FOUNDATIONS OF COMPUTING AND DECISION SCIENCES
Vol. 38 (2013) No. 1

DOI: 10.2478/v10209-011-0020-y

INCREMENTAL RULE-BASED LEARNERS FOR HANDLING CONCEPT DRIFT: AN OVERVIEW

Magdalena DECKERT *

Abstract. Learning from non-stationary environments is a very popular research topic. There already exist algorithms that deal with the concept drift problem. Among them there are online or incremental learners, which process data instance by instance. Their knowledge representation can take different forms such as decision rules, which have not received enough attention in learning with concept drift. This paper reviews incremental rule-based learners designed for changing environments. It describes four of the proposed algorithms: FLORA, AQ11-PM+WAH, FACIL and VFDR. Those four solutions can be compared on several criteria, like: type of processed data, adjustment to changes, type of the maintained memory, knowledge representation, and others.

Keywords: data mining, decision rules, rule-based classifiers, incremental learning, online learning, data streams, concept drift, non-stationary environment, overview

1 Introduction

Data mining is a relatively young and interdisciplinary field of computing science. It is one of the steps in the Knowledge Discovery in Databases (KDD) process that tries to discover patterns and dependencies in large data sets. One subtask of data mining is the classification problem. It identifies class labels to which a new observation belongs using knowledge extracted from labeled training examples. Most of the existing classifiers are created statically. They receive the whole learning set, from which knowledge is extracted. The knowledge is obtained only once and is not updated in the future. Those standard classifiers fail to answer modern challenges like processing streaming data.

*Institute of Computing Science, Poznań University of Technology, 60-965 Poznań, Poland

Data streams are characterized by the large size of data, probably infinite. Processing streaming data may be very expensive due to multiple data access. That is why many classifiers try to minimize the number of reads. The second problem with data streams is how many examples to remember. Classifiers may have a *full memory*—they remember all training data, a *partial memory*—they memorize some important learning examples, or *no memory*. Some of the algorithms remember only meta data connected with learning examples. Data streams can be processed by *online classifiers*. Those classifiers should have the following qualities [28]:

- Single pass through the data. The classifier reads each example only once.
- Limited memory and processing time. Each example should be processed very fast and in a constant period of time.
- Any-time learning. The classifier should provide the best answer at every moment of time.

Processing of data streams is a very popular and interesting research topic. An example of system designed for stream analysis can be found in [30]. While processing streaming data a problem can be encountered that the environment and the classification task may change in time. The concepts of interest may depend on some hidden context [44], which is unknown. Changes in the hidden context can induce more or less radical changes in target concepts, producing what is generally known as *concept drift* [37]. One of the common examples of changing environments is spam detection. The description of assignment to different groups of e-mails changes with time. They depend on user preferences and active spammers, who invent new solutions to trick the up-to-date classifier. The problem with concept drift is real and has a wide range of applications. According to Zliobaite [45] applications' domains can be divided into 4 main groups: Monitoring and control, Assistance and information, Decision making, and AI and robotics. One of the typical monitoring problems is intrusion detection. The attackers try to invent new ways of overcoming current security systems, which is a source of concept drift. Other examples of occurrence of concept drift from Monitoring and control group are fraud detection in financial sector or traffic management. Applications from Assistance and information domain mainly organize and/or personalize the flow of information. The cost of mistake is relatively low. An example of such an application is customer profiling and direct marketing, where customer's needs and interests change with time. Also smart home systems should adapt to the changing environment and user's needs. This is an example of application from the AI and robotics domain. A wide range of occurrences of the concept drift problem was presented in [19, 45]. Systems designed for specific applications like food sales or CFB Boilers were described in [3, 46, 47].

A more formal definition of the concept drift may be as follows. In each point of time t every example is generated by source S_t, which is a distribution over the data. Concepts are *stable* if all examples are sampled by the same source, otherwise concept drift exists [45].

Two main types of concept drift may be distinguished: *sudden (abrupt)* and *gradual (incremental)* [42]. In case when a source at time t is suddenly replaced with

another one a sudden concept drift occurs. For example, John was listening to pop music his whole teenage life but when he graduated from university he changed his preferences and started to listen only to classical music. A gradual drift would occur if John started to listen to classical music while he was still enjoying pop music but the interest in pop decreased with time. In this case, the probability of sampling from the first source decreases with time, while the probability of sampling from the second source increases. In some domains previously seen concepts may reappear after some period of time. This type of change is known as a *recurring context* or *recurring concept*. Periodic seasonality is not considered to be a concept drift problem. Reoccurring concepts differ from common seasonality because it is not known when they may reappear [45]. Other examples of change worth mentioning are noise and blips [29]. Noise is a non-significant change and a good online classifier should not react to it. A blip represents a rare event that should be treated as an outlier and should be discarded.

Mining data streams in the presence of concept drift is rather a new topic in the machine learning world but there already exist algorithms that attempt to solve this problem. For a taxonomy of available concept drift learners see [45]. In general, they can be divided into two main groups: trigger-based and evolving.

The trigger-based model contains a change detector that indicates a need for model change. The change detection process is separate from classification. Standard actions of classifiers equipped with a detector are as following: the classifier predicts a label for received example e; then the true label and the predicted label are submitted to the change detector; if the detector detects a change, the feedback is passed to the classifier; then the classifier is retrained according to the level of change [29]. One of the most popular drift detection methods is DDM proposed by Gama et al. in [17]. This approach detects changes in the probability distribution of examples. The main idea of this method is to monitor the error-rate produced by a classifier. Statistical theory affirms that the error decreases if the distribution is stable [17]. When the error increases, it signifies that the distribution has changed. DDM operates on labeled data that arrive one at a time. Another interesting detector that performs better than DDM for a slow gradual drift is EDDM proposed in [2]. It uses the distance between classification errors in order to detect a change. There is also a solution that detects change from data arriving in batches, called Batch Drift Detection Method (BDDM). It was proposed in [10] and improved in [11].

Evolving methods operate in a different way than trigger-based solutions. They try to build the most accurate classifiers at each moment of time without explicit information about the occurrence of a change. The most popular evolving technique for handling concept drift is an *ensemble of classifiers* [45]. An example of such an ensemble is Accuracy Weighted Ensemble (AWE) [43]. It is the best representative of *block-based ensembles*, where component classifiers are constructed from sequential-coming blocks of training data. When a new block is available, a new classifier is built from it and already existing component classifiers are evaluated. The new classifier usually replaces the worst component in the ensemble. For an overview of available complex methods see [18, 28, 29, 42, 45].

There also exist hybrid methods that incorporate explicit drift detector with an

ensemble of classifiers. An example of such an approach is Batch Weighted Ensemble (BWE) introduced in [10] and improved in [11]. BWE uses Batch Drift Detection Method (BDDM) to detect an occurrence of change and updates its ensemble according to the type of change. Another block ensemble that is combined with an online drift detector is Adaptive Classifiers Ensemble (ACE) proposed in [36]. This system besides a drift detection mechanism and many batch learners contains also an online learner.

This paper focuses on *incremental* or *online* learning. A learning task is incremental if the training examples become available over time, usually one at a time [20]. In this case learning may need to last indefinitely. This type of learning is similar to a human's acquisition of knowledge. People learn all the time and their knowledge is constantly revised based on newly gathered information. The term "incremental" is also applied to learning algorithms. An algorithm is online if, for given training examples, it produces a sequence of hypotheses such that the current hypothesis depends only on the previous one and on the current learning example e [20]. All learning algorithms are applicable to all learning tasks. However, the most natural and flexible way to handle incremental learning tasks is to use incremental learners. Unfortunately, incremental learning is a rather forgotten area in the machine learning world [20]. Nevertheless, there exist many incremental learning algorithms inducing different types of knowledge. An example of an incremental classifier inducing decision rules was described in [21]. However most of the existing solutions are not applicable for processing data streams.

One of the most popular incremental method for mining data streams is Very Fast Decision Trees (VFDT) proposed in [12]. It is a anytime system that builds decision trees using constant memory and constant time per example. VFDT uses Hoeffding bound to guarantee that its output is asymptotically nearly identical to the result obtained by a batch learner. VFDT was improved in [25] to deal with the concept drift problem. CVFDT uses a sliding window on incoming data and old data, which fall outside the window, is forgotten.

Another knowledge representation that was adjusted to processing data streams are decision rules. Decision rules can provide descriptions that are easily interpretable by a human. They are also very flexible and can be quickly updated or removed when a change occurs. Decision rules cover selected parts of the space, so if they become out-of-date there is no need to learn from scratch—only the rules that cover regions with the change should be revised. However, according to Gama [19], they have not received enough attention in the stream mining community so far.

Decision rules can be more effective for mining data streams than other methods. In case of algorithms based on Hoeffding Trees, the adaptation to change is performed via incremental growth of a tree. However, for sudden change the reaction might be to slow due to the fact that it might require rebuilding the whole tree structure. This might be very inefficient. Decision rules are more flexible than trees. A set of decision rules take advantage of individual rules that can be managed independently [27]. Therefore, they can be altered more easily if change occurred or even removed if necessary. For gradual concept drift, the adaptation to change has probably similar complexity for both knowledge representations. Next, decision trees split the data

space, where decision rules cover parts of the data space. While processing data instance by instance, a tree might need more changes in global model, while decision rules are updated independently. On the other hand, the process of incremental rule induction is more sophisticated than induction of decision tree. This may be the reason why decision rules are not as popular as decision trees for mining data streams.

