Machine learning for streaming data: state of the art, challenges, and opportunities

Heitor Murilo Gomes
University of Waikato
Hamilton, New Zealand
LTCI, Télécom ParisTech,
IP-Paris
Paris, France
heitor.gomes@waikato.ac.nz

Jesse Read
LIX, École Polytechnique
Palaiseau, France
jesse.read@polytechnique.edu

Albert Bifet
University of Waikato
Hamilton, New Zealand
LTCI, Télécom ParisTech,
IP-Paris
Paris, France
albert.bifet@waikato.ac.nz

Jean-Paul Barddal
PPGIA, Pontifical Catholic
University of Parana
Curitiba, Brazil
jean.barddal@ppgia.pucpr.br

João Gama
LIAAD, INESC TEC
University of Porto
Porto, Portugal
jgama@fep.up.pt

ABSTRACT

Incremental learning, online learning, and data stream learning are terms commonly associated with learning algorithms that update their models given a continuous influx of data without performing multiple passes over data. Several works have been devoted to this area, either directly or indirectly as characteristics of big data processing, i.e., *Velocity* and *Volume*. Given the current industry needs, there are many challenges to be addressed before existing methods can be efficiently applied to real-world problems. In this work, we focus on elucidating the connections among the current state-of-the-art on related fields; and clarifying open challenges in both academia and industry. We treat with special care topics that were not thoroughly investigated in past position and survey papers. This work aims to evoke discussion and elucidate the current research opportunities, highlighting the relationship of different subareas and suggesting courses of action when possible.

1. INTRODUCTION

Data sources are becoming increasingly ubiquitous and faster in comparison to the previous decade. These characteristics motivated the development of several machine learning algorithms for data streams. We are now on the verge of moving out these methods from the research labs to the industry, similarly to what happened to traditional machine learning methods in the recent past. This current movement requires the development and adaptation of techniques that are adjacent to the learning algorithms, i.e., it is necessary to develop not only efficient and adaptive learners, but also methods to deal with data preprocessing and other practical tasks. On top of that, there is a need to objectively reassess the underlying assumptions of some techniques developed under hypothetical scenarios to clearly understand *when* and *how* they are applicable in practice.

Machine learning for streaming data research yielded several works on supervised learning [19], especially classification, mostly focused on addressing the problem of changes to the underlying data distribution over time, i.e., concept drifts [119]. In general, these works focus on one specific challenge: develop methods that maintain an accurate decision model with the ability to learn and forget concepts incrementally.

The focus given to supervised learning has shifted towards other tasks in the past few years, mostly to accommodate more general requirements of real-world problems. Nowadays one can find work on data stream clustering, pattern mining, anomaly detection, feature selection, multi-output learning, semi-supervised learning, novel class detection, and others. Nevertheless, some fields were less developed than others, e.g., drift detection and recovery has been thoroughly investigated for streaming data where labels are immediately (and fully) available. In contrast, not as much research has been conducted on the efficiency of drift detection methods for streaming data where labels arrive with a non-negligible delay or when some (or all) labels never arrive (semi-supervised and unsupervised learning).

Previous works have shown the importance of some of these problems and their research directions. Krempl et al. [73] discuss important issues, such as how to evaluate data stream algorithms, privacy and the gap between algorithms to full decision support systems. Machine learning for data streams is a recurrent topic in Big Data surveys [44; 127] as it is related to the Velocity and Volume characteristics of the traditional 3V's of big data (Volume, Variety, Velocity). Nevertheless, there is no consensus about how learning from streaming data should be tackled, and depending on the application (and the research group) different abstractions and solutions are going to be used. For example, Stoica et al. [115] discuss continual learning, and from their point of view, learning from heterogeneous environments where changes are expected to occur is better addressed by reinforcement learning. In summary, some reviews and surveys focus on specific tasks or techniques, such as rule mining for data streams [66], activity recognition [1] or ensemble learning [53].

In this paper, we focus on providing an updated view of the field of machine learning for data streams, highlighting the state-of-the-art and possible research (and development) opportunities. Our contributions focus on aspects that were not thoroughly discussed before in similar works [73], and thus, when appropriate, we direct the readers to works that better introduce the original problems while we highlight more complex challenges.

In summary, the topics discussed are organized as follows. The first three sections of the manuscript address the important topics of preprocessing (section 2), learning (section 3), and adaptation (section 4). In the last three sections, we turn our attention to evaluating algorithms (section 5), streaming data and related topics in AI (section 6), and existing tools for exploring machine learning for streaming data (section 7). Finally, section 8 outlines the main takeaways regarding research opportunities and challenges discussed throughout this paper.

