Chapter 9

A Survey of Stream Classification Algorithms

Charu C. Aggarwal
IBM T. J. Watson Research Center
Yorktown Heights, NY 10598
charu@us.ibm.com

9.1 Introduction

Advances in hardware technology have led to the increasing popularity of data streams [1]. Many simple operations of everyday life, such as using a credit card or the phone, often lead to automated creation of data. Since these operations often scale over large numbers of participants, they lead to massive data streams. Similarly, telecommunications and social networks often contain large amounts of network or text data streams. The problem of learning from such data streams presents unprecedented challenges, especially in resource-constrained scenarios.

An important problem in the streaming scenario is that of data classification. In this problem, the data instances are associated with labels, and it is desirable to determine the labels on the test instances. Typically, it is assumed that both the training data and the test data may arrive in the form of a stream. In the most general case, the two types of instances are mixed with one another. Since the test instances are classified independently of one another, it is usually not difficult to perform the real-time classification of the test stream. The problems arise because of the fact that the training model is always based on the aggregate properties of multiple records. These aggregate properties may change over time. This phenomenon is referred to as *concept drift*. All streaming models need to account for concept drift in the model construction process. Therefore, the construction of a training model in a streaming and evolving scenarios can often be very challenging.

Aside from the issue of concept drift, many other issues arise in data stream classification, which are specific to the problem at hand. These issues are dependent on the nature of the application in which streaming classification is used. A single approach may not work well in all scenarios. This diversity in problem scenarios is discussed below:

- The classification may need to be performed in a resource-adaptive way. In particular, the stream needs to be classified *on demand*, since it may not be possible to control the rate at which test instances arrive.

- In many scenarios, some of the classes may be rare, and may arrive only occasionally in the data stream. In such cases, the classification of the stream becomes extremely challenging because of the fact that it is often more important to detect the rare class rather than the normal class. This is typical of cost sensitive scenarios. Furthermore, in some cases, previously unseen classes may be mixed with classes for which training data is available.

- Streaming classifiers can be designed to provide either *incremental* or *decremental* learning. In incremental learning, only the impact of incoming data points is accounted for. In decremental sampling, a sliding window model is used, and expiring items at the other end of the window are accounted for, the model update process. Not all classes of models can be used effectively for decremental learning, though incremental learning is usually much easier to address.

- In the context of discrete (categorical) attributes, the massive domain case is particularly important. In these cases, the attributes are drawn on massive domain of categorical values. For example, if the IP-address is an attribute, then the number of possible values may be in the millions. Therefore, it often becomes difficult to store the relevant summary information for classification purposes.

- Many other data domains such as text [6] and graphs [9] have been studied in the context of classification. Such scenarios may require dedicated techniques, because of the difference in the underlying data format.

- In many cases, the entire stream may not be available at a single processor or location. In such cases, distributed mining of data streams becomes extremely important.

Clearly, different scenarios and data domains present different challenges for the stream classification process. This chapter will discuss the different types of stream classification algorithms that are commonly used in the literature. We will also study different data-centric scenarios, corresponding to different domains or levels of difficulty.

This chapter is organized as follows. The next section presents a number of general algorithms for classification of quantitative data. This includes methods such as decision trees, nearest neighbor classifiers, and ensemble methods. Most of these algorithms have been developed for quantitative data, but can easily be extended to the categorical scenario. Section 9.3 discusses the problem of rare class classification in data streams. In Section 9.4, we will study the problem of massive domain stream classification. Different data domains such as text and graphs are studied in Section 9.5. Section 9.6 contains the conclusions and summary.

9.2 Generic Stream Classification Algorithms

In this section, a variety of generic stream classification algorithms will be studied. This represents the most general case, in which the stream is presented as an incoming stream of multidimensional records. In particular, the extension of different kinds of classifiers such as rule-based methods, decision trees, and nearest neighbor methods to the streaming scenario will be studied. Ensemble methods will also be discussed, which have the advantage of robustness in the context of data streams. It should be pointed out that many of these methods use sampling in order to improve the classification accuracy.

Reservoir sampling [86] is a generic technique, which can be used in order to improve the effectiveness of many stream classifiers. The idea is to maintain a continuous sample of the training data, of modest size. At any given time, a learning algorithm can be applied to this sample in order to create the model. This is, of course, a general meta-algorithm, and it does have the disadvantage that the accuracy of the approach is limited by the size of the sample. Nevertheless, its merit is that it can be used in conjunction with any offline classifier of arbitrary complexity, as long as a reasonable training sample size is used for analysis. Such algorithms can handle concept drift, since reservoir methods for creating time-decayed samples [12] are available. When a learning algorithm is applied to such a model, it will be well adjusted to the changes in the underlying data distribution, because of the time-decay bias of the sample. Since reservoirs can also be collected over sliding windows, such methods can also implicitly support decremental learning.

9.2.1 Decision Trees for Data Streams

While many scalable decision tree methods such as *SLIQ* [68], *RainForest* [44], and *BOAT* [43] have been proposed earlier, these methods are not designed for the streaming scenario.

While relatively little known, a large number of classical methods were designed for incremental decision tree induction, on the basis of the ID3 family in the classical literature [49, 83–85]. Most of these methods preceded the advent of the data stream paradigm, and are therefore not well known, at least within the streaming context. One of the earliest methods based on Quinlan's ID3 algorithm was designed by Schlimmer [78] in 1986. Other methods were designed for multivariate induction as well [84, 85], as early as 1990 and 1994, respectively. Kalles [49] proposed one of the most efficient versions by estimating the minimum number of training items for a new attribute to be selected as a split attribute. These methods were generally understood as purely incremental methods, though many of them do have the potential to be applied to data streams. The major downside of these methods is that they are typically not designed to handle concept-drift in the underlying data stream.

The earliest method *specifically* designed for decision-tree-based stream classification was proposed in [26]. This method is known as the *Very Fast Decision Tree (VFDT)* method. The VFDT method is based on the principle of the Hoeffding tree. This method essentially subsamples the data in order to achieve scalability in the construction of the decision tree. This is related to *BOAT* [43], which uses bootstrapping (rather than subsamples) in order to create more accurate trees. However, the *BOAT* method was designed for scalable decision tree construction, and not for the streaming scenario. Bootstrapping is also not an appropriate methodology for analyzing the streaming scenario.

This argument used in the Hoeffding tree is quite similar to *BOAT*. The idea is to show that the entire decision tree constructed would be the same as the one built on sub-sampled data with high probability. First, we describe a Hoeffding tree algorithm, which is constructed with the use of random sampling on static data. The idea is to determine a random sample of sufficient size so that the tree constructed on the sample is the same as that constructed on the entire data set. The

Hoeffding bound is used to show that the decision tree on the sub-sampled tree would make the same split as on the full stream with high probability. This approach can be used with a variety of criteria such as the gini-index, or information gain. For example, consider the case of the gini-index. For two attributes i and j, we would like to pick the attribute i, for which its gini-index G_i is smaller than G_j. While dealing with sampled data, the problem is that an error could be caused by the sampling process, and the order of the gini-index might be reversed. Therefore, for some threshold level ϵ, if $G_i - G_j < -\epsilon$ is true for the sampled data, it is desired that $G_i - G_j < 0$ on the original data with high probability. This would result in the same split at that node in the sampled and original data, if i and j correspond to the best and second-best, attributes, respectively. The number of examples required to produce the same split as the original data (with high probability) is determined. The Hoeffding bound is used to determine the number of relevant examples, so that this probabilistic guarantee may be achieved. If all splits in the decision tree are the same, then the same decision tree will be created. The probabilistic guarantees on each split can be converted to probabilistic guarantees on the construction of the entire decision tree, by aggregating the probabilities of error over the individual nodes. The Hoeffding tree can also be applied to data streams, by building the tree incrementally, as more examples stream in, from the higher levels to the lower levels. At any given node, one needs to wait until enough tuples are available in order to make decisions about lower levels. The memory requirements are modest, because only the counts of the different discrete values of the attributes (over different classes) need to be maintained in order to make split decisions. The VFDT algorithm is also based on the Hoeffding tree algorithm, though it makes a number of modifications. Specifically, it is more aggressive about making choices in the tie breaking of different attributes for splits. It also allows the deactivation of less promising leaf nodes. It is generally more memory efficient, because of these optimizations.

The original *VFDT* method is not designed for cases where the stream is evolving over time. The work in [47] extends this method to the case of concept-drifting data streams. This method is referred to as *CVFDT*. *CVFDT* incorporates two main ideas in order to address the additional challenges of drift:

1. A sliding window of training items is used to limit the impact of historical behavior.

2. Alternate subtrees at each internal node i are constructed.

Because of the sliding window approach, the main issue here is the update of the attribute frequency statistics at the nodes, as the sliding window moves forward. For the incoming items, their statistics are added to the attribute frequencies in the current window, and the statistics of the expiring items at the other end of the window are decreased. Therefore, when these statistics are updated, some nodes may no longer meet the Hoeffding bound, and somehow need to be replaced. CVFDT associates each internal node i with a list of alternate subtrees split on different attributes. For each internal node i, a periodic testing mode is used, in which it is decided whether the current state of i should be replaced with one of the alternate subtrees. This is achieved by using the next set of incoming training records as a test set to evaluate the impact of the replacement. A detailed discussion of the CVFDT method may be found in the Chapter 4 on decision trees.

One of the major challenges of the VFDT family of methods is that it is naturally suited to categorical data. This is a natural derivative of the fact that decision tree splits implicitly assume categorical data. Furthermore, discretization of numerical to categorical data is often done offline in order to ensure good distribution of records into the discretized intervals. Of course, it is always possible to test all possible split points, while dealing with numerical data, but the number of possible split points may be very large, when the data is numerical. The bounds used for ensuring that the split is the same for the sampled and the original data may also not apply in this case. The technique in [48] uses numerical interval pruning in order to reduce the number of possible split points, and thereby make the approach more effective. Furthermore, the work uses the properties of the gain function on the entropy in order to achieve the same bound as the VFDT method with the use of a smaller number of samples. The work in [40] also extends the VFDT method for continuous data and drift and applies Bayes classifiers at the leaf nodes.

It was pointed out in [27] that it is often assumed that old data is very valuable in improving the accuracy of stream mining. While old data does provide greater robustness in cases where patterns in the stream are stable, this is not always the case. In cases where the stream has evolved, the old data may not represent the *currently* relevant patterns for classification. Therefore, the work in [27] proposes a method that is able to sensibly select the correct choice from the data with the use of a little extra cost. The technique in [27] uses a decision tree ensemble method, in which each component of the ensemble mixture is essentially a decision tree. While many models achieve the same goal using a sliding window model, the merit of this approach is the ability to perform systematic data selection.