According to the author's best knowledge, there does not exist any survey of incremental rule-based classifiers learning from non-stationary environments. The goal of this paper is to present the key online algorithms proposed for mining data streams in the presence of concept drift. It describes four of the proposed algorithms: FLORA, AQ11-PM+WAH, FACIL and VFDR. Those are the only purely incremental rule-based classifiers mining data streams in the presence of concept drift. First, the FLORA framework is described—a first family of algorithms that flexibly react to changes in concepts, can use previous knowledge in situations when contexts reappear and is robust to the noise in data [44]. Then, algorithms from the AQ family are presented with their modifications. AQ-PM [31] is a static learner that selects extreme examples from rules' boundaries and stores them in the partial memory for each incoming batch of data. AQ11-PM [32] is a combination of the incremental AQ11 algorithm with a partial memory mechanism. AQ11-PM+WAH [33] is extended with a heuristic for flexible size of the window with stored examples. The FACIL algorithm behaves similarly to AQ11-PM [13]. However, it differs in a way that examples stored in the partial memory do not have to be extreme ones. Those three main algorithms were not tested on massive datasets. The newest proposal called VFDR [19] was tested on huge data streams. It induces ordered or unordered sets of decision rules that are efficient in terms of memory and learning times.

This paper is organized as follows. The next section presents the basics of rule induction. Section 3 describes the first incremental rule-based learners for a concept drift problem—the FLORA family. Section 4 is devoted to the AQ family algorithms, e.g., AQ11-PM+WAH. Section 5 familiarizes with the FACIL algorithm. Section 6 reveals the newest algorithms VFDR and AVFDR. Section 7 concludes this paper.

2 Basics of the Rule Induction

A classification problem relates to an exploration of hypotheses describing so-called concepts. The term *concept* denotes a set of objects with some common characteristics that distinguish it from other concepts. In order to describe similar features the terms *category* or *class* are also used. *Hypotheses* are results of supervised learning. They are functions which best describe concepts from the supplied learning examples. Generally, hypotheses assign examples to the appropriate category (class). Those functions can be expressed in different forms. One of the most popular methods of knowledge representation are decision rules. There exist many algorithms that induce decision rules. For reviews see [15, 16, 23]. Most of the existing classifiers extract knowledge from static data. As input they obtain the whole learning set, from which hypotheses are found.

The set of learning examples may be represented in several ways, most com-

mon is a decision table. A decision table DT is a data structure of the form: $DT = (U, A \cup \{d\})$. Elements of A are called conditional attributes, where d is a decision attribute. U is a set of learning examples representing different concepts. The collection of objects U can be divided with respect to concept C_k into positive $E^+_{C_k} = E_{C_k}$ and negative examples $E^-_{C_k} = U \backslash E_{C_k}$.

Decision rule r for concept C_k is defined as an expression taking the form:

$$\textit{if } P \textit{ then } Q.$$

P is the *conditional part* of the rule (*premise; antecedent*). For conditional part the term *description item* or *description* can also be used. Q is the *decision part* of the rule (*conclusion; label*) indicating affiliation to concept C_k. In the literature, a decision rule can also take the form:

$$P \rightarrow Q.$$

Conditional part P of a rule r is a *conjunction of elementary conditions* and is represented in the form of:

$$P = \textit{condition}_1 \wedge \ldots \wedge \textit{condition}_l,$$

where l is the number of conditions known as the length of the rule. A single elementary $\textit{condition}_i$ (*selector*) is represented as:

$$at_i \; rel \; v_i,$$

where at_i is a conditional attribute i and v_i is a value from the domain of attribute at_i. rel is a relation operator from the set of relations $\{=, \neq, <, \leq, >, \geq, \in\}$ [40].

Rule r covers an example when attributes of the example match the rule's conditions. Rules can cover both positive and negative examples. Examples from the learning set that fulfill conditional part P of rule r are called coverage and are indicated by [P]. Rule r is *discriminant* or *certain*, when it covers only positive examples (no negative examples covered). Thanks to this the rule distinguishes examples belonging to the class indicated by the rule's decision part. A discriminant rule r is minimal, if removing of one of its selectors results in negative examples being covered. There also exist other types of decision rules like probabilistic rules. They do not indicate a single category but return probabilities connected with every decision class' label. Probability estimation techniques for rule learners are considered in [41].

The problem of finding a minimal set of rules covering learning examples is NP-complete. Many heuristic algorithms exist that induce decision rules. One of the most popular techniques is sequential covering. In general, it relies on learning a single rule for a given concept, removing examples covered by the rule and repeating this process for other examples from the same concept. Next, rules for other concepts are generated sequentially. The pseudocode of a sequential covering mechanism is presented as Algorithm 1.

The function LearnSingleRule (line 5) depends on the used algorithm—sample realizations can be found in [6, 8, 9, 22, 34, 35]. In most of these algorithms, the

Algorithm 1: Sequential Covering algorithm

Input : U—a set of learning examples;
A—conditional attributes
Output: RS—a set of induced rules

```
1 RS = ∅;
2 foreach different concept_i do
3 |  U_i = U;
4 |  while all examples for concept_i from U_i are not covered do
5 |  |  r = LearnSingleRule(concept_i, A, U_i);
6 |  |  RS = RS ∪ r;
7 |  |  U_i = U_i \ [RS];
8 Return RS
```

initial candidate for the conditional part of the rule covers the set of all learning examples including the negative ones. Then the rule is specialized by adding elementary conditions until the acceptance threshold is reached. Candidates for the elementary conditions of a rule are evaluated with respect to different measures depending on the algorithm. The most commonly used criteria are as follows [39]:

- Maximizing the number of positive examples covered by the conjunction of elementary conditions in P.
- Maximizing the ratio of covered positive examples to the total number of examples covered.
- Minimizing the number of elementary conditions in P—minimizing the length of the rule.

Other algorithms use an *entropy* of information to evaluate the conditional part of the rule. It was introduced by Shannon in [38]. The entropy of information of given learning set S is defined as:

$$Ent(S) = -\sum_{i=1}^{n_c} p_i * log_2 p_i, \tag{2.1}$$

where p_i is the probability of class C in the set of examples S and n_c is the number of different class labels. The entropy is a cost type measure—the smaller the value is, the better is the conjunction in P.

Another important measure of evaluating the dependence of P and Q is the *m-estimate* proposed by Cestnik in [7]. The definition of *m-estimate* is:

$$m\text{-}estimate_{C_k}(P) = \frac{n_p + m * p_i}{n + m}, \tag{2.2}$$

where n_p is the number of positive examples covered by P, n is the total number of all examples covered by P, p_i is the prior probability of the class C_k and m is a constant depending on the data.

A special case of m-estimate is the Laplace estimate defined as:

$$L\text{-}estimate(P) = \frac{n_p + n_c - 1}{n + n_c}, \tag{2.3}$$

where n_c is the number of different class labels. More about these measures can be found in [40, 41].

One of the first algorithms basing on the sequential covering idea is AQ, proposed by Michalski [35]. It operates as follows. At the beginning of each iteration, the currently processed decision class is chosen. Next, sets with positive and negative examples are created with respect to the given class label. Then, a seed is selected randomly from the positive examples. In the next step, a star is generated. A star is a set of all rules that cover the seed and does not cover any of the negative examples. Extending the seed against all negative examples is a multistep procedure. While the star covers negative examples, select one of them. Then, all maximally general rules that cover the seed and exclude the negative example are found. The resulting set is called a partial star of the seed against the negative example. Next, a new partial star is generated by intersecting the initial star with the partial star of the seed against the negative example. In the end, a new partial star is trimmed if the number of rules exceeds the user defined threshold and the new partial star becomes a star. This threshold was introduced in order to limit the search space, which would grow rapidly with the number of negative examples and with the number of attributes. A typical criterion for trimming is the number of positive examples covered. In case of a tie, the minimum number of selectors is preferred. The procedure of star extension is repeated until the star no longer covers any negative examples. After the star is created, the best rule from the star is chosen according to the user-defined criteria. The rule is added to the current set of rules. This mechanism iteratively induces decision rules until all positive examples from the given decision class are covered. The whole process is rerun for every label of the decision class. For details see [35].

Another algorithm—CN2, proposed in [9], modifies the AQ algorithm in a way that it removes the dependence on specific examples and increases the space of searched rules. Unlike the AQ-based system, which induces an unordered set of decision rules, CN2 produces an ordered list of if-then rules. CN2 works in an iterative fashion. In each iteration, it searches for a rule that covers a large number of examples of the single class C_k and few of other classes. When the best rule according to the entropy measure is found, the algorithm removes the covered examples from the training set and adds the rule to the end of the rule list. This process is repeated until no more satisfactory rules can be found. CN2 searches for new rules by performing a general-to-specific search. At each stage, CN2 retains a size-limited set or star S of the best rules found so far. The system examines only specializations of this set, performing a beam search of the space of rules. A rule is specialized by either adding a new elementary condition or removing disjunctive values from one of its selectors. Each rule can be specialized in several ways—CN2 generates and evaluates all of them. In the end, star S is trimmed by removing rules with the lowest ranking values measured by given evaluation function—the likelihood ratio statistic. For more details see [9].

Another representative of the rule-based algorithms is MODLEM, which was originally introduced by Stefanowski in [39]. Generally, it is based on the scheme of

sequential covering and it generates an *unordered minimal set of rules* for every decision concept. It is particularly well-suited for analyzing data containing a mixture of numerical and qualitative attributes, inconsistent descriptions of objects, or missing attribute values. Searching for the best single rule and selecting the best condition is controlled by a criterion based on an entropy measure. For more details see [39].

