Stream Classification

Jerzy Stefanowski, Dariusz Brzezinski
Poznan University of Technology, Poznan, Poland

An entry to appear in the Encyclopedia of Machine Learning (Springer)

Definition

Stream classification is a variant of incremental learning of classifiers that has to satisfy requirements specific for massive streams of data: restrictive processing time, limited memory, and one scan of incoming examples. Additionally, stream classifiers often have to be adaptive, as they usually act in dynamic, non-stationary environments where data and target concepts can change over time. To fulfill these requirements new solutions include dedicated data management and forgetting mechanisms, concept drift detectors that monitor the underlying changes in the stream, effective online single classifiers, and adaptive ensembles that continuously react to changes in the streams.

Motivation and Background

In many data intensive applications, like sensor networks, traffic control, market analysis, Web user tracking, and social media, massive volumes of data are continuously generated in the form of data streams. A data stream is a potentially unbounded, ordered sequence of data items, which arrive continuously at high-speeds. These data elements can be simple attribute-values pairs like relational database tuples or more complex structures such as graphs.

The main characteristics of streams include:

- continuous flow (elements arrive one after another),
- huge data volumes (possibly of an infinite length),
- rapid arrival rate (relatively high with respect to the processing power of the system),

- susceptibility to change (data distributions generating examples may change on the fly).

Due to the above characteristics, learning from data streams differs from batch learning, where data are stored in finite, persistent data repositories. The main dissimilarities include the sequential nature of the data, massive volumes, processing speed restrictions, and the fact that data elements cannot be accessed multiple times as it is in the case of learning from static repositories. Moreover, contrary to online learning, stream classification does not assume adversarial actions from the instance generating process, but rather focuses on computational restrictions.

One of the most widely studied tasks in data stream mining is supervised classification. Apart from the aforementioned general difficulties connected with learning from streams, classification is also often performed in non-stationary environments, where the data distribution and target concepts can change over time. This phenomenon, called concept drift, deteriorates the predictive accuracy of classifiers as the instances they were trained on differ from the current data. Typical examples of real-life concept drifts include content changes in unwanted emails in spam categorization or evolving customer preferences.

Several researchers imply the following requirements on algorithms learning classifiers from streams (Bifet et al., 2010):

1. Process one example at a time and inspect it only once.
2. Use a limited amount of memory.
3. Be ready to predict at any time.
4. Be able to react to concept drift in case of evolving data streams.

Typical batch learning algorithms for supervised classification are not capable of fulfilling all of the listed data stream requirements. Incremental learning is also insufficient, as it does not meet tight computational demands and does not tackle concept drift. Therefore, several new learning algorithms have been introduced. Surveys on stream classification, such as (Ditzler et al., 2015; Gama, 2010; Kuncheva, 2004), showcase research on using sliding windows to manage memory and provide a forgetting mechanism, sampling techniques, drift detectors, and new online algorithms.

Structure of the Learning System

Stream classification can be formalized as follows. Learning instances from a stream $\mathcal{S}$ appear incrementally as a sequence of labeled examples $\{\mathbf{x}^t, y^t\}$ for $t = 1, 2, \ldots, T$, where $\mathbf{x}$ is a vector of attribute values and y is a class label ($y \in \{K_1, \ldots, K_l\}$). A new example $\mathbf{x}^t$ is classified by a classifier C, which predicts its class label. Here, we consider a completely supervised framework where after some time the true class label y^t is available and can be used to update the classifier.

Examples from the data stream can be provided either *online*, i.e., instance by instance, or in portions (*blocks*). In the first approach, presented in Fig. 1, algorithms process single examples appearing one by one in consecutive moments in time, while in the other approach, presented in Fig. 2, examples are available only in larger sets called data blocks (or data chunks) $B_1, B_2, \ldots, B_n$, where n denotes the last element of the stream up to the current timepoint. Blocks are usually of equal size and the construction, evaluation, or updating of classifiers is done when all examples from a new block are available. This distinction also refers to the availability of class labels. For instance, in some problems data elements are naturally accumulated through some time and labeled in blocks. However, with class labels appearing online with single instances, algorithms have the possibility of reacting to concept drift much faster than in block based environments.