Bifet et al [19] proposed a method, that shares similarities with the work in [40]. However, the work in [19] replaces the naive Bayes with perceptron classifiers. The idea is to gain greater efficiency, while maintaining competitive accuracy. Hashemi et al [46] developed a flexible decision tree, known as *FlexDT*, based on fuzzy logic. This is done in order to address noise and missing values in streaming data. The problem of decision tree construction has also been extended to the uncertain scenario by Liang [59]. A detailed discussion of some of the streaming decision-tree methods, together with pseudocodes, are provided in Chapter 4.

9.2.2 Rule-Based Methods for Data Streams

In rule-based methods, different combinations of attributes are associated with the class label. For the case of quantitative data, different intervals of the attributes are used in order to relate the attribute values to the class labels. Typically, rule-based methods have a very high level of interpretability for the classification process. The major disadvantage of rule-based methods is that if the data evolves significantly, then these methods cannot be used effectively. There are two kinds of rule-based methods that are primarily used in the literature for the static case:

1. *Rule-based methods with sequential covering:* These methods typically use the sequential covering paradigm in order to grow rules in the same way as a decision tree. Examples include *C4.5Rules* [73], *CN2* [23], and *RIPPER* [24]. Some of the classifiers such as *C4.5Rules* are almost directly rule-based versions of tree classifiers.

2. *Leveraging association rules:* In these cases, association patterns are mined in which the consequent of the rule represents a class label. An example is CBA [61].

Among the aforementioned methods, the second is easier to generalize to the streaming scenario, because online methods exist for frequent pattern mining in data streams. Once the frequent patterns have been determined, rule-sets can be constructed from them using any offline algorithm. Since frequent patterns can also be efficiently determined over sliding windows, such methods can also be used for decremental learning. The reader is referred to [1] for a primer on the streaming methods for frequent pattern mining.

Since decision trees can be extended to streaming data, the corresponding rule-based classifiers can also be extended to the streaming scenario. As discussed in the introduction chapter, the rule-growth phase of sequential covering algorithms shares a number of conceptual similarities with decision tree construction. Therefore, many of the methods for streaming decision tree construction can be extended to rule growth. The major problem is that the sequential covering algorithm assumes the availability of all the training examples at one time. These issues can however be addressed by sampling recent portions of the stream.

An interesting rule-based method, which is based on *C4.5Rules*, is proposed in [90]. This method is able to adapt to concept drift. This work distinguishes between proactive and reactive models. The idea in a proactive model is to try to anticipate the concept drift that will take place, and to make adjustments to the model on this basis. A reactive model is one in which the additional concept drift that has already occurred is used in order to modify the model. The work in [90] uses *C4.5Rules* [73] as the base leaner in order to create the triggers for the different scenarios. The section on text stream classification in this chapter also discusses a number of other rule-based methods for streaming classification.

Another recent method, known as *LOCUST* [4], uses a lazy learning method in order to improve the effectiveness of the classification process in an evolving data stream. The reader is advised to refer to Chapter 6 on instance-based learning. Thus, the training phase is completely dispensed with, except for the fact that the training data is organized in the form of inverted lists on the discretized attributes for efficient retrieval. These lists are maintained using an online approach, as more data arrives. Furthermore, the approach is resource-adaptive, because it can adjust to varying speeds of the underlying data stream. The way in which the algorithm is made resource-adaptive is by structuring it as an *any-time* algorithm. The work in [4] defines an online analytical processing framework for real-time classification. In this technique, the data is received continuously over time, and the classification is performed in real time, by sampling local subsets of attributes, which are relevant to a particular data record. This is achieved by sampling the inverted lists on the discretized attributes. The intersection of these inverted lists represents a subspace local to the test instance. Thus, each of the sampled subsets of attributes represents an *instance-specific* rule in the locality of the test instance. The majority class label of the local instances in that subset of attributes is reported as the relevant class label for a particular record for that sample. Since multiple attribute samples are used, this provides greater robustness to the estimation process. Since the approach is structured as an *any-time* method, it can vary on the number of samples used, during periods of very high load.

9.2.3 Nearest Neighbor Methods for Data Streams

In nearest neighbor methods a sample of the data stream is maintained. The class label is defined as the majority (or largest) label presence among the k-nearest neighbors of the test instance. Therefore, the approach is quite simple, in that it only needs to maintain a dynamic sample from the data stream. The process of finding a dynamic sample from a data stream is referred to as *reservoir sampling*. A variety of unbiased [86] and biased [12] sampling methods are available for this purpose. In biased methods, more recent data points are heavily weighted as compared to earlier data points. Such an approach can account for concept drift in the data stream.

One of the earliest nearest neighbor classification methods that was specifically designed for data streams was the *ANNCAD* algorithm [54]. The core idea in this work is that when data is non-uniform, it is difficult to predetermine the number of nearest neighbors that should be used. Therefore, the approach allows an adaptive choice of the number of k nearest neighbors. Specifically, one adaptively expands the nearby area of a test point until a satisfactory classification is obtained. In order to achieve this efficiently, the feature space of the training set is decomposed with the use of a multi-resolution data representation. The information from different resolution levels can then be used for adaptively preassigning a class to every cell. A test point is classified on the basis of the label of its cell. A number of efficient data structures are proposed in [54] in order to speed up the approach.

Another method proposed in [14] uses micro-clustering in order to perform nearest neighbor classification. While vanilla nearest neighbor methods use the original data points for the classification process, the method proposed in [14] uses supervised *micro-clusters* for this purpose. A cluster can be viewed as a pseudo-point, which is represented by its centroid. The advantage of using clusters rather than the individual data points is that there are fewer clusters than data points. Therefore, the classification can be performed much more efficiently. Of course, since each cluster must contain data points from the same class, the micro-clustering approach needs to be modified in order to create class specific clusters. The clusters must also not be too coarse, in order to ensure that sufficient accuracy is retained.

A micro-cluster is a statistical data structure that maintains the zero-th order, first order, and second order moments from the data stream. It can be shown that these moments are sufficient for maintaining most of the required cluster statistics. In this approach, the algorithm dynamically maintains a set of micro-clusters [11], such that each micro-cluster is constrained to contain data points of the same class. Furthermore, snapshots of the micro-clusters are maintained (indirectly) over different historical horizons. This is done by maintaining micro-cluster statistics since the beginning of the stream and additively updating them. Then, the statistics are stored either uniformly or over pyramidally stored intervals. By storing the statistics over pyramidal intervals, better space efficiency can be achieved. The statistics for a particular horizon $(t_c - h, t_c)$ of length h may be inferred by subtracting the statistics at time $t_c - h$ from the statistics at time t_c. Note that statistics over multiple time horizons can be constructed using such an approach.

In order to perform the classification, a key issue is the choice of the horizon to be used for a particular test instance. For this purpose, the method in [14] separates out a portion of the training stream as the *hold out* stream. The classification accuracy is tested over different time horizons over the hold out streams, and the best accuracy horizon is used in each case. This approach is similar to the standard "bucket of models" approach used for parameter-tuning in data classification [92]. The major difference in this case is that the parameter being tuned is temporal time-horizon, since it is a data stream, that is being considered. This provides more effective results because smaller horizons are automatically used in highly evolving streams, whereas larger horizons are used in less evolving streams. It has been shown in [14] that such an approach is highly adaptive to different levels of evolution of the data stream. The microclustering approach also has the advantage that it can be naturally generalized to positive-unlabeled data stream classification [58].

9.2.4 SVM Methods for Data Streams

SVM methods have rarely been used for data stream classification because of the challenges associated with incrementally updating the SVM classification model with an increasing number of records. Some methods such as *SVMLight* [50] and *SVMPerf* [51] have been proposed for scaling up SVM classification. These methods are discussed in the next chapter on big data. However, these methods are not designed for the streaming model, which is much more restrictive. SVM classifiers are particularly important to generalize to the streaming scenario, because of their widespread applicability. These methods use a quadratic programming formulation with as many constraints as the number of data points. This makes the problem computationally intensive. In the case of kernel methods, the size of the kernel matrix scales with the *square* of the number of data points. In a stream, the number of data points constantly increases with time. Clearly, it is not reasonable to assume that such methods can be used directly in the streaming context.

In the context of support vector machines, the main thrust of the work is in three different directions, either separately or in combination:

1. It is desired to make support vector machines incremental, so that the model can be adjusted efficiently as new examples come in, without having to learn from scratch. Because of the large size of the SVM optimization problem, it is particularly important to avoid learning from scratch.

2. In many cases, incremental learning is combined with learning on a *window* of instances. Learning on a window of instances is more difficult, because one has to adjust not only for the incoming data points, but also points that fall off at the expiring end of the window. The process of removing instances from a trained model is referred to as *decremental learning*.

3. In the context of concept drift detection, it is often difficult to estimate the correct window size. During periods of fast drift, the window size should be small. During periods of slow drift, the window size should be larger, in order to minimize generalization error. This ensures

that a larger number of training data points are used during stable periods. Therefore, the window sizes should be adapted according to the varying trends in the data stream. This can often be quite challenging in the streaming scenario.

The work in streaming SVM-classification is quite significant, but relatively less known than the more popular techniques on streaming decision trees and ensemble analysis. In fact, some of the earliest work [52, 82] in streaming SVM learning precedes the earliest streaming work in decision tree construction. The work in [52] is particularly notable, because it is one of the earliest works that uses a dynamic window-based framework for adjusting to concept drift. This precedes most of the work on streaming concept drift performed subsequently by the data mining community. Other significant works on incremental support vector machines are discussed in [30, 36, 38, 74, 76, 80].

The key idea in most of these methods is that SVM classification is a non-linear programming problem, which can be solved with the use of Kuhn-Tucker conditions. Therefore, as long as it is possible to account for the impact of the addition or removal of test instances on these conditions, such an approach is likely to work well. Therefore, these methods show how to efficiently maintain the optimality conditions while adding or removing instances. This is much more efficient than retraining the SVM from scratch, and is also very useful for the streaming scenario. In terms of popularity, the work in [30] is used most frequently as a representative of support vector learning. The notable feature of this work is that it shows that decremental learning provides insights about the relationship between the generalization performance and the geometry of the data.

9.2.5 Neural Network Classifiers for Data Streams

The basic unit in a neural network is a *neuron* or *unit*. Each unit receives a set of inputs, which are denoted by the vector $\overline{X_i}$, which in this case corresponds to the term frequencies in the ith document. Each neuron is also associated with a set of weights A, which are used in order to compute a function $f(\cdot)$ of its inputs. A typical function that is often used in the neural network is the linear function as follows:

$$p_i = A \cdot \overline{X_i} + b. \tag{9.1}$$

Thus, for a vector $\overline{X_i}$ drawn from a lexicon of d words, the weight vector A should also contain d elements. Here b denotes the bias. Now consider a binary classification problem, in which all labels are drawn from $\{+1, -1\}$. We assume that the class label of $\overline{X_i}$ is denoted by y_i. In that case, the sign of the predicted function p_i yields the class label.