Induced set of decision rules can be used for classification of new incoming examples. Those new examples were not used during the learning phase. Their description of conditional attributes is known and the goal is to determine the correct decision class label. A classification of the new examples is based on matching the description of the new object to the conditional part of a decision rule. Two main matching types can be distinguished: *full* or *strict* and *partial* or *flexible* matching. Full matching takes place, when all elementary conditions of a rule match the example's attributes. In case of partial matching there must exist at least one elementary condition of a rule that does not match the new object's description.

Classification strategy is performed in a different way depending on whether the decision rules are sorted to form a list or create a random set of rules. In case of an unordered list of decision rules, only the first rule that matches the example is fired and the label associated with the rule determines the example's class label. When the first rule covering the example is found, the rest of the rules are not visited. In case when none of the rules match the example, the default rule is used. Generally, the default rule indicates the majority class in the training set—the largest class in the training set.

In case of an unordered set of decision rules using *full* or *strict matching* three situations are possible: a unique match (to one or more rules from the same class); *matching more rules* from different classes or *not matching* any rules at all. In both latter situations the suggestion is ambiguous, thus, a proper resolution strategy is necessary. One of the solutions is the strategy introduced by Grzymala-Busse [24]. It has been successfully applied in many experiments. Generally, it is based on a voting of matching rules with their supports. The total *support* for class C_k is defined as:

$$sup(C_k) = \sum_{i}^{n_r} sup(r_i), \tag{2.4}$$

where r_i is a matched rule that indicates class C_k, n_r is the number of these rules and $sup(r_i)$ is the number of learning objects satisfying both condition and decision parts of the rule r_i. A new object is assigned to the class with the highest total support. In the case of not-matching, so called *partial matching* or *flexible matching* is considered, where at least one of the rule's conditions is satisfied by the corresponding attributes in the new object's description x. In this case, a matching factor *match(r,x)* is introduced as the ratio of conditions matched by object x to all conditions in rule r. The total support is modified to:

$$sup(C_k) = \sum_{i}^{p} match(r, x) * sup(r_i), \tag{2.5}$$

where p is the number of partially-matched rules, and object x is assigned to the class with the highest value of $sup(C_k)$.

Another example of classification strategy is the proposal of Aijun Ann in [1]. It uses a rule quality measure different than rule support, i.e., a *measure of discrimination*:

$$Q_{MD} = log \frac{P(r|C_k) * (1 - P(r|\neg C_k)}{P(r|\neg C_k) * (1 - P(r|C_k))}, \tag{2.6}$$

where P denotes probability. For more technical details of estimating probabilities and adjusting this formula to prevent zero division see [1]. Its interpretation says that it measures the extent to which rule r discriminates between positive and negative objects of class C_k. The only difference between these two described classification strategies is choosing another rule quality measure—putting Q_{MD} in place of $sup(r)$.

Moreover, classification strategies can be adopted to abstaining from a class prediction when the final decision is uncertain. This modification can influence the final accuracy of classification of an ensemble consisting of rule-based component classifiers. This idea was inspected by Błaszczyński et al. in [5].

Because of the natural and easy form of representation, decision rules can be inspected and interpreted by a human. They are also more comprehensive than any other knowledge representation. Generally, they provide good interpretability and flexibility for data mining tasks. They take advantage of not being hierarchically structured, so hypotheses can be easily updated when becoming out-of-date without significant decrease in performance. However, they have not received enough attention in mining data streams.

3 FLORA

Effective learning in environments with hidden contexts and concept drifts requires a learning algorithm which fulfills certain conditions [44]:

- it can detect context changes without being explicitly informed;
- it can quickly recover from a concept change and adjust its hypotheses;
- it can make use of previous descriptions when concepts reappear.

One of the possible solutions is to trust only the latest examples—this is known as the windowing mechanism. The window of examples may be of a fixed or a flexible size. New examples are added to the window as they arrive and the old ones are removed, when appropriate conditions are fulfilled. Those activities in window trigger modifications of current hypotheses in order to be consistent with the examples held in the window. This idea is widely used and states the main essence of the FLORA framework proposed in [44].

The FLORA framework is restricted to processing data containing only nominal attributes and can only solve the binary classification problem. In the FLORA framework each concept is represented by three sets comprising rules' antecedents: ADES (Accepted DEScriptors), NDES (Negative DEScriptors) and PDES (Potential DEScriptors). ADES contains descriptions covering only positive examples, and NDES

only negative examples. PDES consists of descriptions that match both positive and negative examples. ADES is used to classify new incoming examples, while NDES is used to prevent the over-generalization of ADES. PDES acts as a storage for descriptions that might become relevant in the future [44]. Every description item has corresponding counters, which indicate how many positive or negative examples from current window are covered by the given description. The counters are updated with every modification of the learning window (addition or deletion of a learning example). A description item is held in memory as long as it covers at least one example from the window. The simple FLORA framework is presented as Algorithm 2.

The FLORA framework operates as follows. When a new positive example is added to the learning window, three situations are possible: a new description item is added to ADES, descriptions existing in ADES are generalized to match the new example, or/and existing items are moved from NDES to PDES (lines 1–17). First, the ADES set is tested in order to find a description covering the incoming positive example (lines 3–6). If there does not exist such an item, a generalization of descriptions from ADES is performed (lines 7–8). If there is no covering item in ADES and there does not exist any generalization that matches the example, the example's full description is added to the ADES set (lines 9–10). Then, the PDES set is searched and counters of positive examples are incremented for the description items that cover the example (lines 11–13). In the end, the NDES set is visited. Descriptions that match the new positive example are moved to PDES and their counters are updated (lines 14–17). In case when the incoming example is negative—same situations are possible but in respect to the NDES set (lines 18–34). First, the NDES set is tested in order to find a description covering the incoming negative example (lines 20–23). If there does not exist such an item, a generalization of descriptions from NDES is performed (lines 24–25). If there is no covering item in NDES and there does not exist any generalization that matches the example, the example's full description is added to the NDES set (lines 26–27). Then, the PDES set is searched and counters of negative examples are incremented for the description items that cover the example (lines 28–30). In the end, the ADES set is visited. Descriptions that match the new negative example are moved to PDES and their counters are updated (lines 31–34). When an example is deleted from the learning window, appropriate counters are decreased (lines 35–59). This may result in a removal of a description or its migration from PDES to ADES or NDES, with respect to the type of example: negative or positive. If the example to be deleted is positive, first the ADES set is visited. Counters of positive examples are decremented for the description items that match the example. If the counter is equal to 0, then the description from ADES is dropped (lines 38–42). Then the PDES set is tested. Counters of positive examples are decremented for the description items that match the example. If the counter equals 0, then the description is moved from PDES to NDES (lines 43–47). If the example to be deleted is negative, first the NDES set is visited. Counters of negative examples are decremented for the description items that match the example. If the counter is equal to 0, then the description from NDES is dropped (lines 49–53). Then the PDES set is tested. Counters of negative examples are decremented for the description items that match the example. If the counter equals 0, then the description is moved from PDES to ADES (lines 54–58).

Algorithm 2: simple FLORA algorithm

```
Input  : E—incoming example;
         ADES—a set with accepted descriptors;
         PDES—a set with potential descriptors;
         NDES—a set with negative descriptors
Output: ADES,PDES,NDES—modified description sets
if E is positive example then
    boolean match = false;
    foreach i = 1 .. |ADES| do
        if ADES_i covers E then
            increment i-th descriptor's counter for positive examples;
            match = true;
    if match == false then
        find generalization in ADES with respect to PDES and NDES;
    if (match == false) and (generalization does not exist) then
        add full example's description to ADES;
    foreach i = 1 .. |PDES| do
        if PDES_i covers E then
            increment i-th descriptor's counter for positive examples;
    foreach i = 1 .. |NDES| do
        if NDES_i covers E then
            increment i-th descriptor's counter for positive examples;
            move i-th descriptor from NDES to PDES;
else if E is negative example then
    boolean match = false;
    foreach i = 1 .. |NDES| do
        if NDES_i covers E then
            increment i-th descriptor's counter for negative examples;
            match = true;
    if match == false then
        find generalization in NDES with respect to PDES and ADES;
    if (match == false) and (generalization == null) then
        add full example's description to NDES;
    foreach i = 1 .. |PDES| do
        if PDES_i covers E then
            increment i-th descriptor's counter for negative examples;
    foreach i = 1 .. |ADES| do
        if ADES_i covers E then
            increment i-th descriptor's counter for negative examples;
            move i-th descriptor from ADES to PDES;
if learning window is full then
    delete the oldest example E_old from the learning window;
    if E_old is positive example then
        foreach i = 1 .. |ADES| do
            if ADES_i covers E_old then
                decrement i-th descriptor's counter for positive examples;
                if i-th descriptor's counter for positive examples == 0 then
                    delete i-th descriptor from ADES;
        foreach i = 1 .. |PDES| do
            if PDES_i covers E_old then
                decrement i-th descriptor's counter for positive examples;
                if i-th descriptor's counter for positive examples == 0 then
                    move i-th descriptor from PDES to NDES;
    else if E_old is negative example then
        foreach i = 1 .. |NDES| do
            if NDES_i covers E_old then
                decrement i-th descriptor's counter for negative examples;
                if i-th descriptor's counter for negative examples == 0 then
                    delete i-th descriptor from NDES;
        foreach i = 1 .. |PDES| do
            if PDES_i covers E_old then
                decrement i-th descriptor's counter for negative examples;
                if i-th descriptor's counter for negative examples == 0 then
                    move i-th descriptor from PDES to ADES;
Return ADES;PDES;NDES
```

Transitions among the description sets are shown in Figure 1.

