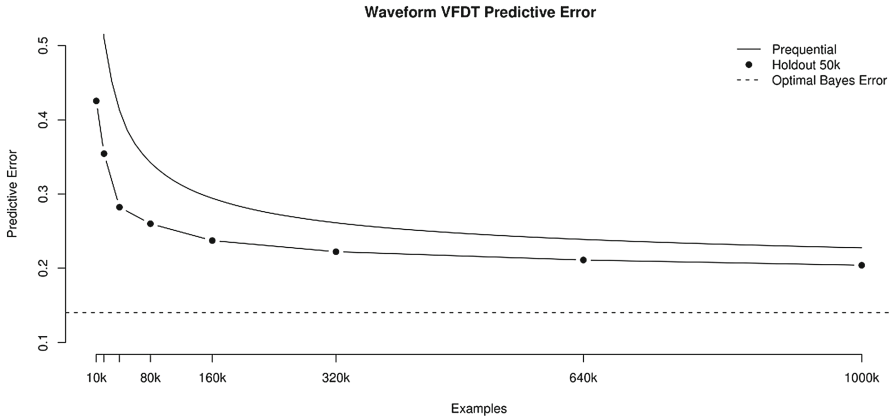


Fig. 6. Comparison between the Holdout Evaluation and the Prequential Evaluation

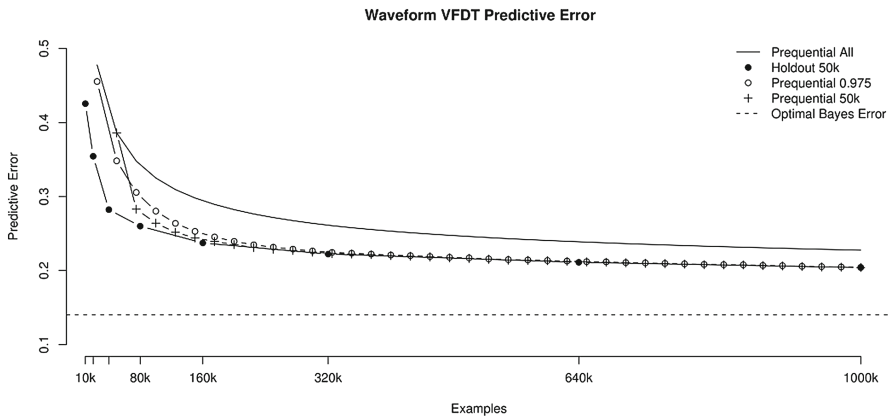


Fig. 7. Comparison between the Holdout Evaluation and the Prequential Evaluation using fading factor

Unlike the Holdout Evaluation, the Prequential Evaluation can be used and is adapted since its test dataset will evolve with the data stream.

6.3 Datasets

In many articles, the UCI Machine Learning Repository is cited as a source of real world datasets. This allows having a first idea of the performance of a new algorithm. To simulate a data stream several approaches can be considered depending on whether the data stream should contain a drift or not. We present below several data streams used in scientific publications. The reader can find other data streams in [8] p. 209.

Synthetic Data Generators without Drift: The best way to have enough data to test online algorithm on data streams, synthetic data generators exist to generate billions of examples. A list and description of these generators is presented in [60]. The main generators are:

- Random RBF [130] creates numerical dataset whose classes are represented in hyper sphere of examples with random centers. Each center has a random position, a single, standard deviation, class label and weight. To create a new example, a direction and a distance are chosen to move the example from the central point. The distance is randomly drawn from a Gaussian distribution with the given centroid standard deviation. The population in a hyper sphere depends on the weight of the centroid.
- Random Tree Generator: produces a decision tree using user parameters and then generates examples according to the decision tree. This generator is biased in favor of decision trees.
- LED Generator [14]: (LED Display Domain Data Set) is composed of 7 Boolean attributes which are predicted LED displays (the light is on or not) and 10 concepts. Each attribute value has the 10 % of its value inverted. It has an optimal Bayes classification rate of 74 %.
- Waveform (Waveform Database Generator Dataset) [14] produces 40 attributes and contains noise. The last 19 attributes are all noise attributes with mean 0 and variance 1. It differentiates between 3 different classes of waves, each of which is generated from a combination of two or three base waves. It has an optimal Bayes classification rate of 86 %.
- Function generator [131] produces a stream containing 9 attributes (6 numeric and 3 categorical) describing hypothetical loan applications. The classes (whether the loan should be approved) are presented in a Boolean label defined by 10 functions.

Synthetic Data Generators with Drift: to test the performance of their online algorithms when there is a concept drift, authors suggested synthetic data generators where the user may include a drift where the position, the size, the level, etc. can be parameterized (see also [8] p. 209). The previous generators may also be changed to contain changes in their data stream.