Since the sign of the function $A \cdot \overline{X_i} + b$ yields the class label, the goal of the approach is to *learn* the set of weights A with the use of the training data. The idea is that we start off with random weights and gradually update them when a mistake is made by applying the current function on the training example. The magnitude of the update is regulated by a learning rate μ. This forms the core idea of the *perceptron algorithm*, which is as follows:

```
Perceptron Algorithm
Inputs: Learning Rate: $\mu$
    Training Data $(\overline{X_i}, y_i)\ \forall i \in \{1 \ldots n\}$
Initialize weight vectors in $A$ and $b$ to 0 or small random numbers
repeat
Apply each training data to the neural network to check if the
    sign of $A \cdot \overline{X_i} + b$ matches $y_i$;
if sign of $A \cdot \overline{X_i} + b$ does not match $y_i$, then
    update weights $A$ based on learning rate $\mu$
until weights in $A$ converge
```

The weights in A are typically updated (increased or decreased) proportionally to $\mu \cdot \overline{X_i}$, so as to reduce the direction of the error of the neuron. We further note that many different update rules have been proposed in the literature. For example, one may simply update each weight by μ, rather than by $\mu \cdot \overline{X_i}$. This is particularly possible in domains such as text, in which all feature values take on small non-negative values of relatively similar magnitude. Most neural network methods iterate through the training data several times in order to learn the vector of weights A. This is evident from the pseudocode description provided above. This is however, not possible in the streaming scenario. However, in the streaming scenario, the number of training examples are so large that it is not necessary to iterate through the data multiple times. It is possible to continuously train the neural network using incoming examples, as long as the number of examples are large enough. If desired, segments of the data stream may be isolated for multiple iterations, where needed. A number of implementations of neural network methods for text data have been studied in [35, 60, 70, 79, 89]. Neural networks are particularly suitable for data stream scenarios because of their incremental approach to the model update process. The extension of these methods to the stream scenario is straightforward, and is discussed in the section on text stream classification. These methods are not specific to text, and can be extended to any scenario, though they are used very commonly for text data.

9.2.6 Ensemble Methods for Data Streams

Ensemble methods are a particularly popular method for stream classification, because of the fact that classification models can often not be built robustly in a fast data stream. Therefore, the models can be made more robust by using a combination of classifiers.

The two most common ensemble methods include bagging and boosting [71]. These methods model the number of times that a data item occurs in one of the base classifiers, and show that it is suitable for streaming data. The work in [81] proposes an ensemble method to read training items sequentially as blocks. The classifier is learned on one block and evaluated on the next. The ensemble components are selected on the basis of this qualitative evaluation. The major weakness of these methods is that they fail to take the concept drift into account.

Such a method was proposed in [87]. The method is also designed to handle concept drift, since it can effectively capture the evolution in the underlying data. The data stream is partitioned into chunks, and multiple classifiers are utilized on each of these chunks. The final classification score is computed as a function of the score on each of these chunks. In particular, ensembles of classification models are scored, such as C4.5, RIPPER, and naive Bayesian, from sequential chunks of the data stream. The classifiers in the ensemble are weighted based on their expected classification accuracy on the test data under the time-evolving environment. This ensures that the approach is able to achieve a higher degree of accuracy, since the classifiers are dynamically tuned in order to optimize the accuracy for that part of the data stream. The work in [87] shows that an ensemble classifier produces a smaller error than a single classifier, if the weights of all classifiers are assigned based on their expected classification accuracy.

A significant body of work [53, 95, 96] uses the approach of example and model weighting or selection in order to obtain the best accuracy over different portions of the concept drifting data stream. Typically, the idea in all of these methods is to pick the best classifier which works for that test instance or a portion of the training data stream. The work in [88] uses a stochastic optimization model in order to reduce the risk of overfitting and minimize classification bias. The diversity arising from the use of the ensemble approach has a significant role in the improvement of classification accuracy [69].

Both homogeneous and heterogeneous ensembles can be very useful in improving the accuracy of classification, as long as the right subset of data is selected for training in the different ensemblar components. An example of a homogeneous decision tree is presented in [27], in which the individual components of the ensemble are decision trees. The work in [27] carefully selects the subset of data on which to apply the method in order to achieve the most accurate results. It was pointed out in [42] that the appropriate assumptions for mining concept drifting data streams are not always easy to infer from the underlying data. It has been shown in [42] that a simple voting-based ensemblar framework can sometimes perform more effectively than relatively complex models.

Ensemble methods have also been extended to the rare class scenario. For example, the work in [41] is able to achieve effective classification in the context of skewed distributions. This scenario is discussed in detail in the next section.

9.3 Rare Class Stream Classification

In the rare class scenario, one or more classes are presented to a much lower degree than the other classes. Such distributions are known as skewed distributions. The rare class scenario is closely related to supervised outlier detection, and is discussed in detail in Chapter 17 of this book, for the static case. An overview on the topic may be found in [31]. In many cases, costs may be associated with examples, which provide an idea of the relative cost of incorrect classification of the different classes. In the static scenario, most classification methods can be extended to the rare class scenario by using a variety of simple methods:

- *Example Re-Weighting:* In this case, the examples are re-weighted using the misclassification costs.
- *Example Over-sampling or Under-sampling:* In this case, the examples are re-weighted implicitly or explicitly by oversampling or undersampling the different classes.

The supervised scenario is a a very rich one in the temporal domain, because *different kinds of temporal and frequency-based aspects of the classes could correspond to rare class*. In many cases, full supervision is not possible, and therefore semi-supervised outliers may need to be found. Such semi-supervised cases will be discussed in this chapter, because of their importance to the classification process. The different scenarios could correspond to novel class outliers, rare class outliers, or infrequently recurring class outliers. Thus, a combination of methods for concept drift analysis, outlier detection, and rare class detection may need to be used in order to accurately identify such cases in the streaming scenario. Such scenarios could arise quite often in applications such as intrusion detection, in which some known intrusions may be labeled, but new intrusions may also arise over time. Therefore, it is critical for the anomaly detection algorithm to use a combination of supervised and unsupervised methods in order to perform the labeling and detect outliers both in a semi-supervised and fully supervised way. Therefore, this problem may be viewed as a hybrid between outlier detection and imbalanced class learning.

The problem of rare class detection is also related to that of *cost-sensitive learning* in which costs are associated with the records. This is discussed in detail in Chapter 17 on rare class detection. It should be pointed out that the ensemble method discussed in [87] is able to perform cost-sensitive learning, though the method is not particularly focussed on different variations of the rare class detection problem in the streaming scenario. In particular, the method in [41] shows how the ensemble method can be extended to the rare class scenario. The work in [41] uses implicit oversampling of the rare class and undersampling of the copious class in order to achieve effective classification in the rare class scenario. The rare class detection problem has numerous variations in the streaming scenario, which are not quite as evident in the static scenario. This is because of the presence of novel classes and recurring classes in the streaming scenario, which need to be detected.

The problem of rare class detection is related to, but distinct from, the problem of classifying data streams with a limited number of training labels [66]. In the latter case, the labels are limited but may not necessarily be restricted to a particular class, However, in practice, the problem of limited labels arises mostly in binary classification problems where the rare class has a relatively small number of labels, and the remaining records are unlabeled.

The different kinds of scenarios that could arise in these contexts are as follows:

- *Rare Classes:* The determination of such classes is similar to the static supervised scenario, except that it needs to be done efficiently in the streaming scenario. In such cases, a small fraction of the records may belong to a rare class, but they may not necessarily be distributed in a non-homogenous way from a temporal perspective. While some concept drift may also need to be accounted for, the modification to standard stream classification algorithms remains quite analogous to the modifications of the static classification problem to the rare-class scenario.

- *Novel Classes:* These are classes, that were not encountered before in the data stream. Therefore, they may not be reflected in the training model at all. Eventually, such classes may become a normal part of the data over time. This scenario is somewhat similar to semi-supervised outlier detection in the static scenario, though the addition of the temporal component brings a number of challenges associated with it.

- *Infrequently Recurring Classes:* These are classes, that have not been encountered for a while, but may re-appear in the stream. Such classes are different from the first type of outliers, because they arrive in *temporally rare bursts*. Since most data stream classification algorithms use some form of discounting in order to address concept drift, they may sometimes completely age out information about old classes. Such classes cannot be distinguished from novel classes, if the infrequently recurring classes are not reflected in the training model. Therefore, issues of model update and discounting are important in the detection of such classes. The third kind of outlier was first proposed in [64].

We discuss each of the above cases below.

9.3.1 Detecting Rare Classes

Numerous classifiers are available for the streaming scenario [1], especially in the presence of concept drift. For detecting *rare classes*, the only change to be made to these classifiers is to add methods that are intended to handle the *class imbalance*. Such changes are not very different from those of addressing class imbalance in the static context. In general, the idea is that classes are often associated with costs, which enhance the importance of the rare class relative to the normal class. Since the broad principles of detecting rare classes do not change very much between the static and dynamic scenario, the discussion in this section will focus on the other two kinds of outliers.

9.3.2 Detecting Novel Classes

Novel class detection is the problem of detecting classes that have not occurred earlier either in the training or test data. Therefore, no ground truth is available for the problem of novel class detection. The problem of novel class detection will be studied in the online setting, which is the most common scenario in which it is encountered. Detecting novel classes is also a form of *semi-supervision*, because models are available about many of the other classes, but not the class that is being detected. The fully unsupervised setting corresponds to the clustering problem [11, 13, 94, 97], in which newly created clusters are reported as novel classes. In the latter case, the models are created in an unsupervised way with the use of clustering or other unsupervised methods. The work in [64, 65] addresses the problem of novel class detection in the streaming scenario.

Much of the traditional work on novel class detection [67] is focussed only on finding novel classes that are different from the current models. However, this approach does not distinguish between the different novel classes that may be encountered over time. A more general way of understanding the novel class detection problem is to view it as a combination of supervised (classification) and unsupervised (clustering) models. Thus, as in unsupervised novel class detection models such as first story detection [13, 97], the cohesion between the test instances of a novel class is important in determining whether they belong to the same novel class or not. The work in [64, 65] combines both supervised and semi-supervised models by:

- Maintaining a supervised model of the classes available in the training data as an ensemble of classification models.

- Maintaining an unsupervised model of the (unlabeled) novel classes received so far as cohesive groups of tightly knit clusters.

When a new test instance is received, the classification model is first applied to it to test whether it belongs to a currently existing (labeled) class. If this is not the case, it is tested whether it naturally belongs to one of the novel classes. The relationship of the test instance to a statistical boundary of the clusters representing the novel classes is used for this purpose. If neither of these conditions hold, it is assumed that the new data point should be in a novel class of its own. Thus, this approach creates a flexible scenario that combines supervised and unsupervised methods for novel class detection.