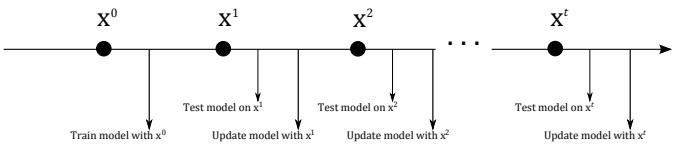

Figure 1: Online processing

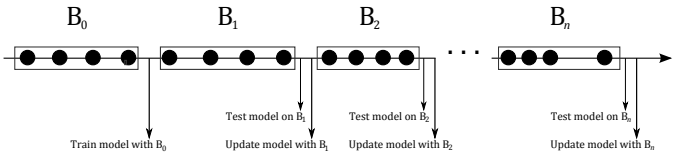

Figure 2: Block processing

Two basic models of data streams are considered: *stationary*, where examples are drawn from a fixed although unknown probability distribution, and *non-stationary*, where data can evolve over time. As process changes

occur in many real-world problems (Zliobaite et al., 2015), most stream classification algorithms are capable of predicting, detecting, and adapting to concept drifts.

Concept drift can be defined from the perspective of hidden data contexts, which are unknown to the learning algorithm. However, in case of evolving streams a more probabilistic view on the matter can be presented (Gama, 2010). In each point in time t, every example is generated by a source with a joint distribution $P^t(\mathbf{x}, y)$ over the data. Concepts in data are *stable* if all examples are generated by the same distribution. If for two distinct points in time t and $t + \Delta$ an $\mathbf{x}$ exists such that $P^t(\mathbf{x}, y) \neq P^{t+\Delta}(\mathbf{x}, y)$, then concept drift occurs. Although different component probabilities of $P^t(\mathbf{x}, y)$ may change (Gama et al., 2014), in case of supervised classification one is mainly interested in *real drift*, i.e., changes in posterior probabilities of classes $P(y|\mathbf{x})$.

Usually two basic types of concept drifts are distinguished: *sudden* (abrupt) and *gradual*. The first type of drift occurs when at a moment in time t the source data distribution in S^t is suddenly replaced by a different distribution in S^{t+1}. Gradual drifts are not so radical and they are connected with a slower rate of changes that can be noticed while observing a data stream for a longer period of time. In some domains, situations when previous concepts reappear after some time are separately treated and analyzed as *recurring* drifts (Gomes et al., 2014). Moreover, data streams can contain outliers and noise, but these are not considered as concept drifts and stream classifiers should be robust to these random changes.

Evaluation

Stream classification requirements make *processing time*, *memory usage*, *predictive performance*, and the *ability to adapt* key evaluation criteria.

The time required to process a single instance and the average memory usage should remain constant throughout the life of a stream classifier. That is why training and testing time as well as model size have to be periodically monitored during stream classification. Additionally, processor time and memory are also considered key costs when deploying a stream classification system and are sometimes measured in a single metric called RAM hours.

The predictive performance of stream classifiers is usually assessed using evaluation measures known from static supervised classification, such as accuracy or error-rate. However, contrary to batch learning scenarios, it is assumed that due to the size and speed of data streams repeated runs over

the data are not necessary to estimate these measures on labeled testing examples. Due to their computational costs, re-sampling techniques such as cross-validation or bootstrapping are deemed too expensive for streams. As a result, simpler error-estimation procedures are used, yet ones that build a picture of performance over time.

One of such evaluation procedures involves using a holdout test set to periodically evaluate the classifier's performance. An alternate scheme of estimating the performance of stream classifiers involves interleaving testing with training. Each individual example is first used to test the classifier before it is used for training (see Fig. 1). This evaluation procedure, often called *test-then-train*, has the advantage that it makes maximum use of the available data. A similar procedure of interleaving testing with training can also be performed with blocks of examples instead of single instances (see Fig. 2). However, for evolving streams the prequential evaluation procedure is suggested (Gama, 2010). The term *prequential* (blend of predictive and sequential) stems from online learning and is used in data stream mining literature to denote algorithms that base their functioning only on the most recent data, rather than the entire stream. Such a procedure highlights the current rather than overall performance and, as a result, showcases changes in the stream more clearly, which is especially important for drift detection. All three of the aforementioned evaluation procedures (holdout, test-then-train, prequential) are usually used to periodically calculate a selected metric, e.g., accuracy, and plot its value creating a line chart depicting classifier performance over time.