- SEA Concept generator [115] contains abrupt drift and generates examples with 3 attributes between 0 and 10 where only the first 2 attributes are relevant. These examples are divided into 4 blocks with different concepts by giving each block a threshold value which is bound by the sum of the first two attributes. This data stream has been used in [132].
- STAGGER [28] is a collection of elements, where each individual element is a Boolean function of attribute-valued pairs represented by a disjunct of conjuncts. This data stream has been used in [79, 98, 116, 133].
- Rotating hyperplane [48] uses a hyperplane in d -dimension as the one in SVM and determines the sign of labels. It is useful for simulating time-changing concepts, because we can change the orientation and position of the hyperplane in a smooth manner by changing the relative size of weights. This data stream has been used in [64, 71, 79, 98].

- Minku’s artificial problems: a total of 54 datasets were generated by Minku et al. in 2010, simulating different types of concept changes [134]. A concept change event is simulated by changing one or more of the problems parameters, creating a change in the conditional probability function.

Real Data: in these datasets a change in the distribution (concept drift or covariate shift or both) could exist but the position is not known. Because of their relatively large size they are used to simulate a real data stream:

- Proxy web of the University of Washington [48,56].
- Electricity dataset (Harries, 1999) [132]: This data was collected from the Australian New South Wales Electricity Market. It contains 45,312 examples drawn from 7 May 1996 to 5 December 1998 with one example for each half hour. The class label (DOWN or UP) identifies the change of the price related to a moving average of the last 24 h. The attributes are time, electricity demands and scheduled power transfer between states, all of which are numeric values.
- Forest Cover type dataset (Covertype) contains 581012 instances of 54 categorical and integer attributes describing wilderness areas condition collected from US Forest Service (USFS) Region 2 Resource Information System (RIS) and US Geological Survey (USGS) and USFS data. The class set consists of 7 forest cover types.
- Poker-Hand dataset (Poker Hand): each record is an example of a hand consisting of five playing cards drawn from a standard deck of 52. Each card is described using two attributes (suit and rank), for a total of 10 predictive attributes. There is one Class attribute that describes the “Poker Hand”. The order of cards is important, which is why there are 480 possible Royal Flush hands as compared to 4. This database has been used in [98].
- The KDD 99 cup dataset is a network intrusion detection dataset [135]. The task consists of learning a network intrusion detector able to distinguish between bad connections or attacks and good or normal connections. The training data consists of 494,021 examples. Each example represents a connection described by a vector of 41 features which contain both categorical (ex: the type of protocol, the network service) and continuous values (ex: the length of the connection, its duration). The class label is either 0 or 1 for normal and bad connection, respectively.
- The Airlines task³ is to predict whether a given flight will be delayed, given the information of the scheduled departure. The dataset contains 539,383 examples. Each example is represented by 7 feature values describing the flight (airline, flight number, source, destination, day of week, time of departure and length of flight) and a target value which is either 1 or 0, depending on whether the flight is delayed or not. This dataset is an example of time-correlated stream of data. It is expected that when a flight is delayed (due to weather condition for instance), other timely close flights will be delayed as well.

³ <http://moa.cms.waikato.ac.nz/datasets/>.

- The SPAM dataset [136] consists of 9,324 examples and was built from the email messages of the Spam Assassin Collection using the Boolean bag-of-words representation with 500 attributes. As mentioned in [51], the characteristics of SPAM messages in this dataset gradually change as time passes (gradual concept drift). This database is small but the presence of a known concept drift is interesting.

6.4 Comparison Between Online Classifiers

The same methodology is mainly used in the literature. At first, the authors of a new algorithm perform a comparison with known algorithms but not designed to operate on data streams as: C4.5, ID3, Naive Bayes, Random Forest, etc. The idea is to see the performance on small amounts of data against known algorithms. Then in a second time they confront against other algorithms already present in the literature on data streams. The reader can find in [60] and in [106] a comparison of evaluation methods between some of the algorithms discussed in this paper.

Platform for evaluation: one of the problems when we want to make a comparison between different algorithms is to easily perform experiments. Indeed, it is often difficult to obtain the source code of the algorithm then to successfully build an executable. Furthermore the input formats and output formats are sometimes different between the various algorithms. For off-line learning the Weka toolkit [137] proposed by the University of Waikato allows to quickly perform experiments. The same University proposed a toolkit to perform experiment on data streams for online learning: MOA [133]. MOA contains most of the data generators previously described and many online classifiers as the Hoeffding Trees.

Other points of comparison: the criteria presented in Sect. 6.1 are the mainly used in the context of supervised classification and off-line learning. But in the case of data streams other evaluation criteria could be used as:

- the size of the model: number of nodes for a decision tree, number of rules for rule-based system, etc.
- the learning speed: number of examples that could be learned per second,
- the return on investment (ROI): the idea is to asses and interprets the gains in performance due to model adaptation for a given learning algorithm on a given prediction task [138],
- the RAM-Hours used which is an evaluation measure of the resources used [139].

Comparing the learning speed of two algorithms can be problematic because it assumes that the code/software are publicly available. In addition, this comparison do not evaluates only the learning speed but also the platform on which runs the algorithm and the quality of its implementation. These measures are useful to give an idea but must be taken with caution and not as absolute elements of comparison.

Kirkby [60] realizes his experiments under different memory constraints simulating various environments: sensor network, PDA, server. The idea is therefore to compare different algorithms with respect to the environment in which they are executed.