2. DATA PREPROCESSING

Data preparation is an essential part of a machine learning solution. Real-world problems require transformations to raw data, preprocessing steps and usually a further selection of 'relevant' data before it can be used to build machine learning models.

Trivial preprocessing, such as normalizing a feature, can be complicated in a streaming setting. The main reason is that statistics about the data are unknown *a priori*, e.g., the minimum and maximum values a given feature can exhibit. There are different approaches to scale and discretize features as discussed in the next subsections; still, as we move into more complex preprocessing, it is usually unknown territory or one which has not been explored in depth.

There are mainly two reasons for performing data preprocessing before training a machine learning model:

1. To allow learning algorithms to handle the data;
2. To improve learning by extracting or keeping only the most relevant data.

The first reason is more restrictive, as some algorithms will not be able to digest data if it is not in the expected format, i.e., the data types do not match. Other algorithms will perform poorly if the data is not normalized. Examples include algorithms that rely on Stochastic Gradient Descent and distance-based algorithms (such as nearest neighbors).

Feature engineering governs the second aspect, as accurate machine learning solutions often rely on a well-thought feature transformation, selection or reduction of the raw data.

In a batch learning pipeline, preprocessing, fitting and testing a model are distinct phases. They are applied in order, and the output of this process is a fitted model that can be used for future data. These same operations are required in a streaming setting. The main difference is that streaming data needs the continuous application of the whole pipeline while it is being used. Consequently, all these phases are interleaved in an online process, which requires an intricate orchestration of the pipeline and ideally does not rely on performing some tasks offline.

García et al. [51] discussed the challenges concerning preprocessing techniques for big data environments, focusing on how different big data frameworks, such as Hadoop, Spark, and Flink deal with them and which methods are implemented for a wide range of preprocessing tasks, including: Feature selection, instance selection, discretization, and others. Ramírez-Gallego et al. [103] focus specifically on preprocessing techniques for data streams, mentioning both existing methods and open challenges. Critical remarks were made in [103], such as: the relevance of proposing novel discretization techniques that do not rely solely on quantiles and that perform better in the presence of concept drifts; expanding existing preprocessing techniques that deal with concept drift to also account for recurrent drifts; and the need for procedures that address non-conventional problems, e.g., multi-label classification.

In this section, we focus on issues and techniques that were not thoroughly discussed in previous works, such as summarization sketches. Other aspects that can be considered as part of preprocessing, notably dealing with imbalanced data, and others, are discussed in further sections.

2.1 Feature transformation

2.1.1 Summarization Sketches

Working with limited memory in streaming data is non-trivial, since data streams produce insurmountable quantities of raw data, which are often not useful as individual instances, but essential when aggregated. These aggregated data structures can be used for further analysis such as: *"what is the average age of all customers that visited a given website in the last hour?"* or to train a machine learning model.

The summary created to avoid storing and maintaining a large amount of data is often referred to as a 'sketch' of the data. Sketches are probabilistic data structures that summarize streams of data, such that any two sketches of individual streams can be combined into the sketch of the combined stream in a space-efficient way. Using sketches requires a number of compromises: sketches must be created by using a constrained amount of memory and processing time, and if the sketches are not accurate, then the information derived from them may be misleading.

Over the last decades, many summarization sketches have been proposed. Ranging from simple membership techniques such as Bloom filters [22] and counting strategies to more advanced methods such as CM-Sketch [30], and ADA-Sketches [112]. Bloom filters are probabilistic data structures used to test whether an element is a member of a set, using hash functions. CM-Sketch is essentially an extension of Bloom filters used to count the frequency of different elements in a stream.

Sketches are becoming a popular approach to cope with data streams [4; 108; 128]. In the previous decade, the adoption of sketching for stream processing was taken with some skepticism, for example, Gaber et al. [45] suggests using dimensionality reduction techniques, such as Principal Components Analysis, as a more sustainable approach for stream processing.

Novel sketches have been proposed based on the idea of building a meta-sketch using several sketches as components. The Slim-Fat Sketch [128] (SF-Sketch) is an example of this, that outperforms single sketches.

Sketches can also be used in machine learning methods for data streams. For example, the Graphical Model Sketch [75] is a sketch used inside Bayesian networks, or in the Naive Bayes classifier, to reduce the size of memory used. How to use sketches inside other machine learning methods is an open question.

2.1.2 Feature Scaling

Feature scaling consists of transforming the features domain in a way that they are on a similar scale. Commonly, scaling refers to normalizing, i.e., transform features such that their mean $\hat{x} = 0$ and standard deviation $\sigma = 1$. In batch learning, feature scaling is both an important and an uninteresting topic, which is often added to the data transformation pipeline without much thought of the process. It is important while fitting learners that rely on gradient descent (e.g., neural networks) as these will converge faster if features are about the same scale; or learners that rely on distances among instances (e.g. k-means, k-nearest neighbors, and others) as this prevent one dimension with a wide range of values dominating others when calculating distances. The two most popular approaches consist of i) centralizing the data by subtracting the mean and diving by the standard deviation; or ii) dividing each value by the range (max - min).