Finally, an important criterion when comparing stream classifiers is their ability to react to various types of concept changes. Adaptability can be evaluated by comparing drift reaction times. This is done by measuring the time between the start of a drift and the moment when the tested classifier's accuracy recovers to a level from before the drift. More elaborate methods of assessing the classifier's ability to adapt include *recovery analysis* and *controlled permutations* (Krempl et al., 2014). Nevertheless, in order to calculate reaction times and other adaptability measures, usually a human expert needs to determine moments when a drift starts and when a classifier recovers from it. Alternately, such evaluations are carried out with synthetic data generators.

Algorithms

The simplest categorization of algorithms for learning stream classifiers makes a distinction between *single classifiers* and *ensembles*. Additionally, from the perspective of learning from drifting environments, most of researchers distinguish *active* approaches, which trigger changes in classifiers when drifts are detected, and *passive* approaches, which continuously update the classifier regardless of whether drifts occur in the data stream or not (Gama et al., 2014). We discuss algorithms from the point of view of both of these taxonomies.

Data Management and Forgetting Mechanisms

Many approaches to dealing with time changing streams involve the use of some sort of data management or forgetting mechanism. Data management strategies specify which data is used for learning, while forgetting strategies specify how old data are discarded. Both mechanisms are necessary to meet time and memory requirements posed by data streams and serve as a way of reacting to drifts by eliminating those examples that come from an old concept.

Online classifiers decide if an example will be included in the learning model on a per-instance basis. Such an approach promotes gradual adaptation to evolving concepts mainly by continuously updating the model with new examples. As an alternative, several classifiers apply *sliding windows* to keep the classifier consistent only with the most recent data. As sliding windows encompass a larger set of examples, they can be used to periodically build classifiers by conventional batch algorithms. From this point of view, this data management mechanism can be viewed as a general approach to transforming batch learners into classifiers for concept-drifting data streams.

The basic windowing algorithm is straightforward. Each example updates the window and later the classifier is updated by that window. The key part of this algorithm lies in the definition of the window, i.e., in the way it models the forgetting process. In the simplest approach, sliding windows are of fixed size and include only the most recent examples from the data stream. With each new data point the oldest example that does not fit in the window is discarded. More complex approaches vary the window size depending on, e.g., the indications of a drift detector (Bifet and Gavaldà, 2007).

Sliding windows are also one of the most popular forgetting mechanisms — examples that fall outside of the window are instantly excluded from

the model. From this perspective, two basic types of windows are defined: *sequence based*, where the size of a window is characterized by the number of instances, and *timestamp based*, where the size is defined by duration time.

There are two common alternatives to forgetting using sliding windows: *sampling* and *fading factors*. The first alternative aims at summarizing the characteristics of the data stream over a long period of time using a limited number of examples. One of the best known data stream sampling algorithms is reservoir sampling, which keeps a fixed-size sample of the stream that is updated with randomly selected instances (Aggarwal, 2007). Fading factors, on the other hand, provide a way of gradually forgetting examples. This is usually done with a decay function that assigns a weight to each example in the entire stream or a large window. Older examples receive smaller weights and are gradually treated as less important by the learner. Popular fading factors include linear, exponential, polynomial, and chordal functions.

Drift Detectors

Apart from sliding windows, another group of techniques that allow to construct an active stream classifier are *drift detectors*. Their task is to detect concept drift and alarm a base learner that its classifier should be rebuilt or updated. For example, when a detector signals a sudden change, an existing classifier can be discarded and replaced by a new one trained only on the most recent data.

Drift detectors are usually implemented using statistical tests based on *sequential analysis*, *process control charts*, or *monitoring differences between two distributions*. Detectors based on sequential analysis check whether the classification error calculated on the most recent instances is significantly different from its value calculated for range of older instances. Examples of sequential tests include CUSUM and the Page-Hinkley test (Gama, 2010). Drift detectors based on control charts take inspiration from statistical techniques used in quality control during product manufacturing. In these approaches, each prediction a classifier makes is treated as a Bernoulli trail. Then, the number of classification errors can be modeled with a Binomial distribution, which in turn can be tested for significantly improbable changes. Examples from this group include algorithms such as DDM, EDDM, and EWMA (Gama et al., 2014). Finally, several detection methods use two subsets of the stream: a reference window and a sliding window of the most recent examples. If the distributions over these two windows are significantly different, a change is signaled, suggesting that only examples from

the sliding window should be used to create a new model.