We can also mention the fact that the precision measurement has no meaning in absolute terms in case of a drift in the data stream. The reader can then turn to learning protocols such as mistake-bound [140]. In this context, learning takes place in cycles where examples are seen one by one; which is consistent with learning on data streams. At the beginning the learning algorithm (\mathcal{A}) learns a hypothesis (f_t) and the prediction for the current instance is $f_t(x_t)$. Then the real label of the example (y_t) is revealed to the learning algorithm which may have made a mistake ($y_t \neq f_t(x_t)$). Then the cycle is repeated until a given horizon time (T). The error bound is related to the error maximum done on the time period T .

Limits of the evaluation techniques: new evaluation techniques have recently been proposed by the community and a consensus is looked for [106, 141]. Indeed, criteria such as the number of examples learned per second or the speed prediction depends on the machine, the memory used and the implementation quality. We must find the criteria and/or common platforms in order to achieve a comparison as impartial as possible.

7 Conclusion

This synthetic introductory article presented the main approaches in the literature for incremental classification. At first (Sect. 2.1) the various problems of learning were presented with the new challenges due to the constraints of data streams: quantity, speed and availability of the data. These characteristics require specific algorithms: incremental (Sect. 3) or specialized for data streams (Sect. 4). The main algorithms have been described in relation to their families: decision tree, naive Bayes, SVM, etc. The Sect. 5 focuses on processing non-stationary data streams that have many concepts. Learning on these streams can be handled by detecting changes in the concept or by having adaptive models. Finally, the last section addresses the evaluation in the context of data streams. New data generators based on drifting concept are needed. At the same time the standard evaluation metrics need to be adapted to the properly evaluate models built on drifting concepts.

Adapting models to changing concept is one of the most important and difficult challenge. Just few methods manage to find a compromise between accuracy, speed and memory. Writing this compromise as a criterion to optimize during the training of the classifier could be a path to explore. We can also note that data streams can be considered as a changing environment over which a classification model is applied. Misclassifications are then available in a limited time horizon. Therefore we can assume that the algorithms related to game theory or reinforcement learning could also be formalized for incremental learning on data streams.

This survey is obviously not exhaustive of all existing methods but we hope that it gives enough information and links on supervised classification for data streams to start on this subject.

References

1. Guyon, I., Lemaire, V., Dror, G., Vogel, D.: Analysis of the kdd cup 2009: fast scoring on a large orange customer database. In: *JMLR: Workshop and Conference Proceedings*, vol. 7, pp. 1–22 (2009)
2. Féraud, R., Boullé, M., Clérot, F., Fessant, F., Lemaire, V.: The orange customer analysis platform. In: Perner, P. (ed.) *ICDM 2010. LNCS*, vol. 6171, pp. 584–594. Springer, Heidelberg (2010)
3. Almaksour, A., Mouchère, H., Anquetil, E.: Apprentissage incrémental et synthèse de données pour la reconnaissance de caractères manuscrits en-ligne. In: *Dixième Colloque International Francophone sur l’écrit et le Document* (2009)
4. Saunier, N., Midenet, S., Grumbach, A.: Apprentissage incrémental par sélection de données dans un flux pour une application de sécurité routière. In: *Conférence d’Apprentissage (CAP)*, pp. 239–251 (2004)
5. Provost, F., Kolluri, V.: A survey of methods for scaling up inductive algorithms. *Data Min. Knowl. Discov.* **3**(2), 131–169 (1999)
6. Dean, T., Boddy, M.: An analysis of time-dependent planning. In: *Proceedings of the Seventh National Conference on Artificial Intelligence*, pp. 49–54 (1988)
7. Michalski, R.S., Mozetic, I., Hong, J., Lavrac, N.: The multi-purpose incremental learning system AQ15 and its testing application to three medical domains. In: *Proceedings of the Fifth National Conference on Artificial Intelligence*, pp. 1041–1045 (1986)
8. Gama, J.: *Knowledge Discovery from Data Streams*. Chapman and Hall/CRC Press, Atlanta (2010)
9. Joaquin Quinonero-Candela, J., Sugiyama, M., Schwaighofer, A., Lawrence, N.D.: *Dataset Shift in Machine Learning*. MIT Press, Cambridge (2009)
10. Bondu, A., Lemaire, V.: Etat de l’art sur les methodes statistiques d’apprentissage actif. *RNTI A2 Apprentissage artificiel et fouille de données*, 189 (2008)
11. Cornuéjols, A.: On-line learning: where are we so far? In: May, M., Saitta, L. (eds.) *Ubiquitous Knowledge Discovery. LNCS*, vol. 6202, pp. 129–147. Springer, Heidelberg (2010)
12. Zilberstein, S., Russell, S.: Optimal composition of real-time systems. *Artif. Intell.* **82**(1), 181–213 (1996)
13. Quinlan, J.R.: Learning efficient classification procedures and their application to chess end games. In: Michalski, R.S., Carbonell, J.G., Mitchell, T.M. (eds.) *Machine Learning - An Artificial Intelligence Approach*, pp. 463–482. Springer, Heidelberg (1986)
14. Breiman, L., Friedman, J., Olshen, R., Stone, C.: *Classification and Regression Trees*. Chapman and Hall/CRC, Boca Raton (1984)
15. Cornuéjols, A., Miclet, L.: *Apprentissage artificiel - Concepts et algorithmes*. Eyrolles (2010)
16. Schlimmer, J., Fisher, D.: A case study of incremental concept induction. In: *Proceedings of the Fifth National Conference on Artificial Intelligence*, pp. 496–501 (1986)
17. Utgoff, P.: Incremental induction of decision trees. *Mach. Learn.* **4**(2), 161–186 (1989)
18. Utgoff, P., Berkman, N., Clouse, J.: Decision tree induction based on efficient tree restructuring. *Mach. Learn.* **29**(1), 5–44 (1997)
19. Boser, B., Guyon, I., Vapnik, V.: A training algorithm for optimal margin classifiers. In: *Proceedings of the Fifth Annual Workshop on Computational Learning Theory*, pp. 144–152. ACM, New York (1992)