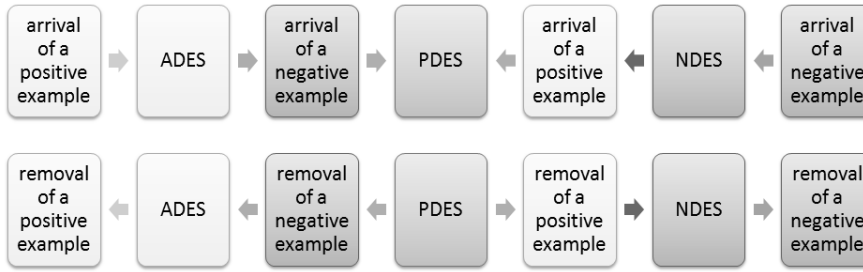

Figure 1: Transitions among the description sets

The ADES, NDES, and PDES sets are kept non-redundant and consistent with respect to the examples in the window. FLORA does not implement any specialization operator. If a new example cannot be covered by any description or generalization its full description is added to ADES or NDES, with respect to the type of example: positive or negative. The new incoming example acts as a specific seed, which may be generalized in the future. FLORA uses a generalization operator known as dropping condition rule, which removes attribute-value pairs from a single description item.

The simple FLORA framework assumes that only the latest fixed number of examples are relevant and should be kept in the window. However the question arises of how many examples are sufficient to describe current concepts. The authors expanded FLORA with a heuristic for flexible windowing in the FLORA2 algorithm. The motivation for this improvement were the effects of an inappropriate window size: too small a window will not contain a sufficient number of examples to describe a stable concept. On the other hand, too large a window will slow down reaction to a concept drift. A good heuristic for flexible windowing should shrink the window when a concept drift seems to occur and keep the window size fixed in case when concepts are stable. Meanwhile the window size should grow until concepts are stabilized. FLORA2's heuristic called Window Adjustment Heuristic (WAH) meets the above requirements. The pseudocode of WAH is presented as Algorithm 3. If a concept drift was detected, the WAH decreases the window size by 20% (lines 1–2). In case of extremely stable concepts the WAH decreases window size by 1 unit (lines 3–4). If the current concepts seems stable the window size remains unchanged (lines 5–6). In the other case, when the algorithm assumes that more examples is necessary, the window size is incresed by 1 unit (lines 7–8).

FLORA2 was tested on an artificial learning problem used by Schlimmer and Granger in [37]—STAGGER concepts. The example space is defined by three attributes: $size \in \{small, medium, large\}$, $color \in \{red, green, blue\}$, and $shape \in \{square, circle, triangle\}$. There also exists a sequence of three target concepts: (1) $size = small \wedge color = red$, (2) $color = green \wedge shape = circle$, and (3) $size = medium \vee size = large$. FLORA's authors randomly generated 120 training examples and labeled them according to some hidden context. After processing each example, the accuracy of the classification was tested on a separate testing set

with 100 examples. The concept was changed after every 40 examples. The obtained results showed that after a sudden change the total accuracy suddenly decreases but FLORA2 quickly adjusts to the new concepts and approaches 100% accuracy. WAH behaves as expected. Sudden change leads to a short increase in window size, followed by narrowing the window size and forgetting irrelevant examples.

Algorithm 3: Window Adjustment Heuristic

Input : N—number of positive examples covered by ADES;
S—number of descriptions in ADES;
Acc—current classification accuracy;
W—size of the learning window;
lc—threshold for low coverage of ADES;
hc—threshold for high coverage of ADES;
p—threshold for an acceptable accuracy of classification

Output: W'—new size of learning window

if $(\frac{N}{S} < lc)$ *or* $((Acc < p)$ *and* $(Acc$ *is decreasing*$))$ **then** /* drift */
  $W' = W - 20\% * W$;
else if $(\frac{N}{S} > 2 * hc)$ *and* $(Acc > p)$ **then** /* high stability */
  $W' = W - 1$;
else if $(\frac{N}{S} > hc)$ *and* $(Acc > p)$ **then** /* stability */
  $W' = W$;
else /* more information needed */
  $W' = W + 1$;
Return W'

Previous versions of FLORA irreversibly drop their old descriptions. However, there exists many natural domains, where a concept may reappear. In such a case, it would be a waste of time and effort to relearn an old concept from scratch. This was the reason for inventing the FLORA3 algorithm, which introduces a mechanism for a previous concept's storage and recall. The mechanism is tightly associated with the WAH heuristic. FLORA3 differs from FLORA2 behavior in a way that after every stage of learning it checks the current state of hypotheses in order to decide whether some old concept's descriptions are useful. The main idea assumes that when a change occurs the system should check which descriptions better explain the examples currently in the window: new concepts or some old ones. On the other hand, in case of stability periods it may be worth to store the current descriptions for future reuse. WAH decides when to store or reexamine the old concepts. If WAH signals a drift, the system examines its storage of old descriptions in order to find the one that fits the current state of the learning window. If one is found that is more appropriate than the current description, it replaces the current one. The procedure for reevaluating old concepts consists of three steps. First, the best candidate is found from all stored concepts that are consistent with the current examples in the window. It is the one with the highest ratio of positive to negative examples matched from the learning window. Then, the best candidate's counters are recalculated to reflect the

examples from the learning window. In the last step, the updated best candidate is compared with the current concept description on a measure of fitness. In FLORA3 the measure of fitness is estimated by the relative complexity of the descriptions—the more compact the ADES is, the better. To maintain the efficiency of the learning algorithm, the old concepts are not checked after every new training example. They are only retrieved when WAH suspects a concept drift. Moreover, the best candidate is determined by a simple heuristic measure. For more details see [44]. FLORA3 was tested on an artificial situation of recurring context. The dataset consisted of three STAGGER concepts repeated three times in cyclic order: 1–2–3–1–2–3–1–2–3. Training and testing examples were created using the same procedure as for FLORA2. Results showed that storing and reusing old concepts leads to a noticeable improvement in reaction time to the reappearing concepts. In most of the cases FLORA3 relearns faster and obtains higher accuracy levels than the simpler FLORA2.

Previous versions of FLORA deal with the main types of concept drift and recurring concepts. However they were not robust to noise. This is one of the difficulties in incremental learning—to distinguish between real concept drift and slight irregularities that may be treated as noise in the data. Methods that react quickly to every sign of change may overreact to noise. This may result in instability and low accuracy of classification. An ideal learner should combine stability and robustness with flexible and effective tracking of concept change [44]. That is why FLORA4 replaces the strict consistency condition, inherited from FLORA2 and FLORA3 with a softer notion of reliability. In FLORA4 for every description item statistical confidence intervals around its classification accuracy are calculated. Decisions when to move descriptions between sets are made based on the relation between these confidence intervals and observed class frequencies. Transitions among the description sets are as follows [44].

- A description item is kept in ADES if the lower endpoint of its accuracy confidence interval is greater than the class frequency interval's upper endpoint.
- A description item from ADES is moved to PDES, when its accuracy interval overlaps with the class frequency interval.
- A description item is dropped from ADES if the upper endpoint of its accuracy interval is lower than the class frequency interval's lower endpoint.
- Description items in NDES are kept as long as the lower endpoint of its accuracy confidence interval is greater than the class frequency interval's upper endpoint computed over negative examples in the window.
- There is no migration between NDES and PDES. Unacceptable hypotheses from NDES are deleted.

The main effect of this strategy is that generalizations in ADES and NDES may cover some negative or positive examples, respectively. PDES acts as a buffer for descriptions that cover too many negative examples or their absolute number of covered examples is too small. The rest of the algorithm's mechanisms remain unchanged. FLORA4 was also tested on STAGGER concepts and was compared with FLORA3

and FLORA2. In noise-free environment FLORA4 is initially a bit slower in reacting to a change than its predecessors. However, eventually it gains higher accuracy of classification faster than the previous versions. For a different amount of noise FLORA4 is again a bit slower in reaction to a change than the predecessors but then soon outperforms them. However, the difference in the classification accuracy is greater than for the noise-free data. FLORA4 was also compared with the IB3 algorithm. For more details see [44].

4 AQ11-PM+WAH

The FLORA framework memorizes only a window of the most recent learning examples. There also exist other algorithms that memorize the selected learning examples instead of the last examples for the future usage. This kind of memory is called a partial instance memory. Selected examples may be extreme, which means that they enforce, map, and strengthen boundaries of induced concept descriptions. For this reason they are the most relevant examples. That is why in each learning phase these examples are combined with the current learning examples to induce the most accurate set of decision rules. Thanks to this mechanism the new set of rules is well adjusted to the previous and current examples. This may also lead to less overtraining while learning from changing environments [31]. Maloof and Michalski proposed the AQ-PM system in [31]. They came up with a general algorithm for inductive learning with the partial instance memory presented as Algorithm 4.

Algorithm 4: General algorithm with partial memory

Input : S_i—current set of learning examples;
$previousRS$—previous set of rules;
$previousPM$—previous partial memory
Output: RS—a set of rules;
PM—partial memory

1 **if** $S_i \neq \emptyset$ **then**
2 $misclassified = FindMisclassifiedExamples(previousRS, S_i)$;
3 $currentTrainingExamples = previousPM \cup misclassified$;
4 $RS = InduceDecisionRules(currentTrainingExamples, previousRS)$;
5 $PM = SelectExtremeExamples(currentTrainingExamples, RS)$;
6 $PM = ForgetUnnecessaryExamples(PM, RS)$;
7 **Return** RS, PM

As an input the algorithm obtains a data set containing learning examples that may consist of both nominal and numerical attributes. There are no restrictions on the cardinality of the data set. Initially, the learner starts with an empty set of rules and an empty partial memory. In the first iteration (for the first set of examples), the learner operates as a batch classifier. Since the set of rules and partial memory are nonexistent, the current training set consists of all available examples (line 3). The

algorithm uses this set to induce a set of decision rules (line 4). Next, the system must choose examples to be stored in the partial memory (line 5). In the end, examples that are too old or do not enforce the concepts' boundaries are forgotten (line 6). In next iterations, the system checks which of the examples from current set S_i are misclassified (line 2). They are combined with the ones in the partial memory to form the current training set, from which hypotheses are induced (line 3). The rest of the steps are the same as for the first iteration.