Single Classifiers

First proposals of stream classifiers concentrated on processing massive stationary data sets in constant time per example. Decision trees were one of the first algorithms to be adapted to meet these requirements using the Hoeffding bound. This bound states that with probability $1-\delta$, the true mean of a random variable of range R will not differ from the estimated mean after n independent observations by more than:

$$\epsilon = \sqrt{\frac{R^2 ln(1/\delta)}{2n}}. \tag{1}$$

Using the Hoeffding bound, Domingos and Hulten (2000) proposed a classifier called Very Fast Decision Tree (VFDT). This algorithm incrementally induces a tree from a massive data stream, without the need for storing examples after they have been used to update the tree. Its key idea is the selection of the split attribute, which is realized differently than in static trees (e.g., C4.5). Instead of selecting the best attribute (in terms of a split evaluation function) after viewing all the examples, VFDT uses the Hoeffding bound to calculate the number of examples necessary to select the right split-node with probability $1-\delta$. From the theoretical point of view, recent studies have shown that other bounds, as the McDiarmid inequality, are more suitable depending on the assumptions made about the distribution of values of the split evaluation function.

Many enhancements to the basic VFDT algorithm, often called the Hoeffding Tree, have been proposed. They include methods of limiting memory usage, the use of alternative bounds which requires less examples for each split node, approaches to dealing with numerical attributes, pruning mechanisms, and the use of sliding windows or drift detectors to adapt the algorithm to non-stationary settings (Gama, 2010). Nevertheless, the VFDT algorithm paved the way for many other learning algorithms that use the Hoeffding bound to incrementally process massive datasets (Ditzler et al., 2015).

Several traditional incremental classifiers were also adapted to computational and concept drift requirements. An illustrative example could be learning neural networks. By abandoning the epoch protocol and presenting examples in a single pass, neural networks can be adapted to changing data streams. Bayesian methods can also learn incrementally and require constant memory. To add a forgetting mechanism to this group of algorithms,

sliding windows are usually employed to "unlearn" the oldest examples. Similarly, nearest neighbor classifiers are naturally transformed to incremental versions with different techniques for selecting the limited subset of the most "useful" examples for accurate predictions. Rule-based algorithms were also adjusted to data stream environments, in fact, FLORA algorithms developed by Kubat and Widmer were one of the first classifiers to cope with concept drift (Deckert, 2013). Other algorithms use a structure similar to a decision tree to create rules and rule-specific drift detectors to react to changes (Kosina and Gama, 2015).

Ensembles

Ensembles are easily adapted to non-stationary data streams. Due to their modular construction they are capable of incorporating new data elements by introducing a new component into the ensemble, updating existing component classifiers, or changing weights in the aggregation phase. Ensembles are usually categorized into block-based and online approaches.

Most block-based ensembles periodically evaluate component classifiers with the newest data block and substitute the worst ensemble member with a new (candidate) classifier. Additionally, practically all proposed approaches work with fixed sized blocks. A generic block-based ensemble scheme is presented in Algorithm 1.

Algorithm 1 Generic Block-based Ensemble

Input: $\mathcal{S}$: data stream of examples partitioned into blocks of size d, k: number of ensemble members, $Q()$: classifier quality measure;
Output: $\mathcal{E}$: ensemble of k weighted classifiers

1: **for all** blocks $B_i \in \mathcal{S}$ **do**
2: build and weight candidate classifier C_c using B_i and $Q()$;
3: weight all classifiers C_j in ensemble $\mathcal{E}$ using B_i and $Q()$;
4: **if** $|\mathcal{E}| < k$ **then**
5: $\mathcal{E} \leftarrow \mathcal{E} \cup \{C_c\}$;
6: **else if** $\exists j : Q(C_c) > Q(C_j)$ **then**
7: replace weakest ensemble member with C_c;
8: **end if**
9: **end for**

For each block B_i, the weights of current component classifiers $C_j \in \mathcal{E}$ are calculated by a quality measure $Q()$, which depends on the particular algorithm. For instance, in Accuracy Weighted Ensemble (AWE), $Q()$ is

realized as a version of the mean square error of the component classifier C_j calculated on the recent block B_i, which is compared to the error of a random classifier on the same block (Wang et al., 2003). In addition to component re-weighting, a candidate classifier C_c is built from the recent block B_i and added to the ensemble if the ensemble's size is not exceeded. If the ensemble is full, the candidate classifier C_c substitutes the weakest ensemble member. It is worth noting that some algorithms, e.g., Learn++.NSE (Ditzler et al., 2015), do not limit the number of component classifiers in order to react to recurring concepts. The label prediction for new examples is usually based on a weighted majority vote of component classifiers. Most block-based ensembles take advantage of batch learning algorithms as component classifiers. This is not the case for hybrid algorithms, like the Accuracy Updated Ensemble (Brzezinski and Stefanowski, 2014), which updates classifiers after processing each block.