9.3.3 Detecting Infrequently Recurring Classes

Many outliers in real applications often arrive in *infrequent temporal bursts*. Many classifiers cannot distinguish between novel classes and rare classes, especially if the old models have aged out in the data stream. Therefore, one solution is to simply report a recurring outlier as a novel outlier.

This is however not a very desirable solution because novel-class detection is a semi-supervised problem, whereas the detection of recurring classes is a fully supervised problem. Therefore, by remembering the distribution of the recurring class over time, it is possible to improve the classification accuracy. The second issue is related to computational and resource efficiency. Since novel class detection is much more computationally and memory intensive than the problem of identifying an existing class, it is inefficient to treat recurring classes as novel classes. The work in [64] is able to identify the recurring classes in the data, and is also able to distinguish between the different kinds of recurring classes, when they are distinct from one another, by examining their relationships in the feature space. For example, two kinds of recurring intrusions in a network intrusion detection application may form distinct clusters in the stream. The work in [64] is able to distinguish between the different kinds of recurring classes as well. Another recent method that improves upon the basic technique of detecting rare and recurring classes is proposed in [16].

9.4 Discrete Attributes: The Massive Domain Scenario

In this section, we will study the problem of stream classification with discrete or categorical attributes. Stream classification with discrete attributes is not very different from continuous attributes, and many of the aforementioned methods can be extended easily to categorical data with minor modifications. A particularly difficult case in the space of discrete attributes is the massive domain scenario.

A massive-domain stream is defined as one in which each attribute takes on an extremely large number of possible values. Some examples are as follows:

1. In internet applications, the number of possible source and destination addresses can be very large. For example, there may be well over 10^8 possible IP-addresses.
2. The individual items in supermarket transactions are often drawn from millions of possibilities.
3. In general, when the attributes contain very detailed information such as locations, addresses, names, phone numbers or other such information, the domain size is very large. Recent years have seen a considerable increase in such applications because of advances in data collection techniques.

Many synopsis techniques such as sketches and distinct-element counting are motivated by the massive-domain scenario [34]. Note that these data structures would not be required for streams with a small number of possible values. Therefore, while the importance of this scenario is well understood in the context of synopsis data structures, it is rarely studied in the context of core mining problems. Recent work has also addressed this scenario in the context of core mining problems such as clustering [10] and classification [5].

The one-pass restrictions of data stream computation create further restrictions on the *computational approach*, which may be used for discriminatory analysis. Thus, the massive-domain size creates challenges in terms of space requirements, whereas the stream model further restricts the classes of algorithms that may be used in order to create space-efficient methods. For example, consider the following types of classification models:

- Techniques such as decision trees [20,73] require the computation of the discriminatory power of each possible attribute value in order to determine how the splits should be constructed. In order to compute the relative behavior of different attribute values, the discriminatory power of different attribute values (or combinations of values) need to be maintained. This becomes difficult in the context of massive data streams.
- Techniques such as rule-based classifiers [24] require the determination of combinations of attributes that are relevant to classification. With increasing domain size, it is no longer possible to compute this efficiently either in terms of space or running time.

The stream scenario presents additional challenges for classifiers. This is because the one-pass constraint dictates the choice of data structures and algorithms that can be used for the classification problem. All stream classifiers such as that discussed in [26] implicitly assume that the underlying domain size can be handled with modest main memory or storage limitations.

In order to discuss further, some notations and definitions will be introduced. The data stream $\mathcal{D}$ contains d-dimensional records that are denoted by $\overline{X_1} \dots \overline{X_N} \dots$. Associated with each record is a class that is drawn from the index $\{1 \dots k\}$. The attributes of record $\overline{X_i}$ are denoted by $(x_i^1 \dots x_i^d)$. It is assumed that the attribute value x_i^k is drawn from the unordered domain set $J_k = \{v_1^k \dots v_{M^k}^k\}$. The value of M^k denotes the domain size for the kth attribute. The value of M^k can be very large, and may range on the order of millions or billions. When the discriminatory power is defined in terms of subspaces of higher dimensionality, this number multiples rapidly to very large values. Such intermediate computations will be difficult to perform on even high-end machines.

Even though an extremely large number of attribute-value combinations may be possible over the different dimensions and domain sizes, only a limited number of these possibilities are usually relevant for classification purposes. Unfortunately, the intermediate computations required to effectively compare these combinations may not be easily feasible. The one-pass constraint of the data stream model creates an additional challenge in the computation process. In order to perform the classification, it is not necessary to explicitly determine the combinations of attributes that are related to a given class label. The more relevant question is the determination of *whether some combinations of attributes exist* that are strongly related to some class label. As it turns out, a sketch-based approach is very effective in such a scenario.

Sketch-based approaches [34] were designed for enumeration of different kinds of frequency statistics of data sets. The work in [5] extends the well-known count-min sketch [34] to the problem of classification of data streams. In this sketch, a total of $w = \lceil \ln(1/\delta) \rceil$ pairwise independent hash functions are used, each of which map onto uniformly random integers in the range $h = [0, e/\varepsilon]$, where e is the base of the natural logarithm. The data structure itself consists of a two dimensional array with $w \cdot h$ cells with a length of h and width of w. Each hash function corresponds to one of w 1-dimensional arrays with h cells each. In standard applications of the count-min sketch, the hash functions are used in order to update the counts of the different cells in this 2-dimensional data structure. For example, consider a 1-dimensional data stream with elements drawn from a massive set of domain values. When a new element of the data stream is received, each of the w hash functions are applied, in order to map onto a number in $[0 \dots h-1]$. The count of each of the set of w cells is incremented by 1. In order to *estimate* the count of an item, the set of w cells to which each of the w hash-functions map are determined. The minimum value among all these cells is determined. Let c_t be the true value of the count being estimated. The estimated count is at least equal to c_t, since all counts are non-negative, and there may be an over-estimation because of collisions among hash cells. As it turns out, a probabilistic upper bound to the estimate may also be determined. It has been shown in [34] that for a data stream with T arrivals, the estimate is at most $c_t + \varepsilon \cdot T$ with probability at least $1 - \delta$.

In typical subspace classifiers such as rule-based classifiers, low dimensional projections such as 2-dimensional or 3-dimensional combinations of attributes in the antecedents of the rule are used. In the case of data sets with massive domain sizes, the number of possible combinations of attributes (even for such low-dimensional combinations) can be so large that the corresponding statistics cannot be maintained explicitly during intermediate computations. However, the sketch-based method provides a unique technique for maintaining counts by creating *super-items* from different combinations of attribute values. Each super-item V containing a concatenation of the attribute value strings along with the dimension indices to which these strings belong. Let the actual value-string corresponding to value i_r be $S(i_r)$, and let the dimension index corresponding to the item i_r be $dim(i_r)$. In order to represent the dimension-value combinations corresponding to items $i_1 \dots i_p$, a new string is created by concatenating the strings $S(i_1) \dots S(i_p)$ and the dimension indices $dim(i_1) \dots dim(i_p)$.

This new super-string is then hashed into the sketch table as if it is the attribute value for the special super-item V. For each of the k-classes, a separate sketch of size $w \cdot h$ is maintained, and the sketch cells for a given class are updated only when a data stream item of the corresponding class is received. *It is important to note that the same set of w hash functions is used for updating the sketch corresponding to each of the k classes in the data.* Then, the sketch is updated once for each 1-dimensional attribute value for the d different attributes, and once for each of the super-items created by attribute combinations. For example, consider the case where it is desired to determine discriminatory combinations or 1- or 2-dimensional attributes. There are a total of $d + d \cdot (d-1)/2 = d \cdot (d+1)/2$ such combinations. Then, the sketch for the corresponding class is updated $L = d \cdot (d+1)/2$ times for each of the attribute-values or combinations of attribute-values. In general, L may be larger if even higher dimensional combinations are used, though for cases of massive domain sizes, even a low-dimensional subspace would have a high enough level of specificity for classification purposes. This is because of the extremely large number of combinations of possibilities, most of which would have very little frequency with respect to the data stream. For all practical purposes, one can assume that the use of 2-dimensional or 3-dimensional combinations provides sufficient discrimination in the massive-domain case. The value of L is dependent only on the dimensionality

```
Algorithm SketchUpdate(Labeled Data Stream: $\mathcal{D}$,
        NumClasses: $k$, MaxDim: $r$)
begin
  Initialize $k$ sketch tables of size $w \cdot h$ each
  with zero counts in each entry;
  repeat
    Receive next data point $\overline{X}$ from $\mathcal{D}$;
    Add 1 to each of the sketch counts in the
     (class-specific) table for all $L$ value-combinations
     in $\overline{X}$ with dimensionality less than $r$;
  until(all points in $\mathcal{D}$ have been processed);
end
```

FIGURE 9.1: Sketch updates for classification (training).

and is independent of the domain size along any of the dimensions. For modest values of d, the value of L is typically much lower than the number of possible combinations of attribute values.

The sketch-based classification algorithm has the advantage of being simple to implement. For each of the classes, a separate sketch table with $w \cdot d$ values is maintained. Thus, there are a total of $w \cdot d \cdot k$ cells that need to be maintained. When a new item from the data stream arrives, a total of $L \cdot w$ cells of the ith sketch table are updated. Specifically, for each item or super-item we update the count of the corresponding w cells in the sketch table by one unit. The overall approach for updating the sketch table is illustrated in Figure 9.1. The input to the algorithm is the data stream $\mathcal{D}$, the maximum dimensionality of the subspace combinations that are tracked, and the number of classes in the data set.

The key to using the sketch-based approach effectively is to be able to efficiently determine discriminative combinations of attributes. While one does not need to determine such combinations *explicitly* in closed form, it suffices to be able to *test* whether a *given* combination of attributes is discriminative. This suffices to perform effective classification of a given test instance. Consider the state of the data stream, when N records have arrived so far. The number of data streams records received from the k different classes are denoted by $N_1 \dots N_k$, so that we have $\sum_{i=1}^{k} N_i = N$.

Most combinations of attribute values have very low frequency of presence in the data stream. Here, one is interested in those combinations of attribute values that have high relative presence in one class compared to the other classes. Here we are referring to high *relative presence for a given class* in combination with a moderate amount of absolute presence. For example, if a particular combination of values occurs in 0.5% of the records corresponding to the class i, but it occurs in less than 0.1% of the records belonging to the other classes, then the relative presence of the combination in that particular class is high enough to be considered significant. Therefore, the discriminative power of a given combination of values (or super-item) V will be defined. Let $f_i(V)$ denote the fractional presence of the super-item V in class i, and $g_i(V)$ be the fractional presence of the super-item V in all classes other than i. In order to identify classification behavior specific to class i, the super-item V is of interest, if $f_i(V)$ is significantly greater than $g_i(V)$.