The first proposal, called AQ-PM [31], uses the AQ covering algorithm to learn a set of decision rules. AQ randomly selects a positive training example, called "the seed". Then, the algorithm generalizes the seed as much as possible, with respect to the constraints from the negative examples and a single decision rule is induced. In the default mode, the algorithm uses a standard sequential covering mechanism—the positive examples from current learning set S_i covered by the rule are removed from the current training set and the whole process is repeated until all positive examples from S_i are covered. Decision rules obtained from AQ are complete and consistent—it means that the rules for given class cover all of the examples of the class and do not cover any differently labeled examples.

AQ-PM operates as follows. First, it finds misclassified training examples by classifying the new training set using a flexible matching strategy (line 2). These difficult examples are combined with the examples held in the partial memory (line 3). The newly created training set is then passed to the learning algorithm (line 4). AQ-PM operates in a temporal batch-mode. That means that the set of decision rules is induced with the static AQ algorithm from the new training set. Each AQ-PM decision rule describes a hyperrectangle in a discreet m-dimensional space, where m is the number of attributes [31]. Thanks to this notation, the extreme examples for the partial memory could be those that can be found on the surfaces, the edges or in the corners of the hyperrectangle covering them. Authors focused only on that examples which lie on the edges. In the next step of the AQ-PM algorithm, content of the partial memory is formed (line 5). The algorithm for finding extreme examples is presented as Algorithm 5.

Algorithm 5: Method for finding extreme examples

Input : S_i—current set of learning examples;
CR—characteristic rules
Output: S_{ee}—a set of extreme examples;

1 $S_{ee} = \emptyset$;
2 **foreach** $ruler \in CR$ **do**
3 $\quad minMaxRule = SelectMinMaxValuesForAttributesInRule(r)$;
4 $\quad$ **foreach** $selector \in minMaxRule$ **do**
5 $\quad\quad intervalRule = CreateIntervalSelectors(selector, minMaxRule)$;
6 $\quad\quad matchedExamples = PerformStrictMatching(S_i, intervalRule)$;
7 $\quad\quad S_{ee} = S_{ee} \cup matchedExamples$;
8 **Return** S_{ee}

In order to discover the extreme examples, AQ-PM uses characteristic rules. Those rules specify common features of examples belonging to the same class. These rules form the tightest possible hyperrectangle around examples from the same decision class. They consist of selectors for every conditional attribute. AQ-PM modifies the set of characteristic rules to match the examples that lie on the rules' boundaries. For each characteristic rule, the algorithm finds minimum and maximum values for each attribute, forgetting intermediate values (line 4). In this step, each selector of the characteristic rule is visited. Only the first and the last value existing in the characteristic rule are left—the intermediate attribute's values are removed. Then, each selector in such a specialized rule is modified to form an interval between selector's minimum and maximum values (line 5). Next, the new extreme examples are selected using the strict matching strategy (line 6). The transformed rule is applied on the current training set. The examples that match the edges of the transformed rule using the strict matching strategy are the extreme ones. In the end, current extreme examples are combined with previously obtained ones (line 7). AQ-PM is equipped in implicit forgetting—examples from partial memory are forgotten when no longer force a boundary.

AQ-PM was tested on three problems: STAGGER concepts, blasting cap detection and computer intrusion detection. The algorithm was compared with a simpler version of AQ-PM (baseline), with partial memory mechanism disabled, and IB2. The STAGGER concepts dataset consisted of 120 examples with sudden changes after every 40 examples. At each time step, a single training example and 100 testing examples were randomly generated. AQ-PM obtained higher results on total accuracy of classification than its opponents. The values of accuracy are comparable to those obtained by the FLORA system. The size of the memory held by AQ-PM was compared with the FLORA2's requirements. Over the entire learning phase, FLORA2 kept 15 examples, while AQ-PM maintained on average 6.6 examples in the partial memory. Blasting cap detection and computer intrusion detection was not evaluated by other researchers, so for the results and more details on these problems see [31].

AQ-PM was extended by combining the method for selecting extreme examples with the incremental learning system AQ11. The resulting AQ11-PM algorithm was described in [32]. The AQ11 learning system does not operate in batch mode but incrementally generates new rules from the existing rules and new training examples. The standard AQ11 algorithm has no instance memory. It reads each example only once and drops it after the learning phase. The AQ11's learning process consists of three main steps. In the first phase, the algorithm searches for difficult examples in the new training set—the ones that are misclassified. If a rule covers a new negative example, then in the second step, the rule is specialized to be consistent using the AQ11 covering algorithm. In the end, the specialized positive rule is combined with the new positive training examples and AQ is used to generalize them as much as possible without intersecting any of the negative rules and without covering any of the new negative examples. AQ11 uses this same procedure to learn rules incrementally for both the positive and negative class. Furthermore, this process can be adjusted to processing multiple classes. In this case, one class is selected and treated as the positive one, while other labels are treated as negative. The learning process is per-

formed on such partitions. This division is performed for each class present in the new training set. Because AQ11 has no instance memory, it relies solely on its current set of rules. Its rules are complete and consistent with respect to the current examples only. Like every incremental learner it can be susceptible to the ordering effect. This can be weakened using a partial instance memory. However, certain applications may require an additional mechanism to remove examples from the partial memory when they become too old.

AQ11-PM was also tested on three problems: STAGGER concepts, blasting cap detection and computer intrusion detection. For STAGGER concepts, the algorithm was compared with the unmodified version of AQ11 and AQ-PM. STAGGER concepts dataset was the same as the one created for the AQ-PM evaluation. At each time step, accuracy of classification and the number of examples in partial memory were recorded. AQ11-PM stores more examples than AQ-PM. However, it was able to achieve higher predictive accuracy on all the target concepts than its predecessor. AQ11-PM outperformed FLORA2 on accuracy of classification on the second and third context, but was weaker on the first one. Regarding memory requirements, both of the AQ family algorithms stored fewer examples during the evaluation than FLORA2. Blasting cap detection and computer intrusion detection was not evaluated by other researchers, so for the results and more details on these problems see [32].

The AQ11-PM algorithm was combined with FLORA's window adjustment heuristic (Algorithm 3) to adjust dynamically the window over which it retains and forgets examples. This mechanism will help to deal with changing concepts. The proposal was described in [33]. AQ11-PM+WAH was evaluated using STAGGER concepts. It was compared on total accuracy of classification and the number of maintained examples with AQ11, AQ11-PM and AQ-PM. The results suggest that the partial-memory classifiers learn faster than do simple incremental systems. AQ11-PM and AQ11-PM+WAH outperformed AQ-PM on all three concepts. Moreover, AQ-PM, AQ11-PM and AQ11-PM+WAH are competitive with FLORA2 in terms of predictive accuracy. In addition the AQ systems store fewer examples in the memory. For more details see [33].

5 FACIL

Previous solutions were not designed to process high-rate data streams. In this environment classifiers have to operate continuously, processing each item in real time only once. This forces memory and time limitations. Moreover, real data streams are susceptible to changes in contexts, so proposed methods should track and adapt to the underlying modifications. The new incremental algorithm—FACIL was proposed in [13]. FACIL is an acronym of the words Fast and Adaptive Classifier by Incremental Learning. It induces a set of decision rules from numerical data streams. This approach allows the rule to be inconsistent by storing positive and negative examples covered by it. Those examples lie very near one another—they are border examples. A rule is inconsistent when it covers both positive and negative examples. The aim of this system is to remember border examples until a minimum purity of the rule

is reached. The purity of the rule is defined as a ratio between number of positive examples covered by the rule to the total number of covered examples. When the value of purity falls below the minimum threshold, the examples associated with the rule are used to create new consistent rules. This approach is similar to AQ11-PM system, however it differs in the way that a rule stores two positive examples for a negative one. This guarantees that an impure rule is always modified from both positive and negative examples. Nevertheless, the examples held in memory are not necessary extreme. Despite the fact that this proposal suffers from the ordering effect, it does not weaken the learning process.

The initial proposal of FACIL operates on m numerical attributes. Every learning example is described by a normalized vector $[0,1]^m$ and a discrete value of a class label. Decision rule r is given by a set of m closed intervals $[I_{jl}, I_{ju}]$, where l stands for a lower bound, and u—upper bound [13]. Rules are separated among different sets according to the appropriate class label. FACIL does not maintain any global window but each rule has a different set of associated examples. Each rule has its own window of border examples. Each rule stores a number of positive and negative examples and also an index of the last covered example. The model is updated every time a new example becomes available. The pseudocode of FACIL is presented as Algorithm 6.