It is unfeasible to perform feature scale for data streams as aggregate calculations must be estimated throughout the execution. For landmark window approaches there are exact solutions to incrementally computing the mean and standard deviation without storing all the points seen so far. We need to maintain only three numbers: the number of data points seen so far n, the sum of the data points $\sum(x)$, and the sum of the squares of the data points $\sum(x^2)$. These statistics are easy to compute incrementally. The mean is given by $\frac{\sum(x)}{n}$ and the variance is given by $\frac{\sum(x^2)-(\sum(x)^2)/n}{n-1}$. In landmark windows, its easy and fast to maintain exact statistics by storing few numbers. However, in time-changing streams the adaptation is too slow. To deal with change, sliding window models are more appropriate. The problem is that exact computation of the mean or variance over a stream in a sliding window model requires to store all data points inside the window. Approximate solutions, using logarithmic space, are the exponential histograms [31]. Exponential histograms store data in buckets of exponential growing size: $2^0, 2^1, 2^2, 2^3, \ldots$. For a window size W, only $log(W)$ space is required. Recent data are stored in fine granularity, while past data are stored in an aggregated form. The statistics computed using exponential histograms are approximate, with error bounds. The error comes from the last bucket, where it is not possible to guarantee all data is inside the window.

The lack of attention on feature scaling by the data stream mining research community may be justified by the pervasiveness of the Hoeffding Tree algorithm [37]. Hoeffding trees maintain the characteristic of conventional decision trees of being resilient to variations in the features range of values. Even if there is not much room for theoretical advances for data stream feature scaling, practical implementations would be welcome by the community (see section 7). The most immediate challenge is to provide efficient implementation of these feature scaling methods that integrate with other operators (e.g. drift detection) in the streaming machine learning pipeline. Finally, another aspect is related to how some learning algorithms were tested on datasets where features were scaled in an offline process. This certainly affects the results obtained (see section 5), and an online transformation would provide more realistic results.

2.1.3 Feature Discretization

Discretization is a process that divides numeric features into categorical ones using intervals. Depending on the application and predictive model being used, discretization can bring several benefits, including faster computation time as discrete variables are usually easier to handle compared to numeric ones; and decreases the chances of overfitting since the feature space becomes less complex.

Targeting feature discretization from data streams, a significant milestone was the Partition Incremental Discretization algorithm (PiD) [101]. PiD discretizes numeric features in two layers. The first layer is responsible for computing a high number of intervals given the arriving data, while the second uses the statistics calculated in the first layer to compute equal-frequency partitions.

Webb [122] proposed two different schemes for feature discretization: Incremental Discretization Algorithm (IDA) and IDAW (where W stands for windowing). IDA uses quantile-based discretization on the entire data stream using random sampling, while IDAW maintains a window of the most recent values for an attribute and discretizes these. IDAW requires more computational time than IDA since it must be updated more frequently.

The ChiMerge discretization algorithm [77] store the features' values on a binary search tree, which makes it more robust to noise in comparison previous methods.

Pfahringer et al. [99] compared a range of discretization schemes for Hoeffding Trees. Based on empirical evaluations, the Gaussian approximation was indicated as the most accurate method in terms of accuracy and tree growth.

Finally, similarly to feature scaling, the effort on feature discretization should target the provisioning of efficient implementations that integrate with different parts of the streaming process, such as classification systems, drift detection, and evaluation [40].

2.2 Invalid entries handling

Invalid entries may refer to missing, noise or other issues that may arise (e.g. unknown formats). Characterizing what is an invalid entry depends on the problem, algorithms, and software. For example, categorical features may be deemed as invalid in many machine learning tools (e.g., scikit-learn), not because they are inherently wrong, but because the software was designed in a way that does not account for them.

Despite technical issues related to invalid entries, the most well known, and studied, problem is missing data. Krempl et al. [73] commented on the relevance of this problem and discussed missing values for the output feature as part of this discussion. We prefer to include this latter case under our discussion of semi-supervised learning (see section 3.2) and solely concentrate on the input data in this section.

To address missing values imputation methods are relatively standard in batch learning [38]. Imputation methods have not been thoroughly investigated for streaming data yet. This is mostly because the techniques often rely on observing the whole data before imputing the values. These techniques are 'feasible' in batch learning, but not for streaming data. For example, mean and median imputation will encounter issues such as: How to estimate the mean and the median for evolving data? The issues with mean estimation were previously discussed in section 2.1.2.