Definition 9.4.1 *The discriminatory power $\theta_i(V)$ of the super-item V is defined as the fractional difference in the relative frequency of the attribute-value combination V in class i versus the relative presence in classes other than i. Formally, the value of $\theta_i(V)$ is defined as follows:*

$$\theta_i(V) = \frac{f_i(V) - g_i(V)}{f_i(V)}. \tag{9.2}$$

Since one is interested only in items from which $f_i(V)$ is greater than $g_i(V)$, the value of $\theta_i(V)$

in super-items of interest will lie between 0 and 1. The larger the value of $\theta_i(V)$, the greater the correlation between the attribute-combination V and the class i. A value of $\theta_i(V) = 0$ indicates no correlation, whereas the value of $\theta_i(V) = 1$ indicates perfect correlation. In addition, it is interesting to determine those combinations of attribute values that occur in at least a fraction s of the records belonging to any class i. Such attribute value combinations are referred to as *discriminatory*. The concept of (θ, s)-discriminatory combinations is defined as follows:

Definition 9.4.2 *An attribute value-combination V is defined as (θ, s)-discriminatory with respect to class i, if $f_i(V) \geq s$ and $\theta_i(V) \geq \theta$.*

In other words, the attribute-value combination V occurs in at least a fraction s of the records belonging to class i, and has a discriminatory power of at least θ with respect to class i. Next, an approach for the *decision problem* of testing whether or not a given attribute-value combination is discriminatory will be discussed. This is required to perform classification of test instances.

In order to estimate the value of $\theta_i(V)$, the values of $f_i(V)$ and $g_i(V)$ also need to be estimated. The value of N_i is simply the sum of the counts in the table corresponding to the ith class divided by L. This is because exactly L sketch-table entries for each incoming record are updated. In order to estimate $f_i(V)$ we take the minimum of the w hash cells to which V maps for various hash functions in the sketch table corresponding to class i. This is the approach used for estimating the counts as discussed in [34]. The value of $g_i(V)$ can be estimated similarly, except that we compute a sketch table that contains the sum of the (corresponding cell-specific) counts in the $(k-1)$ classes other than i. This composite sketch table represents the other $(k-1)$ classes, since the same hash-function on each table is used. As in the previous case, the value of $g_i(V)$ is estimated by using the minimum cell value from all the w cells to which V maps. The value of $\theta_i(V)$ can then be estimated from these values of $f_i(V)$ and $g_i(V)$. The following result can be shown [5].

Lemma 9.4.1 *With probability at least $(1-\delta)$, the values of $f_i(V)$ and $g_i(V)$ are respectively over-estimated to within $L \cdot \epsilon$ of their true values when we use sketch tables with size $w = \lceil ln(1/\delta) \rceil$ and $h = \lceil e/\epsilon \rceil$.*

Next, the accuracy of estimation of $\theta_i(V)$ is estimated. Note that one is only interested in those attribute-combinations V for which $f_i(V) \geq s$ and $f_i(V) \geq g_i(V)$, since such patterns have sufficient statistical counts and are also discriminatory with $\theta_i(V) \geq 0$.

Lemma 9.4.2 *Let $\beta_i(V)$ be the estimated value of $\theta_i(V)$ for an attribute-combination V with fractional selectivity at least s and $f_i(V) \geq g_i(V)$. Let ϵ be chosen such that $\epsilon' = \epsilon \cdot L/s << 1$. With probability at least $1-\delta$, it is the case that $\beta_i(V) \leq \theta_i(V) + \epsilon'$.*

Next, we will examine the case when the value of $\theta_i(V)$ is under-estimated [5].

Lemma 9.4.3 *Let $\beta_i(V)$ be the estimated value of $\theta_i(V)$ for an attribute-combination V with fractional selectivity at least s and $f_i(V) \geq g_i(V)$. Let ϵ be chosen such that $\epsilon' = \epsilon \cdot L/s << 1$. With probability at least $1-\delta$, it is the case that $\beta_i(V) \geq \theta_i(V) - \epsilon'$.*

The results of Lemma 9.4.2 and 9.4.3 can be combined to conclude the following:

Lemma 9.4.4 *Let $\beta_i(V)$ be the estimated value of $\theta_i(V)$ for an attribute-combination V with fractional selectivity at least s and $f_i(V) \geq g_i(V)$. Let ϵ be chosen such that $\epsilon' = \epsilon \cdot L/s << 1$. With probability at least $1-2\cdot\delta$, it is the case that $\beta_i(V) \in (\theta_i(V) - \epsilon', \theta_i(V) + \epsilon')$.*

This result follows from the fact each of the inequalities in Lemmas 9.4.2 and 9.4.3 are true with probability at least $1-\delta$. Therefore, both inequalities are true with probability at least $(1-2\cdot\delta)$. Another natural corollary of this result is that any pattern that is truly (θ, s)-discriminatory will be

TABLE 9.1: Storage Requirement of Sketch Table for Different Data and Pattern Dimensionalities ($\epsilon' = 0.01$, $\delta = 0.01$, $s = 0.01$)

Data Dimensionality	**Max. Pattern Dimensionality**	L	$\epsilon = \epsilon' \cdot s/L$	# Cells = $\lceil e \cdot \ln(1/\delta)/\epsilon \rceil$	Storage Required	Storage $\epsilon' X20$
10	2	55	$10^{-4}/55$	6884350	26.3 MB	1.315 MB
10	3	175	$10^{-4}/175$	$21.9 * 10^6$	83.6 MB	4.18 MB
20	2	210	$10^{-4}/210$	$26.3 * 10^6$	100.27 MB	5.01 MB
20	3	1920	$10^{-4}/1920$	$24 * 10^7$	916.85 MB	45.8 MB
50	2	1275	$10^{-4}/1275$	$15.96 * 10^6$	608.85 MB	30.5 MB
100	2	5050	$10^{-4}/5050$	$63.21 * 10^7$	2.41 GB	120.5 MB
200	2	20100	$10^{-4}/20100$	$25.16 * 10^8$	9.6 GB	480 MB

discovered with probability at least $(1 - 2 \cdot \delta)$ by using the sketch based approach in order to determine all patterns that are at least $(\theta - \epsilon', s \cdot (1 - \epsilon'))$-discriminatory. Furthermore, the discriminatory power of such patterns will not be over- or under-estimated by an inaccuracy greater than ϵ'.

The process of determining whether a super-item V is (θ, s) requires us to determine $f_i(V)$ and $g_i(V)$ only. The value of $f_i(V)$ may be determined in a straightforward way by using the sketch based technique of [34] in order to determine the estimated frequency of the item. We note that $f_i(V)$ can be determined quite accurately since we are only considering those patterns that have a certain minimum support. The value of $g_i(V)$ may be determined by adding up the sketch tables for all the other different classes, and then using the same technique.

What are the space requirements of the approach for classifying large data streams under a variety of practical parameter settings? Consider the case where we have a 10-dimensional massive domain data set that has at least 10^7 values over each attribute. Then, the number of possible 2-dimensional and 3-dimensional value combinations are $10^{14} * 10 * (10 - 1)/2$ and $10^{21} * 10 * (10 - 1) * (10 - 2)/6$, respectively. We note that even the former requirement translates to an order of magnitude of about 4500 tera-bytes. Clearly, the intermediate-space requirements for aggregation-based computations of many standard classifiers are beyond most modern computers. On the other hand, for the sketch-based technique, the requirements continue to be quite modest. For example, let us consider the case where we use the sketch-based approach with $\epsilon' = 0.01$ and $\delta = 0.01$. Also, let us assume that we wish to have the ability to perform discriminatory classification on patterns with specificity at least $s = 0.01$. We have illustrated the storage requirements for data sets of different dimensionalities in Table 9.1. While the table is constructed for the case of $\epsilon' = 0.01$, the storage numbers in the last column of the table are illustrated for the case of $\epsilon' = 0.2$. We will see later that the use of much large values of ϵ' can provide effective and accurate results. It is clear that the storage requirements are quite modest, and can be held in main memory in most cases. For the case of stringent accuracy requirements of $\epsilon' = 0.01$, the high dimensional data sets may require a modest amount of disk storage in order to capture the sketch table. However, a practical choice of $\epsilon = 0.2$ always results in a table that can be stored in main memory. These results are especially useful in light of the fact that even data sets of small dimensionalities cannot be effectively processed with the use of traditional methods.

The sketch table may be leveraged to perform the classification for a given test instance $\overline{Y}$. The first step is to extract all the L patterns in the test instance with dimensionality less than r. Then, we determine those patterns that are (θ, s)-discriminatory with respect to at least one class. The process of finding (θ, s)-discriminatory patterns has already been discussed in the last section. We use a voting scheme in order to determine the final class label. Each pattern that is (θ, s)-discriminatory constitutes a vote for that class. The class with the highest number of votes is reported as the relevant class label. The overall procedure is reported in Figure 9.2.

```
Algorithm SketchClassify(Test Data Point: $\overline{Y}$,
        NumClasses: $k$, MaxDim: $r$)
begin
  Determine all $L$ value-combinations specific
  to test instance;
  Use sketch-based approach to determine which
   value-combinations are $(\theta, s)$-discriminatory;
  Add 1 vote to each class for which a pattern is
     $(\theta, s)$-discriminatory;
  Report class with largest number of votes;
end
```

FIGURE 9.2: Classification (testing phase).

9.5 Other Data Domains

The problem of stream classification is also relevant to other data domains such as text and graphs. In this section, a number of algorithms for these data domains will be studied in detail.

9.5.1 Text Streams

The problem of classification of data streams has been widely studied by the data mining community in the context of different kinds of data [1, 14, 87, 91]. Many of the methods for classifying numerical data can be easily extended to the case of text data, just as the numerical clustering method in [11] has been extended to a text clustering method in [13]. In particular, the classification method of [14] can be extended to the text domain, by adapting the concept of numerical micro-clusters to condensed droplets as discussed in [13]. With the use of the condensed droplet data structure, it is possible to extend all the methods discussed in [14] to the text stream scenario. Similarly, the core idea in [87] uses an ensemble-based approach on chunks of the data stream. This broad approach is essentially a *meta-algorithm* that is not too dependent upon the specifics of the underlying data format. Therefore, the broad method can also be easily extended to the case of text streams.