20. Cortes, C., Vapnik, V.: Support-vector networks. *Mach. Learn.* **20**(3), 273–297 (1995)
21. Domeniconi, C., Gunopulos, D.: Incremental support vector machine construction. In: *ICDM*, pp. 589–592 (2001)
22. Syed, N., Liu, H., Sung, K.: Handling concept drifts in incremental learning with support vector machines. In: *Proceedings of the Fifth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 317–321. ACM, New York (1999)
23. Fung, G., Mangasarian, O.: Incremental support vector machine classification. In: *Proceedings of the Second SIAM International Conference on Data Mining, Arlington, Virginia*, pp. 247–260 (2002)
24. Bordes, A., Bottou, L.: The Huller: a simple and efficient online SVM. In: Gama, J., Camacho, R., Brazdil, P.B., Jorge, A.M., Torgo, L. (eds.) *ECML 2005. LNCS (LNAI)*, vol. 3720, pp. 505–512. Springer, Heidelberg (2005)
25. Bordes, A., Ertekin, S., Weston, J., Bottou, L.: Fast kernel classifiers with online and active learning. *J. Mach. Learn. Res.* **6**, 1579–1619 (2005)
26. Loosli, G., Canu, S., Bottou, L.: SVM et apprentissage des très grandes bases de données. In: *Cap Conférence d’apprentissage* (2006)
27. Lallich, S., Teytaud, O., Prudhomme, E.: Association rule interestingness: measure and statistical validation. In: Guillet, F., Hamilton, H. (eds.) *Quality Measures in Data Mining. SCI*, vol. 43, pp. 251–275. Springer, Heidelberg (2007)
28. Schlimmer, J., Granger, R.: Incremental learning from noisy data. *Mach. Learn.* **1**(3), 317–354 (1986)
29. Widmer, G., Kubat, M.: Learning in the presence of concept drift and hidden contexts. *Mach. Learn.* **23**(1), 69–101 (1996)
30. Maloof, M., Michalski, R.: Selecting examples for partial memory learning. *Mach. Learn.* **41**(1), 27–52 (2000)
31. Langley, P., Iba, W., Thompson, K.: An analysis of Bayesian classifiers. In: *International Conference on Artificial Intelligence*, pp. 223–228. AAAI (1992)
32. Domingos, P., Pazzani, M.: On the optimality of the simple Bayesian classifier under zero-one loss. *Mach. Learn.* **130**, 103–130 (1997)
33. Kohavi, R.: Scaling up the accuracy of naive-Bayes classifiers: a decision-tree hybrid. In: *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining*, vol. 7. AAAI Press, Menlo Park (1996)
34. Heinz, C.: Density estimation over data streams (2007)
35. John, G., Langley, P.: Estimating continuous distributions in Bayesian classifiers. In: *Proceedings of the Eleventh Conference on Uncertainty in Artificial Intelligence*, pp. 338–345. Morgan Kaufmann (1995)
36. Lu, J., Yang, Y., Webb, G.I.: Incremental discretization for Naïve-Bayes classifier. In: Li, X., Zaïane, O.R., Li, Z. (eds.) *ADMA 2006. LNCS (LNAI)*, vol. 4093, pp. 223–238. Springer, Heidelberg (2006)
37. Aha, D.W. (ed.): *Lazy Learning*. Springer, New York (1997)
38. Brighton, H., Mellish, C.: Advances in instance selection for instance-based learning algorithms. *Data Min. Knowl. Discov.* **6**(2), 153–172 (2002)
39. Hooman, V., Li, C.S., Castelli, V.: Fast search and learning for fast similarity search. In: *Storage and Retrieval for Media Databases*, vol. 3972, pp. 32–42 (2000)
40. Moreno-Seco, F., Micó, L., Oncina, J.: Extending LAESA fast nearest neighbour algorithm to find the k nearest neighbours. In: Caelli, T.M., Amin, A., Duin, R.P.W., Kamel, M.S., de Ridder, D. (eds.) *SPR 2002 and SSPR 2002. LNCS*, vol. 2396, pp. 718–724. Springer, Heidelberg (2002)