An option to avoid aggregation calculations is to apply imputation by using a learner. For example, a windowed K-nearest neighbors can be used, such that the k neighbors values can be used to infer the value of a missing feature in a given instance.

2.3 Dimensionality reduction

Dimensionality reduction tackles the retention of patterns in the input data that are relevant to the learning task. We report the works and gaps on dimensionality reduction techniques that apply transformations to the input data, e.g., Principal Component Analysis (PCA), and Random Projections; while the next section discusses feature selection techniques tailored for data streams and their shortcomings. Mitliagkas et al. [86] introduced a memory-limited approximation of PCA based on sampling and sketches that can be calculated under reasonable error bounds. Yu et al. [129] proposed a single-pass randomized PCA method, yet, the method has been solely evaluated on a single image dataset. Zhou et al. [133] presented an incremental feature learning algorithm to determine the optimal model complexity for online data based on the denoising autoencoder.

Another set of techniques that are important for dimensionality reduction are those tailored for text data. A notable implementation of dimensionality reduction in such scenarios is the hashing tricks provided in Vowpal Wabbit [76]. Hashing tricks facilitate the processing of text data as conventional Bag-of-Words and n-grams are unappealing for streaming scenarios since the entire lexicon for a domain must be known before the learning process starts, which is an assumption that is hardly met in streaming domains, as out-of-vocabulary words may appear over time. With the hashing trick, each word (feature) in the original text data is converted into a key using a hash function, which is used to increment counters in a reduced dimensionality space that is later fed to the learner. Such a process has a vital downside as reverse lookups are not possible, i.e., if one wants to determine which words are the most important for predictions.

Finally, Pham et al. [100] proposed an ensemble that combines different random projection techniques (Bernoulli, Achlioptas, and Gaussian projections) with Hoeffding Trees, and results have shown that the technique is feasible and competitive against bagging methods when applied to real-world data.

Further investigation is necessary for all the techniques discussed in this section, specifically to investigate the effect of concept drifts. For instance, most of these methods are single-pass, yet, they have been applied to datasets in a batch processing scheme. In real-world streaming scenarios, drifts in the original data will induce changes in the feature transformation outputs of random projections, so closer analysis is required to investigate how classifiers behave according to such changes.

2.4 Feature selection

Feature selection targets the identification of which features are relevant to the learning task. In contrast to dimensionality reduction techniques, feature selection does not apply transformations to the data, and thus, the features can still be interpreted. As a by-product, feature selection is also known in batch machine learning for potentially improving the computation time, reducing computation requirements, and enhancing the generalization rates of classification systems as these are less prone to overfitting.

Feature selection methods tailored for batch settings require the entire dataset to determine which features are the most important according to some goodness-of-fit criterion. Nevertheless, this is a requirement that does not hold in streaming scenarios, as new data becomes available over time. Targeting data streams, Barddal et al. [10] showed that hoeffding (decision) trees [37] and decision rules [71] are the major representatives of classification and regression systems that can incrementally identify which features are the most important.

Incrementally identifying which features are important is a relevant subject in data streams. New methods must be designed so that the feature selection process can identify and adapt to changes in the relevance of features, a phenomenon called *feature drift* (see section 4.2).

Another critical gap of feature selection in streaming scenarios regards the evaluation of feature selectors. There are different factors to account for when evaluating feature selection proposals. Throughout the years, different quantitative measures, such as accuracy and scalability; and subjective ones, such as "ease of use", have been used to highlight the efficiency of feature selectors [47]. First, it is crucial to assess the behavior of feature selection algorithms when combined with different learners, as each learner builds its predictive model differently despite being fed with the same subset of features. On the other hand, it is important to make sure that the feature selection process is accurate, i.e., the selected subset of features matches the features that are indeed relevant [47].

Finally, feature selectors are expected to be "stable", meaning that they should select the same features despite being trained with different subsets of data [91]. In batch learning, stability metrics target the measuring of whether the selected subset of features across different data samples of the same distribution match [74]. Stable methods are preferred as they facilitate learning a model from the data, i.e., the subset of features is fixed. However, an open challenge in the streaming setting is the contradiction between feature stability and selection accuracy. If the features' importance shifts over time (feature drifts) then the feature selection method will need to compromise either stability or accuracy.

3. THE LEARNING PROCESS

Learning from streaming data requires real-time (or near real-time) updates to a model. There are many important aspects to consider in this 'learning phase', such as dealing with verification latency. In this section, we discuss the relationship between data streams and time series; the problem of dealing with partially and delayed labels; ensemble learning; imbalanced data streams and the essential issue of detecting anomalies from streaming data.