FACIL operates as follows. When a new example arrives, the rules associated with the example's class label are checked (lines 1–9) and the generalization necessary to describe the new example is calculated according to the formula (line 2):

$$Growth(r, x_i) \;=\; \sum_{j=1}^{m} (g_j - r_j), \tag{5.1}$$

where $g_j = max(x_{ij}; I_{ju}) - min(x_{ij}; I_{jl})$ and $r_j = I_{ju} - I_{jl}$. The measure of growth favors the rule that involves the smallest changes in the minimum number of attributes. A rule with the minimum value of growth becomes a candidate (lines 3–4). However, the rule is taken into account as a possible candidate only if the new example can be seized with a moderate growth (lines 3–4). It occurs when $\forall\ j \in \{1..m\} : g_j - r_j \leq \kappa$, where $\kappa \in (0; 1]$ [13]. If the first rule covering the new example is found, then the number of positive examples covered by the rule is increased and the rule's last-covered-example index is updated (lines 5–7). The example is added to the rule's window, if the number of negative examples covered by the rule increased by one unit (lines 8–9). If any of rules associated with the example's class label does not fire for the example (line 10), the rest of the rules with different class labels are visited (lines 11–21). If a rule with a different label does not cover the example, the intersection with the candidate is checked (line 21). If there exists such an intersection, the candidate is rejected (line 24). When the different-labeled rule covers the example (line 12), its negative support is increased (line 13). Additionally, the example is added to the rule's window of examples (line 14). If the purity of the rule dropped below the minimum value given by the user (line 15), new consistent rules are created from examples associated with the initial rule and added to the model (lines 16–17). The old rule is marked as unreliable (line 18) and cannot be used in the generalization process, even for rules with different labels. A window

Algorithm 6: FACIL algorithm

Input : e—a new learning example;
p_{min}—a minimum purity of a rule;
κ—a moderate growth threshold;
RS—current set of rules
Output: RS'—modified set of rules

foreach *rule* r_p *with same class label as example* e **do**
  G = compute $Growth(r_p, e)$;
  if *(*G *is minimum) and (*G *is moderate)* **then**
    candidate = rule r_p;
  if *rule* r_p *covers example* e **then**
    increase positive support of rule r_p;
    update rule's r_p index of last covered example;
    if *rule's* r_p *negative support has increased* **then**
      add example e to rule's r_p window of examples;
if *example* e *is not covered by a rule from positive class label* **then**
  foreach *rule* r_n *with different class label as example* e **do**
    if *example* e *is covered by a rule* r_n *from negative class label* **then**
      update negative support of the rule r_n;
      add example e to the negative rule's r_n window;
      if *purity of the rule* $r_n < p_{min}$ **then**
        R_c = induce new consistent rules from r_n;
        $RS' = RS \cup R_c$;
        mark rule r_n as unreliable;
        reset rule's r_n window of examples;
    else
      intersection = calculate intersection of rule r_n with the candidate;
if *example* e *is not covered by a rule from negative class label* **then**
  if $intersection \neq \emptyset$ **then**
    reject candidate;
    R_e = generate maximally specific rule to describe example e;
    $RS' = RS \cup R_e$;
  else if $intersection = \emptyset$ **then**
    G = generalize candidate with respect to example e;
    $RS' = RS \cup G$;
remove unnecessary rules from RS';
update window with learning examples for each rule in RS';
Return RS'

of examples connected with the unreliable rule is reset (line 19). Afterwards, if the intersection of the candidate with different labeled rules is empty, then the candidate rule is generalized (lines 27–29). If there exists no rule that covers the example and there is no candidate for generalization, then a maximally specific rule to describe the new example is added to the appropriate set of rules (lines 22–26). Rules can also be deleted from the appropriate sets (line 30). A rule is removed if it is unreliable with a support smaller than the support of any rule generated from it. The second condition for rule removal is when the number of times the rule prevented generalization of a different label rule is greater than its support. FACIL is also equipped in a forgetting mechanism for dropping learning examples (line 31). This mechanism can be either explicit or implicit. Examples which are older than a user's defined threshold are deleted—this is explicit forgetting. Implicit forgetting takes place when examples are no longer relevant—they no longer lie on any of the rules boundary.

Like every rule-based classifier, FACIL is supplemented with a classification strategy. A new test example is classified by rules that cover it. Unreliable rules that cover the example are rejected. Reliable rules are used to classify the test example. Consistent rules classify new examples by strict matching. Inconsistent rules acts like the nearest neighbor algorithm and classify the new example by its distance. The authors do not explain how exactly it is performed. Probably, they calculate the Euclidean distance between the test example and the rule's boundaries. In the case when no rule covers the example, it is classified to the label associated with the reliable rule with the minimal value of growth and an empty intersection with any other different label rules.

The initial version of FACIL was evaluated on 12 real datasets from the UCI repository[1] and on a synthetic data stream generated from a moving hyperplane. In case of real data, a concept drift is not present. During the experiments the total accuracy of classification, the learning time and the number of induced rules was recorded. FACIL was compared with the C4.5Rules algorithm. In half of the real problems, FACIL obtains better results on the classification accuracy. Because FACIL is a single-pass solution, the processing time is always significantly shorter than for multi-pass C4.5Rules. For the hyperplane data stream, authors evaluated the computational cost as a function of the number of attributes. FACIL was not compared with any other existing stream mining solution. For detailed results see [13].

The initial version of FACIL was extended to process symbolic attributes in [14]. The formula for calculating the growth of a rule was changed in a way to process nominal attributes: $Growth(r,x) = \sum_{j=1}^{m} \Delta(\mathcal{T}_j, x_j)$, where for numeric attributes: $\Delta(\mathcal{T}_j, x_j) = min(|I_{jl} - x_j|; |x_j - I_{ju}|)$ and for nominal attributes: if example's attribute value x_i is covered by the rule then $\Delta(\mathcal{T}_j, x_j) = 0$, in the opposite case—$\Delta(\mathcal{T}_j, x_j) = \frac{1}{|Domain(attribute_j)|}$.

The extension of FACIL was tested on a moving hyperplane problem. Again, the authors focused on evaluating the computational cost as a function of the number of attributes. The total accuracy of classification drops with the number of attributes. The processing time increases with the growth of the hyperplane problem. For detailed results see [14].

[1] see http://archive.ics.uci.edu/ml/

6 VFDR

The Very Fast Decision Rules (VFDR) algorithm proposed by Gama and Kosina in [19] was also designed for high-speed massive data streams. It reads every learning example only once and induces an ordered or an unordered list of rules. VFDR enables processing both nominal and numeric attributes. The algorithm starts with an empty rule set RS and an empty default rule $\{\} \rightarrow \mathcal{L}$. $\mathcal{L}$ is a data structure that contains the necessary information for classification of new examples and includes the statistics used for extending the rule. Each rule r is associated with the corresponding data structure $\mathcal{L}_r$. Every $\mathcal{L}_r$ (also $\mathcal{L}$) stores: the number of examples covered by rule r, a vector to calculate the probability of observing examples of class c_i, a matrix to calculate the probability of observing value v_i of a nominal attribute at_i per class and a b-tree to compute the probability per class of observing values greater than v_j for a numerical attribute at_i [19]. In general, $\mathcal{L}_r$ accumulates sufficient statistics to compute the entropy for every label of a decision class. $\mathcal{L}_r$ is updated when its corresponding rule covers a labeled example. The pseudocode of VFDR is presented as Algorithm 7.

VFDR operates as follows. When a new learning example e is available all decision rules are visited (lines 1–15). If rule r covers example e (line 2), its corresponding structure $\mathcal{L}_r$ is updated (line 3). The Hoeffding bound states the number of examples after which a rule set RS should be updated either by extending some existing rule or inducting a new rule (line 4). The Hoeffding bound guarantees that with the probability $1 - \delta$ the true mean of a random variable x with a range R will not differ from the estimated mean after n independent observations by more than $\epsilon = \sqrt{\frac{R^2 * ln(\frac{1}{\delta})}{2*n}}$ [19]. In the next step, the initial value of the entropy is calculated from the statistics gathered in $\mathcal{L}_r$ (line 5). If the value of entropy exceeded the Hoeffding bound, then a rule should be enhanced (line 7). The rule is extended as follows. For each attribute and for each of this attribute's values that were observed in more than 10% of examples, the value of the split evaluation function is computed (lines 8–13). If the value of the interesting measure for the best split is better than for not splitting, the rule is extended with a new selector obtained from the best split (lines 12–13). The selector that minimizes the entropy of the class labels of the examples covered by the rule is added to the previous elementary conditions of the rule. The class label of the rule is then assigned according to the majority class of observations. VFDR can learn an ordered or unordered set of decision rules. In the former case, every labeled example updates the statistics of the first rule that covers it (line 14–15). For the latter—every rule that covers the example is updated. Those sets of rules are learned in parallel. In case when none of the rules cover example e (line 16), the default rule is updated (line 17). Then, if the number of examples in $\mathcal{L}$ exceeds the minimum number of examples obtained from the Hoeffding bound, new decision rules are induced from the default rule—using the same mechanism of a rule's growth as described earlier (lines 18–19).

VFDR, as every rule-based classifier, is equipped with a classification strategy. The simplest strategy uses the stored distribution of classes—an example is classified to the class with the maximum value of probability. A more sophisticated strategy bases

Algorithm 7: Very Fast Decision Rules algorithm

Input : e—a new learning example;
RS—current set of rules;
ordered—flag indicating induction of ordered set of rules;
S_{min}—a minimum number of examples from Hoeffding bound;
δ—threshold for probability used in Hoeffding bound;
SEF—a split evaluation function;
Output: RS'—modified set of decision rules

```
foreach rule r ∈ RS do
    if r covers e then
        update statistics in L_r;
        if number of examples in L_r > S_min then
            ent_0 = calculate the entropy from L_r;
            compute Hoeffding bound;
            if ent_0 > Hoeffding bound then
                foreach attribute at_i do
                    ent_ij - the best split from SEF on attribute at_i and value v_j;
                    if (ent_ij < ent_best) and (v_j observed > 10% examples) then
                        ent_best = ent_ij;
                if (ent_0 − ent_best) > Hoeffding bound then
                    r = r ∪ {at_i = v_j};
        if ordered == true then
            break;
if none of the rules covers e then
    update statistics L of the default rule;
    if number of examples in L > S_min then
        RS' = RS ∪ a rule induced from default rule;
Return RS'
```

on the Bayes rule with the assumption of attribute independence for the class. The Naive Bayes strategy uses the prior distribution of the classes and also the conditional probabilities of the attribute-value pairs given the class. As a result, for each testing example $e = (v_1, ..., v_j)$, the probability that example e belongs to decision class c_k is $P(c_k|e) \propto P(c_k) \prod jP(v_j|c_k)$ [19]. Thanks to using this strategy more information available with each rule is exploited. An example is classified to the class with the maximum value of the posteriori probability. In case of the ordered set of rules only the first rule that covers an example is fired. Using the unordered set of rules—results returned by all rules that match the example are combined using weighted voting. This type of voting assumes that not all voters are equal. Instead, they are diversified by giving them different amounts of weights. The authors did not provide information on how weights are assigned to each of the decision rules.