41. Kononenko, I., Robnik, M.: Theoretical and empirical analysis of relief and rrelief. *Mach. Learn. J.* **53**, 23–69 (2003)
42. Globersonn, A., Roweis, S.: Metric learning by collapsing classes. In: *Neural Information Processing Systems (NIPS)* (2005)
43. Weinberger, K., Saul, L.: Distance metric learning for large margin nearest neighbor classification. *J. Mach. Learn. Res. (JMLR)* **10**, 207–244 (2009)
44. Sankaranarayanan, J., Samet, H., Varshney, A.: A fast all nearest neighbor algorithm for applications involving large point-clouds. *Comput. Graph.* **31**, 157–174 (2007)
45. Domingos, P., Hulten, G.: Catching up with the data: research issues in mining data streams. In: *Workshop on Research Issues in Data Mining and Knowledge Discovery* (2001)
46. Fayyad, U.M., Piatetsky-Shapiro, G., Smyth, P., Uthurusamy, R.: *Advances in Knowledge Discovery and Data Mining*. American Association for Artificial Intelligence, Menlo Park (1996)
47. Stonebraker, M., Çetintemel, U., Zdonik, S.: The 8 requirements of real-time stream processing. *ACM SIGMOD Rec.* **34**(4), 42–47 (2005)
48. Hulten, G., Spencer, L., Domingos, P.: Mining time-changing data streams. In: *Proceedings of the Seventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 97–106. ACM, New York (2001)
49. Zighed, D., Rakotomalala, R.: *Graphes d'induction: apprentissage et data mining*. Hermes Science Publications, Paris (2000)
50. Mehta, M., Agrawal, R., Rissanen, J.: SLIQ: a fast scalable classifier for data mining. In: Apers, P.M.G., Bouzeghoub, M., Gardarin, G. (eds.) *EDBT 1996*. LNCS, vol. 1057, pp. 18–34. Springer, Heidelberg (1996)
51. Shafer, J., Agrawal, R., Mehta, M.: SPRINT: a scalable parallel classifier for data mining. In: *Proceedings of the International Conference on Very Large Data Bases*, pp. 544–555 (1996)
52. Gehrke, J., Ramakrishnan, R., Ganti, V.: RainForest - a framework for fast decision tree construction of large datasets. *Data Min. Knowl. Disc.* **4**(2), 127–162 (2000)
53. Oates, T., Jensen, D.: The effects of training set size on decision tree complexity. In: *ICML 1997: Proceedings of the Fourteenth International Conference on Machine Learning*, pp. 254–262 (1997)
54. Hoeffding, W.: Probability inequalities for sums of bounded random variables. *J. Am. Stat. Assoc.* **58**, 13–30 (1963)
55. Matuszyk, P., Kreml, G., Spiliopoulou, M.: Correcting the usage of the hoeffding inequality in stream mining. In: Tucker, A., Höppner, F., Siebes, A., Swift, S. (eds.) *IDA 2013*. LNCS, vol. 8207, pp. 298–309. Springer, Heidelberg (2013)
56. Domingos, P., Hulten, G.: Mining high-speed data streams. In: *Proceedings of the Sixth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 71–80. ACM, New York (2000)
57. Gama, J., Rocha, R., Medas, P.: Accurate decision trees for mining high-speed data streams. In: *Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 523–528. ACM, New York (2003)
58. Ramos-Jiménez, G., del Campo-Avila, J., Morales-Bueno, R.: Incremental algorithm driven by error margins. In: Todorovski, L., Lavrač, N., Jantke, K.P. (eds.) *DS 2006*. LNCS (LNAI), vol. 4265, pp. 358–362. Springer, Heidelberg (2006)