3.1 Time series

Time series data may commonly arrive in the form of online data, and it thus can be treated as a data stream. Another way of seeing it: data streams may often involve temporal dependence and thus be considered as time series. A comparison of data streams and time series methods is given in

Žliobaite et al. [121]. It was pointed out that many benchmark datasets used in data streams research exhibit time series elements, as exemplified in Fig. 1.

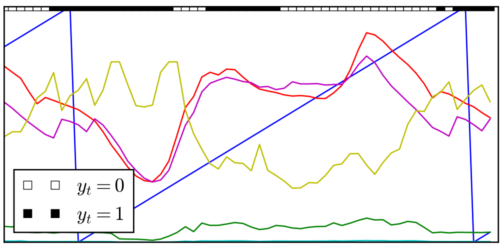

Figure 1: A small section of the well-known Electricity dataset; a common benchmark in data stream evaluations. A time series nature is clearly visible with regard to temporal dependence, both in the features (plotted in solid lines) and the class labels (shown above).

Unlike a regular data stream, where instances are assumed to be independently and identically distributed[1] (i.i.d.), data points in a time series are expected to exhibit strong temporal dependence.

Data stream methods can be adapted to such scenarios using relatively simple strategies, such as aggregating the labels of earlier instances into the instance space of the current instance [121]. There are special considerations in terms of evaluation under concept drifting streams, in particular regarding the selection of benchmark classifiers (see, again, Žliobaite et al. [121]). Further discussion on evaluation strategies is given in section 5.

Considering a moving window of instances can be viewed as *removing* time dependence, and thus converting a time series into an ordinary data stream. If $P(y_t|x_t, x_{t-1}) = P(y_t|x_t)$ does *not* hold, it indicates temporal dependence in the data stream. The idea is to produce a new stream of instances $x'_t := [x_t, x_{t-1}, \dots, x_{t-\ell}]$ over a window of size ℓ, sufficient such that $P(y_t|x'_t) = P(y_t|x'_t, x'_{t-1})$; thus producing a temporally-independent (i.e., 'regular') data stream.

It is also possible to use a memory device to embed an arbitrarily long series into such an instance representation, for example, by using an echo state network or another kind of recurrent neural network (see section 6.2 for further consideration in the context of streams). An experimentation of such an approach in data streams was carried out in [83] under Hoeffding tree methods.

We can note that the filtering task of sequential state-space models, such as the hidden Markov model (HMM), are directly applicable to classification in data streams. Indeed, one can see HMMs as a sequential version of naive Bayes (a common data-streams benchmark), simply by modelling; see Fig. 2 for a graphical intuition. Kalman filters and particle filters can similarly be considered under the continuous output (i.e., regression) scenario. See [8; 41] for a comprehensive overview of these methods.

We do remark that, unlike in a typical scenario of these models, learning cannot be done on a full forward-backward pass

[1]The 'identically *distributed*' assumption may be relaxed in the presence of concept drift; but in any case we may say i.i.d. with regrad to a particular concept.

$$\begin{tikzcd}
y_1 \arrow[r] \arrow[d] & y_2 \arrow[r] \arrow[d] & y_3 \arrow[r] \arrow[d] & y_4 \arrow[d] \\
x_1 & x_2 & x_3 & x_4
\end{tikzcd}$$

Figure 2: A probabilistic graphical model representation of a generative model (such as a hidden Markov model, where temporal dependence is considered; exemplified over four time points.

(such as using the Baum-Welch algorithm [13]) because in a stream there is no fixed end to the sequence and predictions are needed at the current timestep, not retrospectively. It can, however, be done over a window, and we are not aware of any work that considers explicitly this in the data-stream context – an open challenge.

The open challenge relating to data stream learning is to draw more solid links to the existing time series literature. On the theoretical level, connections between models in the respective areas should be formalized, thus clarifying which time series models are applicable in data streams and in which contexts. An empirical comparison of methods across these domains is needed, not only to reveal the extent of time series nature within standard data sources found in the time series literature but also indicate on the practical applicability of methods from the rich and diverse time-series literature. Undoubtedly, techniques (in particular those of drift detection), could also be used to enhance time-series methods.

3.2 Semi-Supervised learning

Semi-supervised learning (SSL) problems are challenging, appear in a multitude of domains, and are particularly relevant to streaming applications[2] where data are abundant, but labeled data may be rare. To address SSL problems, one can either ignore the unlabeled data and focus on the labeled data; try to leverage the unlabeled data; or assume some labels are available per request (active learning). The first implies a supervised problem; the second relies on finding and exploring a specific characteristic of the data; while the third depends on an external agent to provide the required labels on time.