VFDR was tested on six different data streams: disjunctive concepts, hyperplane, LED, SEA, STAGGER, and Waveform. The authors tested two different classification strategies. Usage of the Bayes theorem improves the predictive capabilities of the algorithm. Authors also compared an ordered versus an unordered set of rules. The experimental evaluation showed that unordered rule set is more competitive than the ordered one with respect to the accuracy of classification. In the end, VFDR (with the Bayes classification strategy and an unordered set of rules) was compared with VFDT and C4.5Rules. VFDR is much more efficient than C4.5Rules in terms of memory and processing time. It also obtained competitive results against VFDT. For more details see [19].

The initial version of VFDR was extended to deal with multi-class problems in [26]. The proposed algorithm VFDR-MC decomposes a multi-class problem into a set of two-class problems and induces a set of discriminative rules for each binary problem. VFDR-MC applies one versus all strategy in which examples of one class are positive and other are negative. It considers a rule expansion for each of the classes observed with current rule. The expansion of a rule is different for the default rule and for the already existing rule. It also depends on the type of generated rule set: ordered or unordered. The default rule is expanded to a new rule with a literal for which a gain function, adopted from FOIL classifier, obtains the best value. The rule's decision class is indicated by the class with minimum frequency among those that satisfy the Hoeffding bound condition. The ordered strategy stops after finding the first candidate rule that is better than the previous one. The unordered strategy checks all possible expansions for every decision class. In case of extending a rule that already exists, the procedure also depends whether the ordered or unordered set of decision rules is induced. In case of ordered set only literals for positive class are tested. For unordered set the decision rules the class of expanded rule is maintained as positive for the first calculations of the gain measure. Next, computations for other classes set as positive ones are performed. This allows to produce more than one rule but not always for all the available decision classes. For more details see [26].

VFDR-MC was tested on six different data streams: KDDCup99, covtype, hyperplane, SEA, LED, and Random Tree. The authors observed that the unordered version obtained generally better results of classification accuracy than the ordered one. Moreover, unordered VFDR-MC mostly outperforms base version of VFDR on

multi-class data sets. Learning time for ordered rule set is almost the same as in case of creation of the Hoeffding Tree. In case of unordered set of decision rules the learning time grows with the number of rules. For more details see [26].

VFDR was also improved in order to handle time changing data. The resulting algorithm Adaptive Very Fast Decision Rules (AVFDR) was described in [27]. AVFDR extends VFDR-MC with explicit drift detection. Each rule in the set of decision rules is equipped in a drift detection method, which tracks the performance of the rule during learning. Applied drift detector is presented as Algorithm 8. For every learning example covered by the rule, the rule updates its error of classification. Moreover, the drift detector manages two additional statistics: $error_{min}$ and $stddev_{min}$. Those registers are updated if for given learning example e $error_e + stddev_e < error_{min} + stddev_{min}$. The flag indicating type of change for given rule can take one of three values: None, Warning or Drift. If the rule achieved warning level, then the rule's learning process is stopped until the flag is set to None again. In case of Drift level, the rule is so weak that it is removed from the set of decision rules. This helps to keep the final set of decision rules effective and up-to-date. For more details see [27].

Algorithm 8: AVFDR Drift Detection Method

Input : r—tested decision rule;
e—current learning example;
Output: $flag \in \{None, Warning, Drift\}$—flag indicating type of change

```
flag = None;
compute error of classification error_e for given learning example e with its
standard deviation stddev_e.;
if (error_e + stddev_e) < (error_min + stddev_min) then
    error_min = error_e;
    stddev_min = stddev_e;
if (error_e + stddev_e) ≥ (error_min + 3 * stddev_min) then
    flag = Drift;
else if (error_e + stddev_e) ≥ (error_min + 2 * stddev_min) then
    flag = Warning;
Return flag
```

AVFDR was tested on five artificial data streams: Hyperplane, SEA, LED, RBF, and Waveform and six real datasets: KDDCup99, Covtype, Elec, Airlines, Connect-4, and Activity. The results obtained on artificial data show that AVFDR works best for changing environments. The accuracy of classification of VFDR's base version decreases with time. In case of real datasets, AVFDR^u obtains competitive results on the accuracy of classification with a lower size of the induced model. For more details see [27].

7 Conclusions

Mining data streams recently became a very popular topic of research. Data streams are susceptible to changes in the hidden context, producing what is generally known as concept drift. There exist two main types of concept drift: sudden and gradual. However, there are also other types like recurring context and two cases, to which a good classifier should be resistant: blips and noise. Learning from non-stationary environments is rather a new discipline, but there already exist algorithms that attempt to solve this problem. They can be divided into two main groups: trigger-based and evolving methods. In this paper four key rule-based online algorithms proposed for mining data streams in the presence of concept drift were presented. First, FLORA was described—a first family of algorithms that flexibly react to changes in concepts, can use previous hypotheses in situations when context reappears and are robust to noise in data [44]. Then, algorithms from the AQ family were presented with their modifications. AQ-PM [31] is a static learner that selects extreme examples from rules' boundaries for each incoming batch of data and stores them in the partial memory. AQ11-PM [32] is a combination of the incremental AQ11 algorithm with a partial memory mechanism. AQ11-PM+WAH [33] is extended with a heuristic for a flexible size of the window with stored examples. The FACIL algorithm operates similarly to AQ11-PM [13]. However, it differs in the way that examples stored in the partial memory do not have to be extreme ones. Those three main algorithms were not tested on huge datasets. For massive high-speed data streams a new algorithm called VFDR was proposed in [19]. It induces an ordered or an unordered sets of decision rules that are efficient in terms of memory and processing time.

Those solutions use the same representation of knowledge—decision rules, however they operate in a differet way. These four algorithms can be compared on several criteria, like the type of data. FLORA is restricted only to nominal attributes, where AQ11-PM+WAH, FACIL and VFDR process both nominal and numerical attributes. On the other hand, FLORA, AQ11-PM+WAH and FACIL are adjusted to deal with concept drift, where VFDR are suitable only to stationary environments. Moreover, FLORA was designed and tested on different types of concept drift: sudden, recurring, and noise. Unfortunately, the first three solutions were not tested on massive data streams with concept drift. Two of them (FLORA and AQ11-PM+WAH) were tested on STAGGER concepts with 120 learning examples, where FACIL was evaluated on the moving hyperplane problem. FLORA and AQ11-PM+WAH solve binary classification problem, but the latter one can be extended for the multi-class problem. FACIL and VFDR do not have any restrictions on the number of decision classes. The four proposals differ also on the type of memory that they maintain. FLORA remembers only a window of the most recent examples. AQ11-PM+WAH has a partial memory with extreme examples that lie on the boundaries of induced decision rules. Additionally, application of WAH heuristic introduced a global learning window, outside which old examples are forgotten. FACIL also maintains a partial memory but the stored examples do not have to be extreme ones. Every decision rule has its own window of learning examples. Moreover, it remembers more examples than its predecessor (it stores two positive per one negative example). On the other

hand, VFDR has no instance memory—it only maintains a set of decision rules with their corresponding data structures $\mathcal{L}_r$ containing all necessary statistics. Knowledge representation is also maintained in a different way. FLORA stores the conditional part of rules in three description sets: ADES, PDES, and NDES. AQ11-PM+WAH induces a classical unordered set of decision rules. In case of FACIL, rules consist of all conditional attributes, which define an m-dimensional space (intervals). VFDR is the only algorithm that can induce either an unordered or an ordered set of decision rules. Its rules have to be as short as possible. Another criterion that differs the four described algorithms is the way of use of induced decision rules for new examples' classification. Moreover, all algorithms were evaluated in different setups and on different data sets, so the obtained results cannot be compared with each other.

It is difficult to state which of the described algorithms is the best. They were introduced in different times and were tested on different data sets. It would be interesting to perform a comparison of those solutions on many data streams containing different types of concept drift with respect to the total accuracy of classification, the memory usage and the processing time. Nowadays the MOA environment—a framework for data stream mining, is very helpful. It contains a collection of machine learning algorithms, data generators and tools for evaluation. More can be found about this project in the literature [4] and on the MOA project website[2]. MOA can be easily extended with new mining algorithms, but also with new stream generators or evaluation measures. Unfortunately the implementations of FLORA, AQ11-PM+WAH, FACIL, and VDFR are not publicly available, hindering such a comparison at present.

References

[1] An A., Learning Classification Rules from Data, *Computers and Mathematics with Applications*, vol. 45, p. 737-748, 2003.

[2] Baena-Garcia M., Del Campo-Avila J., Fidalgo R., Bifet A., Early Drift Detection Method, *Proceedings of the 4th ECML PKDD International Workshop on Knowledge Discovery from Data Streams*, p. 77-86, Berlin, Germany, 2006.