The problem of text stream classification arises in the context of a number of different IR tasks. The most important of these is *news filtering* [55], in which it is desirable to automatically assign incoming documents to pre-defined categories. A second application is that of email spam filtering [17], in which it is desirable to determine whether incoming email messages are spam or not. As discussed earlier, the problem of text stream classification can arise in two different contexts, depending upon whether the training or the test data arrives in the form of a stream:

- In the first case, the training data may be available for batch learning, but the test data may arrive in the form of a stream.
- In the second case, both the training and the test data may arrive in the form of a stream. The patterns in the training data may continuously change over time, as a result of which the models need to be updated dynamically.

The first scenario is usually easy to handle, because most classifier models are compact and classify individual test instances efficiently. On the other hand, in the second scenario, the training model needs to be constantly updated in order to account for changes in the patterns of the underlying training data. The easiest approach to such a problem is to incorporate temporal decay factors into model construction algorithms, so as to age out the old data. This ensures that the new (and more timely) data is weighted more significantly in the classification model. An interesting technique along this direction has been proposed in [77], in which a temporal weighting factor is introduced in order to modify the classification algorithms. Specifically, the approach has been applied to the Naive Bayes, Rocchio, and k-nearest neighbor classification algorithms. It has been shown that the incorporation of temporal weighting factors is useful in improving the classification accuracy, when the underlying data is evolving over time.

A number of methods have also been designed specifically for the case of text streams. In particular, the method discussed in [39] studies methods for classifying text streams in which the classification model may evolve over time. This problem has been studied extensively in the literature in the context of multi-dimensional data streams [14, 87]. For example, in a spam filtering application, a user may generally delete the spam emails for a particular topic, such as those corresponding to political advertisements. However, in a particular period such as the presidential elections, the user may be interested in the emails for that topic, and so it may not be appropriate to continue to classify that email as spam.

The work in [39] looks at the particular problem of classification in the context of user-interests. In this problem, the label of a document is considered either *interesting* or *non-interesting*. In order to achieve this goal, the work in [39] maintains the interesting and non-interesting topics in a text stream together with the evolution of the theme of the interesting topics. A document collection is classified into multiple topics, each of which is labeled either interesting or non-interesting at a given point. A concept refers to the main theme of interesting topics. A concept drift refers to the fact that the main theme of the interesting topic has changed.

The main goals of the work are to maximize the accuracy of classification and minimize the cost of re-classification. In order to achieve this goal, the method in [39] designs methods for detecting both *concept drift* as well as *model adaptation*. The former refers to the change in the theme of user-interests, whereas the latter refers to the detection of brand new concepts. In order to detect concept drifts, the method in [39] measures the classification error-rates in the data stream in terms of true and false positives. When the stream evolves, these error rates will increase, if the change in the concepts are not detection. In order to determine the change in concepts, techniques from statistical quality control are used, in which we determine the mean μ and standard deviation σ of the error rates, and determine whether this error rate remains within a particular tolerance, which is $[\mu - k \cdot \sigma, \mu + k \cdot \sigma]$. Here the tolerance is regulated by the parameter k. When the error rate changes, we determine when the concepts should be dropped or included. In addition, the drift rate is measured in order to determine the rate at which the concepts should be changed for classification purposes. In addition, methods for dynamic construction and removal of sketches are discussed in [39].

Another related work is that of one-class classification of text streams [93], in which only training data for the positive class is available, but there is no training data available for the negative class. This is quite common in many real applications in which it is easy to find representative documents for a particular topic, but it is hard to find the representative documents in order to model the background collection. The method works by designing an ensemble of classifiers in which some of the classifiers corresponds to a recent model, whereas others correspond to a long-term model. This is done in order to incorporate the fact that the classification should be performed with a combination of short-term and long-term trends. Another method for positive-unlabeled stream classification is discussed in [62].

A rule-based technique, which can learn classifiers incrementally from data streams, is the *sleeping-experts systems* [29, 33]. One characteristic of this rule-based system is that it uses the position of the words in generating the classification rules. Typically, the rules correspond to sets of words that are placed close together in a given text document. These sets of words are related to a class label. For a given test document, it is determined whether these sets of words occur in the document, and are used for classification. This system essentially learns a set of rules (or sleeping experts), which can be updated incrementally by the system. While the technique was proposed prior to the advent of data stream technology, its online nature ensures that it can be effectively used for the stream scenario.

One of the classes of methods that can be easily adapted to stream classification is the broad category of *neural networks* [79, 89]. This is because neural networks are essentially designed as a classification model with a network of perceptrons and corresponding weights associated with the term-class pairs. Such an incremental update process can be naturally adapted to the streaming context. These weights are incrementally learned as new examples arrive. The first neural network methods for online learning were proposed in [79, 89]. In these methods, the classifier starts off by setting all the weights in the neural network to the same value. The incoming training example is classified with the neural network. In the event that the result of the classification process is correct, then the weights are not modified. On the other hand, if the classification is incorrect, then the weights for the terms are either increased or decreased depending upon which class the training example belongs to. Specifically, if the class to which the training example belongs is a positive instance, the weights of the corresponding terms (in the training document) are increased by α. Otherwise, the weights of these terms are reduced by α. The value of α is also known as the *learning rate*. Many other variations are possible in terms of how the weights may be modified. For example, the method in [25] uses a multiplicative update rule, in which two multiplicative constants $\alpha_1 > 1$ and $\alpha_2 < 1$ are used for the classification process. The weights are multiplied by α_1, when the example belongs to the positive class, and is multiplied by α_2 otherwise. Another variation [56] also allows the modification of weights, when the classification process is correct. A number of other online neural network methods for text classification (together with background on the topic) may be found in [21, 28, 70, 75]. A Bayesian method for classification of text streams is discussed in [22]. The method in [22] constructs a Bayesian model of the text, which can be used for online classification. The key components of this approach are the design of a Bayesian online perceptron and a Bayesian online Gaussian process, which can be used effectively for online learning.

9.5.2 Graph Streams

Recently, the emergence of Internet applications has lead to increasing interest in *dynamic graph streaming applications* [7, 8, 37]. Such network applications are defined on a *massive underlying universe* of nodes. Some examples are as follows:

- The communication pattern of users in a social network in a modest time window can be decomposed into a group of disconnected graphs, which are defined over a massive-domain of user nodes.

- The browsing pattern of a single user (constructed from a proxy log) is typically a small subgraph of the Web graph. The browsing pattern over all users can be interpreted as a continuous stream of such graphs.

- The intrusion traffic on a communication network is a stream of localized graphs on the massive IP-network.

The most challenging case is one in which the data is available only in the form of a stream of graphs (or edge streams with graph identifiers). Furthermore, it is assumed that the *number of distinct edges is so large, that it is impossible to hold summary information about the underlying structures explicitly*. For example, a graph with more than 10^7 nodes may contains as many as 10^{13} edges. Clearly, it may not be possible to store information about such a large number of edges explicitly. Thus, the complexity of the graph stream classification problem arises in two separate ways:

1. Since the graphs are available only in the form of a stream, this restrains the kinds of algorithms that can be used to mine structural information for future analysis. An additional

complication is caused by the fact the the edges for a particular graph may arrive *out of order* (i.e., not contiguously) in the data stream. Since re-processing is not possible in a data stream, such out-of-order edges create challenges for algorithms that extract structural characteristics from the graphs.

2. The massive size of the graph creates a challenge for effective extraction of information that is relevant to classification. For example, it is difficult to even store summary information about the large number of distinct edges in the data. In such cases, the determination of frequent discriminative subgraphs may be computationally and space-inefficient to the point of being impractical.

A *discriminative subgraph mining approach* was proposed in [9] for graph stream classification. We will define discriminative subgraphs both in terms of edge pattern co-occurrence and the class label distributions. The presence of such subgraphs in test instances can be used in order to infer their class labels. Such discriminative subgraphs are difficult to determine with enumeration-based algorithms because of the massive domain of the underlying data. A probabilistic algorithm was proposed for determining the discriminative patterns. The probabilistic algorithm uses a hash-based summarization approach to capture the discriminative behavior of the underlying massive graphs. This compressed representation can be leveraged for effective classification of test (graph) instances.

The graph is defined over the node set N. This node set is assumed to be drawn from a massive domain of identifiers. The individual graphs in the stream are denoted by $G_1 \dots G_n \dots$. Each graph G_i is associated with the class label C_i, which is drawn from $\{1 \dots m\}$. It is assumed that the data is *sparse*. The sparsity property implies that even though the node and edge domain may be very large, the number of edges in the individual graphs may be relatively modest. This is generally true over a wide variety of real applications such as those discussed in the introduction section. In the streaming case, the edges of each graph G_i may not be neatly received at a given moment in time. Rather, the edges of different graphs may appear *out of order* in the data stream. This means that the edges for a given graph may not appear contiguously in the stream. This is definitely the case for many applications such as social networks and communication networks in which one cannot control the ordering of messages and communications across different edges. This is a particularly difficult case. It is assumed that the edges are received in the form $< GraphId >< Edge >< ClassLabel >$. Note that the class label is the same across all edges for a particular graph identifier. For notational convenience, we can assume that the class label is appended to the graph identifier, and therefore we can assume (without loss of generality or class information) that the incoming stream is of the form $< GraphId >< Edge >$. The value of the variable $< Edge >$ is defined by its two constituent nodes.

A rule-based approach was designed in [9], which associates discriminative subgraphs to specific class labels. Therefore, a way of quantifying the significance of a particular subset of edges P for the purposes of classification is needed. Ideally, one would like the subgraph P to have significant statistical presence in terms of the *relative frequency* of its constituent edges. At the same time, one would like P to be discriminative in terms of the class distribution.

The first criterion retains subgraphs with high relative frequency of presence, whereas the second criterion retains only subgraphs with high discriminative behavior. It is important to design effective techniques for determining patterns that satisfy both of these characteristics. First, the concept of a significant subgraph in the data will be defined. A significant subgraph P is defined as a subgraph (set of edges), for which the edges are correlated with one another in terms of absolute presence. This is also referred to as *edge coherence*. This concept is formally defined as follows:

Definition 9.5.1 *Let $f_\cap(P)$ be the fraction of graphs in $G_1 \dots G_n$ in which* **all** *edges of P are present. Let $f_\cup(P)$ be the fraction of graphs in which* **at least one or more** *of the edges of P are present. Then, the edge coherence $C(P)$ of the subgraph P is denoted by $f_\cap(P)/f_\cup(P)$.*

We note that the above definition of edge coherence is focussed on *relative presence* of subgraph patterns rather than the absolute presence. This ensures that only significant patterns are found. Therefore, large numbers of irrelevant patterns with high frequency but low significance are not considered. While the coherence definition is more effective, it is computationally quite challenging because of the size of the search space that may need to be explored. The randomized scheme discussed here is specifically designed in order to handle this challenge.

Next, the class discrimination power of the different subgraphs is defined. For this purpose, the *class confidence* of the edge set P with respect to the class label $r \in \{1 \ldots m\}$ is defined as follows.