59. del Campo-Avila, J., Ramos-Jiménez, G., Gama, J., Morales-Bueno, R.: Improving prediction accuracy of an incremental algorithm driven by error margins. *Knowledge Discovery from Data Streams*, 57 (2006)
60. Kirkby, R.: Improving hoeffding trees. Ph.D. thesis, University of Waikato (2008)
61. Kohavi, R., Kunz, C.: Option decision trees with majority votes. In: *ICML 1997: Proceedings of the Fourteenth International Conference on Machine Learning*, pp. 161–169. Morgan Kaufmann Publishers Inc., San Francisco (1997)
62. Breiman, L.: Bagging predictors. *Mach. Learn.* **24**(2), 123–140 (1996)
63. Robert, E., Freund, Y.: *Boosting - Foundations and Algorithms*. MIT Press, Cambridge (2012)
64. Wang, H., Fan, W., Yu, P.S., Han, J.: Mining concept-drifting data streams using ensemble classifiers. In: *Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining - KDD 2003*, pp. 226–235. ACM Press, New York (2003)
65. Seidl, T., Assent, I., Kranen, P., Krieger, R., Herrmann, J.: Indexing density models for incremental learning and anytime classification on data streams. In: *Proceedings of the 12th International Conference on Extending Database Technology: Advances in Database Technology*, pp. 311–322. ACM (2009)
66. Rutkowski, L., Pietruczuk, L., Duda, P., Jaworski, M.: Decision trees for mining data streams based on the McDiarmid’s bound. *IEEE Trans. Knowl. Data Eng.* **25**(6), 1272–1279 (2013)
67. Tsang, I., Kwok, J., Cheung, P.: Core vector machines: fast SVM training on very large data sets. *J. Mach. Learn. Res.* **6**(1), 363 (2006)
68. Dong, J.X., Krzyzak, A., Suen, C.Y.: Fast SVM training algorithm with decomposition on very large data sets. *IEEE Trans. Pattern Anal. Mach. Intell.* **27**(4), 603–618 (2005)
69. Usunier, N., Bordes, A., Bottou, L.: Guarantees for approximate incremental SVMs. In: *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, vol. 9, pp. 884–891 (2010)
70. Do, T., Nguyen, V., Poulet, F.: GPU-based parallel SVM algorithm. *Jisuanji Kexue yu Tansuo* **3**(4), 368–377 (2009)
71. Ferrer-Troyano, F., Aguilar-Ruiz, J.S., Riquelme, J.C.: Incremental rule learning based on example nearness from numerical data streams. In: *Proceedings of the 2005 ACM Symposium on Applied Computing*, p. 572. ACM (2005)
72. Ferrer-Troyano, F., Aguilar-Ruiz, J., Riquelme, J.: Data streams classification by incremental rule learning with parameterized generalization. In: *Proceedings of the 2006 ACM Symposium on Applied Computing*, p. 661. ACM (2006)
73. Gama, J.A., Kosina, P.: Learning decision rules from data streams. In: *Proceedings of the Twenty-Second International Joint Conference on Artificial Intelligence, IJCAI 2011*, vol. 2, pp. 1255–1260. AAAI Press (2011)
74. Gama, J., Pinto, C.: Discretization from data streams: applications to histograms and data mining. In: *Proceedings of the 2006 ACM Symposium on Applied Computing*, p. 661. ACM (2006)
75. Gibbons, P., Matias, Y., Poosala, V.: Fast incremental maintenance of approximate histograms. *ACM Trans. Database* **27**(3), 261–298 (2002)
76. Vitter, J.: Random sampling with a reservoir. *ACM Trans. Math. Softw.* **11**(1), 37–57 (1985)
77. Salperwyck, C., Lemaire, V., Hue, C.: *Incremental weighted naive Bayes classifiers for data streams. Studies in Classification, Data Analysis, and Knowledge Organization*. Springer, Heidelberg (2014)

78. Law, Y.-N., Zaniolo, C.: An adaptive nearest neighbor classification algorithm for data streams. In: Jorge, A.M., Torgo, L., Brazdil, P.B., Camacho, R., Gama, J. (eds.) PKDD 2005. LNCS (LNAI), vol. 3721, pp. 108–120. Springer, Heidelberg (2005)
79. Beringer, J., Hüllermeier, E.: Efficient instance-based learning on data streams. *Intell. Data Anal.* **11**(6), 627–650 (2007)
80. Shaker, A., Hüllermeier, E.: Iblstreams: a system for instance-based classification and regression on data streams. *Evolving Syst.* **3**(4), 235–249 (2012)
81. Cesa-Bianchi, N., Conconi, A., Gentile, C.: On the generalization ability of on-line learning algorithms. *IEEE Trans. Inf. Theory* **50**(9), 2050–2057 (2004)
82. Block, H.: The perceptron: a model for brain functioning. *Rev. Mod. Phys.* **34**, 123–135 (1962)
83. Novikoff, A.B.: On convergence proofs for perceptrons. In: Proceedings of the Symposium on the Mathematical Theory of Automata, vol. 12, pp. 615–622 (1963)
84. Cesa-Bianchi, N., Lugosi, G.: Prediction, Learning, and Games. Cambridge University Press, New York (2006)
85. Crammer, K., Kandola, J., Holloway, R., Singer, Y.: Online classification on a budget. In: Advances in Neural Information Processing Systems 16. MIT Press, Cambridge (2003)
86. Crammer, K., Dekel, O., Keshet, J., Shalev-Shwartz, S., Singer, Y.: Online passive-aggressive algorithms. *J. Mach. Learn. Res.* **7**, 551–585 (2006)
87. Shalev-Shwartz, S., Singer, Y., Srebro, N.: Pegasos: primal estimated sub-gradient solver for svm. In: Proceedings of the 24th International Conference on Machine Learning, ICML 2007, pp. 807–814. ACM, New York (2007)
88. Kivinen, J., Smola, A.J., Williamson, R.C.: Online learning with kernels. *IEEE Trans. Sig. Process.* **52**(8), 2165–2176 (2004)
89. Engel, Y., Mannor, S., Meir, R.: The kernel recursive least squares algorithm. *IEEE Trans. Sig. Process.* **52**, 2275–2285 (2003)
90. Csató, L., Opper, M.: Sparse on-line Gaussian processes. *Neural Comput.* **14**(3), 641–668 (2002)
91. Thompson, W.R.: On the likelihood that one unknown probability exceeds another in view of the evidence of two samples. *Biometrika* **25**, 285–294 (1933)
92. Bubeck, S., Cesa-Bianchi, N.: Regret analysis of stochastic and nonstochastic multi-armed bandit problems. *Found. Trends Mach. Learn.* **5**(1), 1–122 (2012)
93. Auer, P., Cesa-Bianchi, N., Freund, Y., Schapire, R.E.: The nonstochastic multi-armed bandit problem. *SIAM J. Comput.* **32**(1), 48–77 (2003)
94. Shawe-Taylor, J., Cristianini, N.: Kernel Methods for Pattern Analysis. Cambridge University Press, New York (2004)
95. Sutskever, I.: A simpler unified analysis of budget perceptrons. In: Proceedings of the 26th Annual International Conference on Machine Learning, ICML 2009, Montreal, Quebec, Canada, 14–18 June, pp. 985–992 (2009)
96. Dekel, O., Shalev-Shwartz, S., Singer, Y.: The forgetron: a kernel-based perceptron on a budget. *SIAM J. Comput.* **37**(5), 1342–1372 (2008)
97. Orabona, F., Keshet, J., Caputo, B.: The projectron: a bounded kernel-based perceptron. In: International Conference on Machine Learning (2008)
98. Bifet, A., Holmes, G., Pfahringer, B., Kirkby, R., Gavaldà, R.: New ensemble methods for evolving data streams. In: Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining - KDD 2009, p. 139 (2009)
99. Žliobaite, I.: Learning under concept drift: an overview. *CoRR* abs/1010.4784 (2010)