[3] Bakker J., Pechenizkiy M., Food Wholesales Prediction: What is Your Baseline?, *Proceedings of the 18th Symposium on Methodologies for Intelligent Systems*, ISMIS 2009, Prague, Czech Republic, LNCS, vol. 5722, p. 493-502, 2009.

[4] Bifet A., Holmes G., Pfahringer B., Kranen P., Kremer H., Jansen T., Seidl T.: MOA: Massive Online Analysis a Framework for Stream Classification and Clustering, *Workshop on Applications of Pattern Analysis*, HaCDAIS, 2010.

[5] Błaszczyński J., Stefanowski J., Zając M., Ensembles of Abstaining Classifiers Based on Rule Sets, *Proceedings of the 18th International Symposium on Methodologies for Intelligent Systems*, ISMIS 2009, Prague, Czech Republic, LNCS, vol. 5722, p. 382-391, 2009.

[2] see: http://moa.cs.waikato.ac.nz/

[6] Cendrowska J., PRISM An Algorithm for Inducing Modular Rules, *International Journal Man-Machine Studies*, vol. 27, p. 349-370, 1987.

[7] Cestnik B., Estimating Probabilities: A Crucial Task in Machine Learning, *Proceedings ECAO 1990*, Stockholm, Sweden, 1990.

[8] Clark P, Boswell R., Rule Induction with CN2: some recent improvement, *Proceedings of 5th European Working Session on Learning*, ESWL 1991, Porto, Portugal, p. 151-163, 1991.

[9] Clark P., Niblett T., The CN2 Induction Algorithm, *Machine Learning*, vol. 3, p. 261-283, 1989.

[10] Deckert M., Batch Weighted Ensemble for Mining Data Streams with Concept Drift, *Proceedings of the 19th International Symposium on Methodologies for Intelligent Systems*, ISMIS 2011, Warsaw, Poland, LNCS, vol. 6804, p. 290-299, 2011.

[11] Deckert M., Stefanowski J., Comparing Block Ensembles for Data Streams with Concept Drift, *Proc. of Workshop Mining Complex and Stream Data, ADBIS 2012*, Poznań, Poland, AISC, vol. 185, p. 69-78, 2012.

[12] Domingos P., Hulten G., Mining High-Speed Data Streams, *Proceedings of the KDD 2000*, ACM Press, p. 71-80, 2000.

[13] Ferrer-Troyano F.J., Aguilar-Ruiz J.A., Riquelme J.C., Incremental Rule Learning and Border Examples Selection from Numerical Data Streams, *Journal of Universal Computer Science*, vol. 11(8), p. 1426-1439, 2005.

[14] Ferrer-Troyano F.J., Aguilar-Ruiz J.A., Riquelme J.C., Data Streams Classification by Incremental Rule Learning with Parametrized Generalization, *Proceedings of ACM Symposium on Applied Computing 2006*, SAC 2006, p. 657-661, ACM, 2006.

[15] Fürnkranz J., Separate-and-Conquer Rule Learning, *Artificial Intelligence Review*, vol. 13, p.3-54, 1999.

[16] Fürnkranz J., Gamberger D., Lavrač N., Foundations of Rule Learning, *Cognitive Technologies*, 2012.

[17] Gama J., Medas P., Castillo G., Rodrigues P., Learning with Drift Detection, *Proceedings of Brazilian Symposium on Artificial Intelligence, SBIA 2004*, LNAI, vol. 3171, p. 286-295, Springer-Verlag, 2004.

[18] Gama J., *Knowledge Discovery from Data Streams*, Chapman and Hall/CRC 2010.

[19] Gama J., Kosina P., Learning Decision Rules from Data Streams, *Proceedings of 22th International Joint Conference on Artificial Intelligence*, IJCAI 11, vol. 2, p. 1255-1260, AAAI Press, 2011.

[20] Giraud-Carrier C., A Note on the Utility of Incremental Learning, *AI Communications*, vol. 13, p. 215-223, 2000.

[21] Greco S., Słowiński R., Stefanowski J., Żurawski M., Incremental versus Non-incremental Rule Induction for Multicriteria Classification, *Transactions on Rough Sets II*, LNCS, vol. 3135, p. 33-53, 2004.

[22] Grzymala-Busse J.W., LERS - A System for Learning from Examples Based on Rough Sets, *Intelligent Decision Support. Handbook of Applications and Advances of the Rough Sets Theory*, p. 3-18, 1992.

[23] Grzymala-Busse J.W., Selected Algorithms of Machine Learning from Examples, *Fundamenta Informaticae*, vol. 18, p. 193-207, 1993.

[24] Grzymala-Busse J.W., Managing Uncertainty in Machine Learning from Examples. *Proceedings of 3rd International Symposium in Intelligent Systems*, p. 70-84, 1994.

[25] Hulten G., Spencer L., Domingos P., Mining Time-changing Data Streams, *Proceedings of the KDD 2001*, ACM Press, p. 97-106, 2001.

[26] Kosina P., Gama J., Very Fast Decision Rules for Multi-class Problems, *Proceedings of the 2012 ACM Symposium on Applied Computing*, New York, USA, p. 795-800, 2012.

[27] Kosina P., Gama J., Handling Time Changing Data with Adaptive Very Fast Decision Rules, *Proceedings of the 2012 European conference on Machine Learning and Knowledge Discovery in Databases*, ECML/PKDD 2012, Bristol, United Kingdom, vol. 1, p. 827-842, 2012.

[28] Kuncheva L. I., Classifier Ensembles for Changing Environments, *Proceedings of 5th International Workshop on Multiple Classifier Systems, MCS 04*, LNCS, vol. 3077, p. 1-15, Springer-Verlag, 2004.

[29] Kuncheva L. I., Classifier Ensembles for Detecting Concept Change in Streaming Data: Overview and Perspectives, *Proceedings 2nd Workshop SUEMA 2008*, ECAI 2008, p. 5-10, Patras, Greece, 2008.

[30] Maison R., Zakrzewicz M., Content-based Load Shedding in Multimedia Data Stream Management System, *Foundations of Computing and Decision Sciences*, vol. 37(2), p. 79-95, 2012.

[31] Maloof M., Michalski R., Selecting Examples for Partial Memory Learning, *Machine Learning*, vol. 41, p. 27-52, Kluwer Academic Publishers, 2000.

[32] Maloof M., Michalski R., Incremental Learning with Partial Instance Memory, *Artificial Intelligence*, vol. 154, p. 95-126, Elsevier, 2003.

[33] Maloof M., Incremental Rule Learning with Partial Instance Memory for Changing Concepts, *Proceedings of the International Joint Conference on Neural Networks 2003*, IJCNN-03, vol. 4, p. 2764-2769, IEEE Press, 2003.

[34] Michalski R.S., A Theory and Methodology of Inductive Learning, *Machine Learning: An Artificial Intelligence Approach*, p. 83-134, 1983.

[35] Michalski R.S., Mozetic I., Hong J., Lavrac N., The AQ15 Inductive Learning System: An Overview and Experiments, Report 1260, Department of Computer Science, University of Illinois, 1986.

[36] Nishida K., Yamauchi K., Omori T., ACE: Adaptive Classifiers-Ensemble System for Concept-Drifting Environments, *Multiple Classifier Systems*, LNCS, vol. 3541, p. 176-185, 2005.

[37] Schlimmer J., Granger R., Incremental Learning from Noisy Data, *Machine Learning*, vol. 1(3), p. 317-357, 1986.

[38] Shannon C.E., A Mathematical Theory of Communication, *Bell System Technical Journal*, vol. 27(3), p. 379-423, 1948.

[39] Stefanowski J., The Rough Set Based Rule Induction Technique for Classification Problems. *Proceedings of the 6th European Conference on Intelligent Techniques and Soft Computing*, EUFIT-98, p. 109-113, 1998.

[40] Stefanowski J., Algorytmy Indukcji Reguł Decyzyjnych w Odkrywaniu Wiedzy [in Polish], Habilitation thesis, Rozprawy series, vol. 361, Poznań University of Technology, 2001.

[41] Sulzmann J.N., Fürnkranz J., A Study of Probability Estimation Techniques for Rule Learning, From Local Patterns to Global Models. *Proceedings of the ECML/PKDD 2009 Workshop*, p. 123-138, 2009.

[42] Tsymbal A., The Problem of Concept Drift: Definitions and Related Work, Technical Report, Department of Computer Science, Trinity College Dublin, Ireland, 2004.

[43] Wang H., Fan W., Yu P.S. and Han J., Mining Concept-drifting Data Streams Using Ensemble Classifiers, *Proceedings ACM SIGKDD*, p. 226-235, 2003.

[44] Widmer G., Kubat M., Learning in the Presence of Concept Drift and Hidden Contexts, *Machine Learning*, vol. 23, p. 69-101, 1996.

[45] Zliobaite I., Learning Under Concept Drift: An Overview, Technical Report, Faculty of Mathematics and Informatics, Vilnius University, Vilnius, Lithuania, 2009.

[46] Zliobaite I., Bakker J., Pechenizkiy M., OMFP: An Approach for Online Mass Flow Prediction in CFB Boilers, *Discovery Science*, p. 272-286, 2009.

[47] Zliobaite I., Bakker J., Pechenizkiy M., Towards Context Aware Food Sales Prediction. In *Proceedings of the 3nd International Workshop on Domain Driven Data Mining (DDDM'09)*, IEEE International Conference on Data Mining ICDM'09, Miami, Florida, USA, p. 94-99, 2009.

Received October, 2012