The origins of online stationary ensembles come from research on the Winnow algorithm and the Weighted Majority Algorithm (Littlestone and Warmuth, 1994), which combine the predictions of several experts (classifiers) by majority voting. When the ensemble misclassifies an instance, the weights of the wrong experts are decreased by a user-specified coefficient. The Dynamic Weighted Majority (DWM) is an extension of this idea for drifting data streams (Kolter and Maloof, 2007). It uses a set of incremental classifiers, which are generated by the same learning algorithm. When a new example is available, the final prediction is obtained as a weighted vote of all classifiers. The weights of all classifiers that misclassify the example are decreased in the same way as in the Weighted Majority Algorithm. However, DWM dynamically creates and deletes component classifiers in response to changes in classification performance. If the ensemble's overall prediction is incorrect, a new classifier is added to the ensemble.

Another group of online ensembles includes generalizations of static ensembles. The most well know are online versions of bagging and boosting (Oza and Russell, 2001). In case of online bagging the key idea is to adapt the bootstrap sampling step to a streaming setting. This is done by using single examples multiple times according to the Poisson distribution. This proposal of randomly updating training sets was an inspiration to develop several other approaches, e.g., Leveraging bagging, Online Boosting or the DDD ensemble (Ditzler et al., 2015).

Comprehensive reviews of various ensembles can be found in (Ditzler et al., 2015; Gama, 2010; Kuncheva, 2004).

Other Approaches

Although developing classifiers for concept drifting streams is in itself a non-trivial task, some other characteristics of learning problems can make this task even more difficult. In most current algorithms, it is assumed that all information, in particular class labels of instances, are complete, immediately available and received for free (Krempl et al., 2014). However, these assumptions may not hold true in some real-world problems, e.g. in fraud detection or patient health monitoring, where the labeling of examples is scarce or missing. In the case of static data these problems are studied with semi-supervised learning. For adapting such techniques to streams, the availability of at least some labeled data from the most recent distribution is required. For instance, (Masud et al., 2008) divide the stream into blocks containing partly labeled examples and then propose various approaches to combine learning ensemble classifiers with semi-supervised clustering. Active learning is also often related to semi-supervised frameworks. However, many sampling techniques developed for static data are not well suited for non-stationary streams (Spiliopoulou and Krempl, 2013). A review of recent active learning strategies is presented in (Žliobaitė et al., 2011).

A particularly challenging problem is learning classifiers from initially labeled non-stationary streams, where completely labeled examples are available for the first period only, followed by unlabeled data which may be drawn from a different distribution. Research on this topic is still at an early stage. Yet another problem is dealing with delayed information. In the case of *verification latency*, the class labels of preceding examples are not available before the subsequent instance has to be predicted. Therefore, feedback from correct predictions cannot be instantly used to improve the classifier. For a review of approaches that try to deal with this problem see (Ditzler et al., 2015).

Dealing with the class imbalance problem in non-stationary streams also introduces additional difficulties. Recent proposals to this problem pay attention to drifts of the minority class and specialized evaluation methods (Wang et al., 2015). The problem of class imbalance is also related to an increasing interest in studying other types of changes (Gama et al., 2014). Finally, other research concerns more complex representations of instances in streams, as graphs, semi-structured documents or text messages, as well as complex target outputs, like multi-labeled or ordinal classification. Other open issues are discussed in (Ditzler et al., 2015; Krempl et al., 2014).

Applications

Applications of stream classification can be organized into three groups: monitoring and control, information management, and analytics and diagnostics (Zliobaite et al., 2015).

Monitoring and control mostly relates to the detection of abnormal events. Domains from this group include sensor networks, telecommunications, traffic control, and fraud detection. Information management encompasses applications such as product recommendation, crime prediction, personalized search, and customer profiling. Analytics and diagnostics address domains like evaluation of creditworthiness, budget planning, or drug resistance prediction.

Each of the aforementioned groups differs also in the way stream classification is modeled. Monitoring and control usually involves sequential data where the task is to detect sudden changes. Information management is mostly based on relational data and gradual rather than abrupt changes are to be expected. Finally, diagnostic applications often involve recurring concepts. For an in-depth analysis of different application settings see Zliobaite et al. (2015).