100. Lazarescu, M.M., Venkatesh, S., Bui, H.H.: Using multiple windows to track concept drift. *Intell. Data Anal.* **8**(1), 29–59 (2004)
101. Bifet, A., Gama, J., Pechenizkiy, M., Žliobaite, I.: *Pakdd tutorial: Handling concept drift: Importance, challenges and solutions* (2011)
102. Marsland, S.: Novelty detection in learning systems. *Neural Comput. Surv.* **3**, 157–195 (2003)
103. Faria, E.R., Goncalves, I.J.C.R., Gama, J., Carvalho, A.C.P.L.F.: Evaluation methodology for multiclass novelty detection algorithms. In: *Brazilian Conference on Intelligent Systems, BRACIS 2013, Fortaleza, CE, Brazil, 19–24 October*, pp. 19–25 (2013)
104. Gama, J., Medas, P., Castillo, G., Rodrigues, P.: Learning with drift detection. In: *Bazzan, A.L.C., Labidi, S. (eds.) SBIA 2004. LNCS (LNAI), vol. 3171*, pp. 286–295. Springer, Heidelberg (2004)
105. Baena-García, M., Del Campo-Ávila, J., Fidalgo, R., Bifet, A., Gavaldà, R., Morales-Bueno, R.: Early drift detection method. In: *Fourth International Workshop on Knowledge Discovery from Data Streams*, vol. 6, pp. 77–86 (2006)
106. Gama, J., Rodrigues, P.P., Sebastiao, R., Rodrigues, P.: Issues in evaluation of stream learning algorithms. In: *Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 329–338. ACM, New York (2009)
107. Page, E.: Continuous inspection schemes. *Biometrika* **41**(1–2), 100 (1954)
108. Mouss, H., Mouss, D., Mouss, N., Sefouhi, L.: Test of page-hinkley, an approach for fault detection in an agro-alimentary production system. In: *5th Asian Control Conference*, vol. 2, pp. 815–818 (2004)
109. Bondu, A., Boullé, M.: A supervised approach for change detection in data streams (2011)
110. Boullé, M.: MODL: a Bayes optimal discretization method for continuous attributes. *Mach. Learn.* **65**(1), 131–165 (2006)
111. Minku, L., Yao, X.: DDD: a new ensemble approach for dealing with concept drift. *IEEE Trans. Knowl. Data Eng.* **24**, 619–633 (2012)
112. Widmer, G., Kubat, M.: Learning flexible concepts from streams of examples: FLORA2. In: *Proceedings of the 10th European Conference on Artificial Intelligence*. Number section 5, pp. 463–467. Wiley (1992)
113. Greenwald, M., Khanna, S.: Space-efficient online computation of quantile summaries. In: *SIGMOD*, pp. 58–66 (2001)
114. Breiman, L.: Bagging predictors. *Mach. Learn.* **24**(2), 123–140 (1996)
115. Street, W., Kim, Y.: A streaming ensemble algorithm (SEA) for large-scale classification. In: *Proceedings of the Seventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 377–382. ACM, New York (2001)
116. Kolter, J., Maloof, M.: Dynamic weighted majority: a new ensemble method for tracking concept drift. In: *Proceedings of the Third International IEEE Conference on Data Mining*, pp. 123–130 (2003)
117. Bifet, A., Gavaldà, R.: Learning from time-changing data with adaptive windowing. In: *SIAM International Conference on Data Mining*, pp. 443–448 (2007)
118. Jaber, G.: An approach for online learning in the presence of concept changes. Ph.D. thesis, Université AgroParisTech (France) (2013)
119. Gama, J., Kosina, P.: Tracking recurring concepts with metalearners. In: *Progress in Artificial Intelligence: 14th Portuguese Conference on Artificial Intelligence*, p. 423 (2009)