Definition 9.5.2 *Among all graphs containing subgraph P, let $s(P,r)$ be the fraction belonging to class label r. We denote this fraction as the confidence of the pattern P with respect to the class r.*

Correspondingly, the concept of the dominant class confidence for a particular subgraph is defined.

Definition 9.5.3 *The dominant class confidence $DI(P)$ or subgraph P is defined as the maximum class confidence across all the different classes $\{1 \ldots m\}$.*

A significantly large value of $DI(P)$ for a particular test instance indicates that the pattern P is very relevant to classification, and the corresponding dominant class label may be an attractive candidate for the test instance label.

In general, one would like to determine patterns that are interesting in terms of absolute presence, and are also discriminative for a particular class. Therefore, the parameter-pair (α, θ) is defined, which corresponds to threshold parameters on the edge coherence and class interest ratio.

Definition 9.5.4 *A subgraph P is said to be be (α, θ)-significant, if it satisfies the following two* edge-coherence *and* class discrimination *constraints:*
(a) Edge Coherence Constraint: *The edge-coherence $C(P)$ of subgraph P is at least α. In other words, it is the case that $C(P) \geq \alpha$.*
(b) Class Discrimination Constraint: *The dominant class confidence $DI(P)$ is at least θ. In other words, it is the case that $DI(P) \geq \theta$.*

The above constraints are quite challenging because of the size of the search space that needs to be explored in order to determine relevant patterns. This approach is used, because it is well suited to massive graphs in which it is important to prune out as many irrelevant patterns as possible. The edge coherence constraint is designed to prune out many patterns that are abundantly present, but are not very significant from a relative perspective. This helps in more effective classification.

A probabilistic *min-hash approach* is used for determining discriminative subgraphs. The min-hash technique is an elegant probabilistic method, which has been used for the problem of finding interesting 2-itemsets [32]. This technique cannot be easily adapted to the graph classification problem, because of the large number of distinct edges in the graph. Therefore, w a 2-dimensional compression technique was used, in which a min-hash function will be used in combination with a more straightforward randomized hashing technique. We will see that this combination approach is extremely effective for graph stream classification.

The aim of constructing this synopsis is to be able to design a continuously updatable data structure, which can determine the most relevant discriminative subgraphs for classification. Since the size of the synopsis is small, it can be maintained in main memory and be used at any point during the arrival of the data stream. The ability to continuously update an in-memory data structure is a natural and efficient design for the stream scenario. At the same time, this *structural synopsis* maintains sufficient information, which is necessary for classification purposes.

For ease in further discussion, a tabular binary representation of the graph data set with N rows and n columns will be utilized. This table is only *conceptually* used for description purposes, but it is *not explicitly maintained* in the algorithms or synopsis construction process. The N rows correspond to the N different graphs present in the data. While columns represent the different edges in the data, this is not a one-to-one mapping. This is because the number of possible edges is so large that it is necessary to use a uniform random hash function in order to map the edges onto the n columns. The choice of n depends upon the space available to hold the synopsis effectively, and affects the quality of the final results obtained. Since many edges are mapped onto the same column by the hash function, the support counts of subgraph patterns are over-estimated with this approach. Since the edge-coherence $C(P)$ is represented as a ratio of supports, the edge coherence may either be over- or under-estimated. The details of this approach are discussed in [9]. A number of other models and algorithms for graph stream classification have recently been proposed [45,57,72]. The work in [57] uses discriminative hash kernels for graph stream classification. A method that uses a combination of hashing and factorization is proposed in [45]. Finally, the work in [72] addresses the problem of graph stream classification in the presence of imbalanced distributions and noise.

Social streams can be viewed as graph streams that contain a combination of text and structural information. An example is the *Twitter* stream, which contains both structural information (in the form of sender-recipient information), and text in the form of the actual tweet. The problem of event detection is closely related to that of classification. The main difference is that event labels are associated with time-instants rather than individual records. A method for performing supervised event detection in social streams is discussed in [15]. The work in [15] also discusses unsupervised methods for event detection.

9.5.3 Uncertain Data Streams

Probabilistic data has become increasingly popular because of the presence of numerous scenarios in which the data is recorded probabilistically. In such cases, the information about the probabilistic distribution of the data can be used in order to make better predictions. The work in [2] creates a density transform that can be used in order to create a kernel density estimate of the data. The kernel density estimate can be computed more accurately by using the information about the uncertainty behavior of the data. The work in [2] uses a relatively simple estimate of the uncertainty in terms of the standard deviation. It has been shown in [2] that the approach can be extended to real-time classification of data streams by using a micro-clustering approach. The micro-clusters can be maintained in real time, and the corresponding density estimates can be also be computed in real time. These can be used in order to create rules that are specific to the test instance in lazy fashion and perform the classification. The decision tree method has also been extended to the case of uncertain data streams [59].

9.6 Conclusions and Summary

This chapter provides a survey of data stream classification algorithms. We present the different types of methods for stream classification, which include the extension of decision trees, rule-based classifiers, nearest neighbor methods, and Bayes methods. In addition, ensemble methods were discussed for classification. A number of different scenarios were also discussed in this chapter, such as rare class methods and categorical data. The massive domain scenario is particularly challenging, because of the large number of distinct attribute values, that need to be considered. Different data domains such as text and graph data were also studied in the chapter. Domains such as graph data are likely to be especially important in the near future, because of the increasing importance of social streams and social networks in which such data sets arise. The related topic of big data classification is discussed in the next chapter.

Bibliography

[1] C. Aggarwal. *Data Streams: Models and Algorithms*, Springer, 2007.

[2] C. Aggarwal. On density-based transforms for uncertain data mining, *Proceedings of the ICDE Conference*, pages 866–875, 2007.

[3] C. Aggarwal. *Outlier Analysis*, Springer, 2013.

[4] C. Aggarwal and P. Yu. LOCUST: An online analytical processing framework for high dimensional classification of data streams, *Proceedings of the ICDE Conference*, pages 426–435, 2008.

[5] C. Aggarwal and P. Yu. On classification of high cardinality data streams. *Proceedings of the SDM Conference*, pages 802–813, 2010.

[6] C. Aggarwal and C. X. Zhai, *Mining Text Data*, Springer, 2012.

[7] C. Aggarwal, Y. Li, P. Yu, and R. Jin. On dense pattern mining in graph streams, *Proceedings of the VLDB Conference*, 3(1-2):975–984, 2010.

[8] C. Aggarwal, Y. Zhao, and P. Yu. On clustering graph streams, *Proceedings of the SDM Conference*, 2010.

[9] C. Aggarwal, On classification of graph streams, *Proceedings of the SDM Conference*, 2010.

[10] C. Aggarwal. A framework for clustering massive-domain data streams, *Proceedings of the ICDE Conference*, pages 102–113, 2009.

[11] C. C. Aggarwal, J. Han. J. Wang, and P. Yu. A framework for clustering evolving data streams, *Proceedings of the VLDB Conference*, pages 81–92, 2003.

[12] C. C. Aggarwal. On biased reservoir sampling in the presence of stream evolution, *Proceedings of the VLDB Conference*, pages 607–618, 2006.

[13] C. C. Aggarwal and P. Yu. On clustering massive text and categorical data streams, *Knowledge and Information Systems*, 24(2), pp. 171–196, 2010.

[14] C. Aggarwal, J. Han, J. Wang, and P. Yu. A framework for classification of evolving data streams. In *IEEE Transactions on Knowledge and Data Engineering*, 18(5):577–589, 2006.

[15] C. Aggarwal and K. Subbian. Event detection in social streams, *Proceedings of the SDM Conference*, 2012.

[16] T. Al-Khateeb, M. Masud, L. Khan, C. Aggarwal, J. Han, and B. Thuraisingham. Recurring and novel class detection using class-based ensemble, *Proceedings of the ICDM Conference*, 2012.

[17] I. Androutsopoulos, J. Koutsias, K. V. Chandrinos, and C. D. Spyropoulos. An experimental comparison of naive Bayesian and keyword-based anti-spam filtering with personal e-mail messages. *Proceedings of the ACM SIGIR Conference*, pages 160–167, 2000.

[18] A. Banerjee and J. Ghosh. Competitive learning mechanisms for scalable, balanced and incremental clustering of streaming texts, *Neural Networks*, pages 2697–2702, 2003.

[19] A. Bifet, G. Holmes, B. Pfahringer, and E. Frank. Fast perceptron decision tree learning from evolving data streams. *Advances in Knowledge Discovery and Data Mining*, pages 299–310, 2010.

[20] L. Breiman, J. Friedman, and C. Stone, *Classification and Regrssion Trees*, Chapman and Hall, 1984.

[21] K. Crammer and Y. Singer. A new family of online algorithms for category ranking, *ACM SIGIR Conference*, pages 151–158, 2002.

[22] K. Chai, H. Ng, and H. Chiu. Bayesian online classifiers for text classification and filtering, *ACM SIGIR Conference*, pages 97–104, 2002.

[23] P. Clark and T. Niblett. The CN2 induction algorithm, *Machine Learning*, 3(4): 261–283, 1989.

[24] W. Cohen. Fast effective rule induction, *ICML Conference*, pages 115–123, 1995.

[25] I. Dagan, Y. Karov, and D. Roth. Mistake-driven learning in text categorization. *Proceedings of Conference Empirical Methods in Natural Language Processing*, pages 55–63, 1997.

[26] P. Domingos and G. Hulten, Mining high-speed data streams, *ACM KDD Conference*, pages 71–80, 2000.

[27] W. Fan. Systematic data selection to mining concept drifting data streams, *ACM KDD Conference*, pages 128–137, 2004.

[28] Y. Freund and R. Schapire. Large margin classification using the perceptron algorithm, *Machine Learning*, 37(3):277–296, 1998.

[29] Y. Freund, R. Schapire, Y. Singer, M. Warmuth. Using and combining predictors that specialize. *Proceedings of the 29th Annual ACM Symposium on Theory of Computing*, pages 334–343, 1997.

[30] G. Cauwenberghs and T. Poggio. Incremental and decremental support vector machine learning, *NIPS Conference*, pages 409–415, 2000.

[31] N. Chawla, N. Japkowicz, and A. Kotcz. Editorial: special issue on learning from imbalanced data sets. *ACM SIGKDD Explorations Newsletter*, 6(1):1–6, 2004.

[32] E. Cohen, M. Datar, S. Fujiwara, A. Gionis, P. Indyk, R. Motwani, J. Ullman, and C. Yang, Finding interesting associations without support pruning, *IEEE TKDE*, 13(1): 64–78, 2001.

[33] W. Cohen and Y. Singer. Context-sensitive learning methods for text categorization. *ACM Transactions on Information Systems*, 17(2): 141–173, 1999.