Cross References

Classification, Concept Drift, Incremental Learning, Online Learning

Recommended Reading

Aggarwal, C. C., editor (2007). *Data Streams - Models and Algorithms*, volume 31 of *Advances in Database Systems*. Springer.

Bifet, A. and Gavaldà, R. (2007). Learning from time-changing data with adaptive windowing. In *Proceedings of the 7th SIAM International Conference on Data Mining*, pages 443–448.

Bifet, A., Holmes, G., Kirkby, R., and Pfahringer, B. (2010). Moa: Massive online analysis. *Journal of Machine Learning Research*, 11:1601–1604.

Brzezinski, D. and Stefanowski, J. (2014). Reacting to different types of concept drift: The accuracy updated ensemble algorithm. *IEEE Transactions on Neural Networks and Learning Systems*, 25:81–94.

Deckert, M. (2013). Incremental rule-based learners for handling concept drift: An overview. *Foundations of Computing and Decision Sciences*, 38(1):35–65.

Ditzler, G., Roveri, M., Alippi, C., and Polikar, R. (2015). Learning in nonstationary environments: A survey. *IEEE Computational Intelligence Magazine*, 10(4):12–25.

Domingos, P. and Hulten, G. (2000). Mining high-speed data streams. In *Proceedings of the 6th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 71–80.

Gama, J. (2010). *Knowledge Discovery from Data Streams*. Chapman and Hall/CRC.

Gama, J., Žliobaitė, I., Bifet, A., Pechenizkiy, M., and Bouchachia, A. (2014). A survey on concept drift adaptation. *ACM Computing Surveys*, 46(4):44:1–44:37.

Gomes, J. B., Gaber, M. M., Sousa, P. A. C., and Ruiz, E. M. (2014). Mining recurring concepts in a dynamic feature space. *IEEE Trans. Neural Netw. Learning Syst.*, 25(1):95–110.

Kolter, J. Z. and Maloof, M. A. (2007). Dynamic weighted majority: An ensemble method for drifting concepts. *Journal of Machine Learning Research*, 8:2755–2790.

Kosina, P. and Gama, J. (2015). Very fast decision rules for classification in data streams. *Data Mining and Knowledge Discovery*, 29(1):168–202.

Krempl, G., Žliobaitė, I., Brzezinski, D., Hüllermeier, E., Last, M., Lemaire, V., Noack, T., Shaker, A., Sievi, S., Spiliopoulou, M., and Stefanowski, J. (2014). Open challenges for data stream mining research. *SIGKDD Explorations*, 16(1):1–10.

Kuncheva, L. I. (2004). Classifier ensembles for changing environments. In *Proceedings of 5th International Workshop on Multiple Classifier Systems, MCS 04*, volume 3077 of *Springer LNCS*, pages 1–15.

Littlestone, N. and Warmuth, M. K. (1994). The weighted majority algorithm. *Inf. Comput.*, 108(2):212–261.

Masud, M., Gao, J., Khan, L., and Thuraisingham, B. (2008). A practical approach to classify evolving data streams: training with limited amount

of labeled data. In *Proceedings of the 8th IEEE International Conference on Data Mining*, pages 929–934.

Oza, N. C. and Russell, S. J. (2001). Experimental comparisons of online and batch versions of bagging and boosting. In *Proceedings of the 7th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 359–364.

Spiliopoulou, M. and Krempl, G. (2013). Tutorial mining multiple threads of streaming data. In *Proceedings of the Pacific-Asia Conference on Knowledge Discovery and Data Mining, PAKDD 2013*.

Žliobaitė, I., Bifet, A., Pfahringer, B., and Holmes, G. (2011). Active learning with evolving streaming data. In *Proceedings of the 2011 European Conference on Machine Learning and Knowledge Discovery in Databases*, volume 6913 of *Springer LNCS*, pages 597–612.

Wang, H., Fan, W., Yu, P. S., and Han, J. (2003). Mining concept-drifting data streams using ensemble classifiers. In *Proceedings of the 9th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 226–235.

Wang, S., Minku, L., and Yao, X. (2015). Resampling-based ensemble methods for online class imbalance learning. *IEEE Transactions on Knowledge and Data Engineering*, 27(5):1356–1368.

Zliobaite, I., Pechenizkiy, M., and Gama, J. (2015). An overview of concept drift applications. In Japkowicz, N. and Stefanowski, J., editors, *Big Data Analysis: New Algorithms for a New Society*. Springer. In press.