In this section we focus the discussion on the last two options, leveraging unlabeled data and active learning. Still, it is essential to consider the first option, supervised learning, for practical applications as discussed by Oliver et al. [92], since a robust supervised model trained only on the labeled data may outperform intricate semi-supervised methods. On top of that, active learning might not always be feasible as labeling the data can be costly financially and in terms of time.

Even for supervised problems, it is reasonable to assume that immediately labeled data will not be available. For example, in a real-world data stream problem it is often the case that the algorithm is required to predict x and only after several time units its true label y is going to be available. This problem setting leads to a situation where verification

[2]Also referred to as 'partially labelled streams' or 'infinitely delayed stream'.

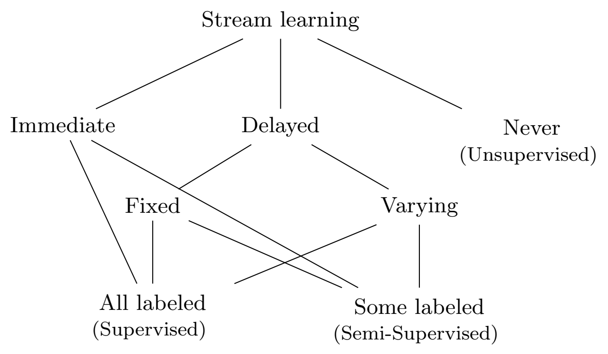

Figure 3: Stream learning according to labels arrival time [54].

latency, or delay, takes place. One can assume a delayed labeled stream as an SSL problem and ignore that some (or all) of the labels will be available at some point in the future. Therefore delayed labeled streams can be tackled with SSL techniques. Figure 3 presents a categorization of how supervised, semi-supervised and verification latency are related w.r.t the label arrival. Plasse and Adams [102] introduce a taxonomy to determine the delay mechanism and magnitude; present real-world applications where delayed labels occur, e.g., credit scoring; notation for the delayed labeling setting; and how the set of delayed labels can be used to pre-update the classifier. However, Plasse and Adams [102] do not introduce a specific evaluation procedure that accounts for verification latency.

Žliobaite [134] raise the critical questions of *if* and *when* it is possible to detect a concept drift from delayed labeled data. The work is motivated by a large number of real-world problems in which labels are delayed due to the problem characteristics. For example, the ground truth for credit default prediction can only be obtained several months, or years, after a decision was made. The work also discusses the relationship between delayed labeling and active learning. The former concerns *when* new labels are needed, while the latter is related to *which* instances must be labeled to minimize cost and maximize prediction performance. It was concluded that both problems are complementary [134].

The SSL techniques for streaming data includes unsupervised learning combined with supervised learning, e.g., clustering the unlabeled data and pseudo-labeling it based on the clusters and existing labeled data [109; 85; 64]; active learning [29]; and hybrid approaches [95]. Each of these approaches makes assumptions about the problem and the data distribution, but not very often these assumptions are explicitly discussed.

Active learning is a popular choice for streaming data. Active learning promises to reduce the amount of required labeled data in supervised learning to achieve a given level of predictive performance. In simple terms, the algorithm decides which instances should be labeled given some criteria. There are a few concerns regarding this approach, such that it presumably assumes that any instance can be labeled, which may not be true (it depends on the domain); and it includes an outsider (usually a human) in the learning process, i.e., someone who is going to provide the labels required by the algorithm. Assuming a traditional evolving stream setting, by the time the label is provided the concept has already changed or the volume of data to be labeled exceeds the capabilities of the responsible for labeling. For example, assuming an algorithm performs well with 5% labeled data, however, to label 5% of a data stream that generates thousands of instances per day is still a difficult, and costly, task in a variety of domains.

The amount of literature concerning how to exploit unlabeled instances and how to deal with verification latency has increased in the past years. Still, open issues include: how to effectively evaluate algorithms when labels arrive with delay; how to deal with out-of-order data [78]; and fundamental theoretical aspects behind existing SSL methods proposed for non-stationary problems. On top of that, some strategies developed for batch data have not been thoroughly explored in a streaming scenario, including multi-view learning and co-training [23]; and transductive support vector machines [67; 113]. Finally, transfer learning is a somewhat popular method to alleviate the problem of few labeled instances [94], and it has not been widely adopted for the streaming setting yet.

3.3 Ensemble learning

Ensemble learning receives much attention for data stream learning as ensembles can be integrated with drift detection algorithms and incorporate dynamic updates, such as selective removal or addition of base models [53]. On top of that, several issues such as concept evolution, feature evolution, semi-supervised learning, anomaly detection, are often approached with an ensemble approach for data streams.