120. Gomes, J.B., Menasalvas, E., Sousa, P.A.C.: Tracking recurrent concepts using context. In: Szczuka, M., Kryszkiewicz, M., Ramanna, S., Jensen, R., Hu, Q. (eds.) *RSCTC 2010*. LNCS, vol. 6086, pp. 168–177. Springer, Heidelberg (2010)
121. Salganicoff, M.: Tolerating concept and sampling shift in lazy learning using prediction error context switching. *Artif. Intell. Rev.* **11**(1), 133–155 (1997)
122. Cao, F., Ester, M., Qian, W., Zhou, A.: Density-based clustering over an evolving data stream with noise. In: *2006 SIAM Conference on Data Mining*, pp. 328–339 (2006)
123. Quionero-Candela, J., Sugiyama, M., Schwaighofer, A., Lawrence, N.D.: *Dataset Shift in Machine Learning*. The MIT Press, Cambridge (2009)
124. Bifet, B., Gama, J., Gavaldà, R., Kreml, G., Pechenizkiy, M., Pfahringer, B., Spiliopoulou, M., Žliobaite, I.: Advanced topics on data stream mining. Tutorial at the *ECMLPKDD 2012* (2012)
125. Fawcett, T.: ROC graphs: notes and practical considerations for researchers. *Mach. Learn.* **31**, 1–38 (2004)
126. Bifet, A., Read, J., Žliobaite, I., Pfahringer, B., Holmes, G.: Pitfalls in benchmarking data stream classification and how to avoid them. In: Blockeel, H., Kersting, K., Nijssen, S., Železný, F. (eds.) *ECML PKDD 2013, Part I*. LNCS (LNAI), vol. 8188, pp. 465–479. Springer, Heidelberg (2013)
127. Žliobaite, I., Bifet, A., Read, J., Pfahringer, B., Holmes, G.: Evaluation methods and decision theory for classification of streaming data with temporal dependence. *Mach. Learn.* **98**, 455–482 (2015)
128. Dawid, A.: Present position and potential developments: some personal views: statistical theory: the prequential approach. *J. Roy. Stat. Soc. Ser. A (General)* **147**, 278–292 (1984)
129. Brzezinski, D., Stefanowski, J.: Prequential AUC for classifier evaluation and drift detection in evolving data streams. In: *Proceedings of the Workshop New Frontiers in Mining Complex Patterns (NFMCP 2014) held in European Conference on Machine Learning (ECML)* (2014)
130. Bifet, A.: Adaptive learning and mining for data streams and frequent patterns. Ph.D. thesis, Universitat Politècnica de Catalunya (2009)
131. Agrawal, R.: Database mining: a performance perspective. *IEEE Trans. Knowl. Data Eng.* **5**(6), 914–925 (1993)
132. Gama, J., Medas, P., Rodrigues, P.: Learning decision trees from dynamic data streams. *J. Univ. Comput. Sci.* **11**(8), 1353–1366 (2005)
133. Bifet, A., Kirkby, R.: Data stream mining a practical approach. *J. Empirical Finance* **8**(3), 325–342 (2009)
134. Minku, L.L., White, A.P., Yao, X.: The impact of diversity on online ensemble learning in the presence of concept drift. *IEEE Trans. Knowl. Data Eng.* **22**(5), 730–742 (2010)
135. Tavallaee, M., Bagheri, E., Lu, W., Ghorbani, A.A.: A detailed analysis of the kdd cup 99 data set. In: *Proceedings of the Second IEEE International Conference on Computational Intelligence for Security and Defense Applications, CISDA 2009*, pp. 53–58. IEEE Press, Piscataway (2009)
136. Katakis, I., Tsoumakas, G., Vlahavas, I.: Tracking recurring contexts using ensemble classifiers: an application to email filtering. *Knowl. Inf. Syst.* **22**(3), 371–391 (2010)
137. Witten, I.H., Frank, E.: *Data Mining: Practical Machine Learning Tools and Techniques*. Morgan Kaufmann Series in Data Management Systems, 2nd edn. Morgan Kaufmann, San Francisco (2005)

138. Žliobaitė, I., Budka, M., Stahl, F.: Towards cost-sensitive adaptation: when is it worth updating your predictive model? *Neurocomputing* **150**, 240–249 (2014)
139. Bifet, A., Holmes, G., Pfahringer, B., Frank, E.: Fast perceptron decision tree learning from evolving data streams. In: Zaki, M.J., Yu, J.X., Ravindran, B., Pudi, V. (eds.) *PAKDD 2010. LNCS*, vol. 6119, pp. 299–310. Springer, Heidelberg (2010)
140. Littlestone, N., Warmuth, M.: The weighted majority algorithm. In: *30th Annual Symposium on Foundations of Computer Science*, pp. 256–261 (1989)
141. Kreml, G., Žliobaite, I., Brzezinski, D., Hllermeier, E., Last, M., Lemaire, V., Noack, T., Shaker, A., Sievi, S., Spiliopoulou, M., Stefanowski, J.: Open challenges for data stream mining research. *SIGKDD Explorations (Special Issue on Big Data)* **16**, 1–10 (2014)