[34] G. Cormode and S. Muthukrishnan. An improved data-stream summary: The count-min sketch and its applications, *Journal of Algorithms*, 55(1):58–75, 2005.

[35] I. Dagan, Y. Karov, and D. Roth. Mistake-driven learning in text categorization, *Proceedings of EMNLP*, pages 55-63, 1997.

[36] C. Domeniconi and D. Gunopulos. Incremental support vector machine construction. *ICDM Conference*, pages 589–592, 2001.

[37] J. Feigenbaum, S. Kannan, A. McGregor, S. Suri, and J. Zhang, Graph distances in the data stream model. *SIAM Jour. on Comp.*, 38(5):1709–1727, 2005.

[38] G. Fung and O. L. Mangasarian. Incremental support vector machine classification. *SIAM Conference on Data Mining*, pages 77–86, 2002.

[39] G. P. C. Fung, J. X. Yu, and H. Lu. Classifying text streams in the presence of concept drifts. *Advances in Knowledge Discovery and Data Mining*, 3056:373–383, 2004.

[40] J. Gama, R. Fernandes, and R. Rocha. Decision trees for mining data streams. *Intelligent Data Analysis*, 10:23–45, 2006.

[41] J. Gao, W. Fan, J. Han, and P. Yu. A general framework for mining concept drifting data stream with skewed distributions, *SDM Conference*, 2007.

[42] J. Gao, W. Fan, and J. Han. On appropriate assumptions to mine data streams: Analysis and practice, *ICDM Conference*, pages 143–152, 2007.

[43] J. Gehrke, V. Ganti, R. Ramakrishnan, and W.-Y. Loh. BOAT: Optimistic decision tree construction, *ACM SIGMOD Conference*, pages 169–180, 1999.

[44] J. Gehrke, R. Ramakrishnan, and V. Ganti. Rainforest—A framework for fast decision tree construction of large datasets, *VLDB Conference*, pages 416–427, 1998.

[45] T. Guo, L. Chi, and X. Zhu. Graph hashing and factorization for fast graph stream classification. *ACM CIKM Conference*, pages 1607–1612, 2013.

[46] S. Hashemi and Y. Yang. Flexible decision tree for data stream classification in the presence of concept change, noise and missing values. *Data Mining and Knowledge Discovery*, 19(1):95–131, 2009.

[47] G. Hulten, L. Spencer, and P. Domingos. Mining time-changing data streams. *ACM KDD Conference*, pages 97–106, 2001.

[48] R. Jin and G. Agrawal. Efficient Decision Tree Construction on Streaming Data, *ACM KDD Conference*, pages 571–576, 2003.

[49] D. Kalles and T. Morris. Efficient incremental induction of decision trees. *Machine Learning*, 24(3):231–242, 1996.

[50] T. Joachims. Making large scale SVMs practical. *Advances in Kernel Methods, Support Vector Learning*, pp. 169–184, MIT Press, Cambridge, 1998.

[51] T. Joachims. Training linear SVMs in linear time. *KDD*, pages 217–226, 2006.

[52] R. Klinkenberg and T. Joachims. Detecting concept drift with support vector machines. *ICML Conference*, pages 487–494, 2000.

[53] J. Kolter and M. Maloof. Dynamic weighted majority: A new ensemble method for tracking concept drift, *ICDM Conference*, pages 123–130, 2003.

[54] Y.-N. Law and C. Zaniolo. An adaptive nearest neighbor classification algorithm for data streams, *PKDD Conference*, pages 108–120, 2005.

[55] D. Lewis. The TREC-4 filtering track: description and analysis. *Proceedings of TREC-4, 4th Text Retrieval Conference*, pages 165–180, 1995.

[56] D. Lewis, R. E. Schapire, J. P. Callan, and R. Papka. Training algorithms for linear text classifiers. *ACM SIGIR Conference*, pages 298–306, 1996.

[57] B. Li, X. Zhu, L. Chi, and C. Zhang. Nested subtree hash kernels for large-scale graph classification over streams. *ICDM Conference*, pages 399–408, 2012.

[58] X. Li, P. Yu, B. Liu, and S. K. Ng. Positive-unlabeled learning for data stream classification, *SDM Conference*, pages 257–268, 2009.

[59] C. Liang, Y. Zhang, and Q. Song. Decision tree for dynamic and uncertain data streams. *2nd Asian Conference on Machine Learning*, volume 3, pages 209–224, 2010.

[60] N. Littlestone. Learning quickly when irrelevant attributes abound: A new linear-threshold algorithm. *Machine Learning*, 2: pages 285–318, 1988.

[61] B. Liu, W. Hsu, and Y. Ma, Integrating classification and association rule mining, *ACM KDD Conference*, pages 80–86, 1998.

[62] S. Pan, Y. Zhang, and X. Li. Dynamic classifier ensemble for positive unlabeled text stream classification. *Knowledge and Information Systems*, 33(2):267–287, 2012.

[63] J. R. Quinlan. *C4.5: Programs in Machine Learning*, Morgan-Kaufmann Inc, 1993.

[64] M. Masud, T. Al-Khateeb, L. Khan, C. Aggarwal, J. Gao, J. Han, and B. Thuraisingham. Detecting recurring and novel classes in concept-drifting data streams. *ICDM Conference*, pages 1176–1181, 2011.

[65] M. Masud, Q. Chen, L. Khan, C. Aggarwal, J. Gao, J. Han, A. Srivastava, and N. Oza. Classification and adaptive novel class detection of feature-evolving data streams, *IEEE Transactions on Knowledge and Data Engineering*, 25(7):1484–1487, 2013. Available at `http://doi.ieeecomputersociety.org/10.1109/TKDE.2012.109`

[66] M. Masud, J. Gao, L. Khan, J. Han, and B. Thuraisingham: A practical approach to classify evolving data streams: Training with limited amount of labeled data. *ICDM Conference*, pages 929–934, 2008.

[67] M. Markou and S. Singh. Novelty detection: A review, Part 1: Statistical approaches, *Signal Processing*, 83(12):2481–2497, 2003.

[68] M. Mehta, R. Agrawal, and J. Rissanen. SLIQ: A fast scalable classifier for data mining, *EDBT Conference*, pages 18–32, 1996.

[69] L. Minku, A. White, and X. Yao. The impact of diversity on online ensemble learning in the presence of concept drift, *IEEE Transactions on Knowledge and Data Engineering*, 22(6):730–742, 2010.

[70] H. T. Ng, W. B. Goh, and K. L. Low. Feature selection, perceptron learning, and a usability case study for text categorization. *SIGIR Conference*, pages 67–73, 1997.

[71] N. Oza and S. Russell. Online bagging and boosting. In *Eighth Int. Workshop on Artificial Intelligence and Statistics*, pages 105–112. Morgan Kaufmann, 2001.

[72] S. Pan and X. Zhu. Graph classification with imbalanced class distributions and noise. *AAAI Conference*, pages 1586–1592, 2013.

[73] R. Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann Publishers, 1993.

[74] L. Ralaivola and F. d'Alché-Buc. Incremental support vector machine learning: A local approach. *Artificial Neural Network*, pages 322–330, 2001.

[75] F. Rosenblatt. The perceptron: A probabilistic model for information and storage organization in the brain, *Psychological Review*, 65: pages 386–407, 1958.

[76] S. Ruping. Incremental learning with support vector machines. *IEEE ICDM Conference*, pp. 641–642, 2001.

[77] T. Salles, L. Rocha, G. Pappa, G. Mourao, W. Meira Jr., and M. Goncalves. Temporally-aware algorithms for document classification. *ACM SIGIR Conference*, pages 307–314, 2010.

[78] J. Schlimmer and D. Fisher. A case study of incremental concept induction. *Proceedings of the Fifth National Conference on Artificial Intelligence*, pages 495–501. Morgan Kaufmann, 1986.

[79] H. Schutze, D. Hull, and J. Pedersen. A comparison of classifiers and document representations for the routing problem. *ACM SIGIR Conference*, pages 229–237, 1995.

[80] A. Shilton, M. Palaniswami, D. Ralph, and A. Tsoi. Incremental training of support vector machines. *IEEE Transactions on Neural Networks*, 16(1):114–131, 2005.

[81] W. N. Street and Y. Kim. A streaming ensemble algorithm (sea) for large-scale classification. *ACM KDD Conference*, pages 377–382, 2001.

[82] N. Syed, H. Liu, and K. Sung. Handling concept drifts in incremental learning with support vector machines. *ACM KDD Conference*, pages 317–321, 1999.

[83] P. Utgoff. Incremental induction of decision trees. *Machine Learning*, 4(2):161–186, 1989.

[84] P. Utgoff and C. Brodley. An incremental method for finding multivariate splits for decision trees. *Proceedings of the Seventh International Conference on Machine Learning*, pages 58–65, 1990.

[85] P. Utgoff. An improved algorithm for incremental induction of decision trees. *Proceedings of the Eleventh International Conference on Machine Learning*, pages 318–325, 1994.

[86] J. Vitter. Random sampling with a reservoir, *ACM Transactions on Mathematical Software (TOMS)*, 11(1):37–57, 1985.

[87] H. Wang, W. Fan, P. Yu, J. Han. Mining concept-drifting data streams with ensemble classifiers, *KDD Conference*, pages 226–235, 2003.

[88] H. Wang, J. Yin, J. Pei, P. Yu, and J. X. Yu. Suppressing model overfitting in concept drifting data streams, *ACM KDD Conference*, pages 736–741, 2006.

[89] E. Wiener, J. O. Pedersen, and A. S. Weigend. A neural network approach to topic spotting, *SDAIR*, pages 317–332, 1995.

[90] Y. Yang, X. Wu, and X. Zhu. Combining proactive and reactive predictions for data streams, *ACM KDD Conference*, pages 710–715, 2005.

[91] K. L. Yu and W. Lam. A new on-line learning algorithm for adaptive text filtering, *ACM CIKM Conference*, pages 156–160, 1998.

[92] S. Dzeroski and B. Zenko. Is combining classifiers better than selecting the best one? *Machine Learning*, 54(3):255–273, 2004.

[93] Y. Zhang, X. Li, and M. Orlowska. One class classification of text streams with concept drift, *ICDMW Workshop*, pages 116–125, 2008.

[94] J. Zhang, Z. Ghahramani, and Y. Yang. A probabilistic model for online document clustering with application to novelty detection, *NIPS*, 2005.

[95] P. Zhang, Y. Zhu, and Y. Shi. Categorizing and mining concept drifting data streams, *ACM KDD Conference*, pages 812–820, 2008.

[96] X. Zhu, X. Wu, and Y. Zhang. Dynamic classifier selection for effective mining from noisy data streams, *ICDM Conference*, pages 305–312, 2004.

[97] `http://www.itl.nist.gov/iad/mig/tests/tdt/tasks/fsd.html`