The most common use of ensemble models is to allow **recovery from concept drifts**. Ensemble models can rely on reactive or active strategies to cope with concept drift. **Reactive** strategies continuously update the ensemble, often assigning different weights to base models according to their prediction performance. This weighting function may take into account the recency of prediction mistakes, such that correct predictions in the latest instances receive a higher weight in comparison to correct predictions in oldest instances. Examples of these strategies include the Streaming Ensemble Algorithm (SEA) [116] and Dynamic Weighted Ensemble (DWM) [70].

A canonical example of an **active strategy** is ADWIN Bagging, i.e., the combination of ADWIN [17] and online bagging [93]. In ADWIN Bagging, each base model classification output is monitored by an ADWIN instance and whenever a drift is flagged the corresponding model is reset. Recently, ensembles that combine reactive and active strategies have been proposed in the literature. Examples include the Adaptive Random Forest (ARF) [54] and the Streaming Random Subspaces (SRP) [57], which are adaptations of the Random Forest [24] and Random Patches algorithms [82], respectively, to streaming data with the addition of drift detectors and weighting functions based on models prediction performance. The main difference between ARF and SRP is that ARF is based on local subspace randomization and SRP uses a global subspace randomization strategy. In [57] authors showed that the global subspace strategy is a more flexible model, which increases diversity among base models and the overall accuracy of the ensemble.

An ensemble-based method is often used along with other techniques to address **concept evolution**. Even though the ensemble may not be used directly to detect the novel classes, it is useful to dynamically incorporate the novel class instances into the whole learning model without major changes to the already learned models, i.e., other ensemble members. For instance, ensemble approaches combined with One-Versus-All (OVA) approach to address concept evolution includes OVA Decision Trees [63], Learn^{++}.NC and Learn^{++}.UDNC [35].

Ensemble strategies have been used to address **feature drift** problems by removing/adding the influence of specific features by using single-feature classifiers, such that if a feature disappears or is identified as irrelevant, its influence can be wholly removed from the whole system by removing the classifier associated with it. This approach is similar to that mentioned previously to cope with concept evolution (one classifier per class), and it is the approach used in HSMiner [96], with the addition of using different classifiers according to the feature domain. On top of that, using single-feature classifiers, or a limited size of features per learner that improves the algorithm's scalability as its processing can be distributed among multiple machines using a map-reduce approach [61].

The flexibility that an ensemble strategy allows (i.e., add and remove models) make it an attractive strategy to deal with **partially labeled (i.e., semi-supervised) streams**. An example is the SluiceBox AnyMeans (SluiceBoxAM) algorithm [97]. SluiceBoxAM is based on the SluiceBox algorithm [95], which already combined different methods to address problems, such as multi-domain features, concept drift, novel class detection, and feature evolution. Besides using a clustering method (AnyMeans) capable of discovering non-spherical clusters SluiceBoxAM can be used with other ensemble classifiers, e.g., Parker and Khan [97] report the performance of SluiceBoxAM combined with the leveraging bagging algorithm [21]. The overall idea behind SliceBoxAM and other ensemble methods combined with clustering methods.

Open issues related to the deployment of ensemble methods to practical streaming data include overly complex models and massive computational resources demands. Some algorithms are too complex as they contain different learning strategies and heuristics to cope with various problems. While addressing different problems is a good trait, these models are often too complicated for a framework developer to understand all of their idiosyncrasies, which make them less attractive to be implemented and deployed in practice. On top of that, as discussed by Gomes et al. [53] the combination of too many heuristics raises the question: "Does the ensemble perform well because of the combination of all its methods, or simply because some of them are very effective, while others are effectively useless or even harmful?"

To address these questions, it is important to present in-depth analysis of how each of the strategies embedded into the ensemble behaves individually and in combination with the others. This requires creating experiments that are beyond measuring the overall ensemble classification performance. In general, it is difficult to isolate aspects of the method for comparisons, but it is worthwhile to verify if it is possible when proposing a novel method, especially if it lacks theoretical guarantees.

The computational resources used by a machine learning algorithm developed for data streams are of critical importance. An accurate, yet inefficient method might not be fit for use in environments with strictly limited resources. Some ensemble algorithms approach this problem by removing redundant models when their current predictions are too similar [55; 56], or their coverage overlaps during training [109]. These techniques may not enhance the ensemble performance from the learning performance perspective; in fact they might negatively impact it. Algorithms that solve more problems are often more challenging to manage, for example, algorithms that combine clustering and ensembles to address partially labeled streams. One approach is to distribute the computation using multiple threads [54]. However, there are limits to what can be accomplished with algorithms executed in a single machine, even if they are multi-threaded. As a consequence, the machine learning community is investing efforts into scalable and distributed systems.

The challenge is how to maintain the characteristics of the ensemble methods and efficiently distributed them over several machines. Some ensemble methods are straightforward to be adapted to distributed environments (e.g., bagging), while others are more complicated (e.g., random forests). Efforts have been driven towards integrated platforms for stream learning in this context, which resulted in frameworks (or libraries) such as Apache Scalable Advanced Massive Online Analysis (SAMOA) [33] and StreamDM.

Currently, there are efforts in deploying stream learning algorithms (ensembles included) in a distributed setting. Examples include the Streaming Parallel Decision Tree [14], HSMiner [61] and Vertical Hoeffding Tree (VHT) [72]. Ensembles are attractive techniques, as discussed in this section, and they are probably going to play an essential role in stream processing software, such as Apache Spark Streaming [130] and Apache Flink [25].

3.4 Imbalanced Learning

Imbalanced datasets are characterized by one class outnumbering the instances of the other one [80]. The later is referred to as the minority class, while the former is identified as the majority class. These concepts can be generalized to multi-class classification and other learning tasks, e.g., regression [118]. The imbalance may be inherent to the problem (intrinsic) or caused by some fault in the data acquisition (extrinsic). Learning from imbalanced datasets is challenging as most learning algorithms are designed to optimize for generalization, and as a consequence, the minority class may be completely ignored.

The approaches for dealing with imbalanced datasets commonly rely on **cost-sensitive learning**; **resampling methods** (oversampling and undersampling); and **ensemble learning**. Cost-sensitive strategies rely on assigning different weights to incorrect predictions. This can be used to increase the cost of a minority class error, which shall 'bias' the learning process in its favor. Resampling methods rely on removing instances from the majority class (undersampling) or creating synthetic instances for the minority class (oversampling). These methods tend to be costly even for batch learning, as in general, they require multiple distance computations among training instances, e.g., SMOTE [26]. Finally, ensemble strategies for imbalanced learning use the ensemble structure alongside cost-sensitive learning or resampling strategies [46].

Besides the issues related to the dataset imbalance, in a streaming scenario, other challenges may arise. For example, given two classes labels which distribution is balanced, for a given period one of them may be underrepresented; thus leading to a 'temporary' imbalance. Another possibility is that the class distribution variations indicate a concept drift (e.g., a period of transition in a gradual drift) or perhaps a concept evolution (e.g., one of the classes is disappearing). How to differentiate between these situations and propose general strategies to address them is still an open issue. This motivates the development of methods to address imbalanced streaming data; examples include: Learn++.NSE [42]; SMOTE [36]; REA [27]; and an adapted Neural Network [52]. Finally, another challenge is how to develop algorithms that are effective in addressing the imbalance problem, without compromising the computational resources. To this end, the cost-sensitive and ensemble strategies seems to be more effectively than the resampling strategies (specially oversampling).

3.5 Anomaly detection

A significant task in imbalanced learning is that of anomaly detection. Supervised anomaly detection, where labeled normal examples and anomalies are part of the dataset, is indistinguishable from a standard imbalanced learning problem – where the anomalous instances belong to the underrepresented (minority) class. However, in most practical scenarios, it is not feasible to get verified labels, particularly for non-stationary data streams. Therefore, in a real scenario, one might need to choose between unsupervised learning and semi-supervised learning.

For the sake of generality, many methods assume that no labeled data is available, and the problem is tackled essentially as a clustering or density-estimation task. In this case, instances 'too far' from the centers of established clusters, or densities, are considered anomalies. Existing clustering algorithms for streaming data, such as CluStream [3] can be used for this purpose. However, a challenge is that some of these methods rely on an offline step where the actual clustering method, e.g., k-means, is executed. The online step is responsible only for updates to the data structures that summarize the incoming data. Salehi and Rashidi [110] presents a recent survey on anomaly detection for evolving data, with a particular focus on unsupervised learning approaches.

In some scenarios, a small set of labeled normal instances and anomalies is available. This case can neither be characterized as supervised nor unsupervised, but as semi-supervised learning (see section 3.2). In a semi-supervised anomaly detection it is assumed that some normal examples and anomalies will not be labeled, but besides that, some anomalies might not even be known beforehand, while others might cease to exist altogether. The critical aspect in such scenario is the evolution of the labels overtime. This can take the form of **adversarial machine learning** [65], where an adversarial opponent actively attempt to jeopardize the learning model using several types of attacks[3]. Furthermore, a problem where labels appear and disappear overtime can be formulated as an evolving concept (or novel class detection) problem [88] (see section 4).

We highlight the need for further discussion around the intersection among anomaly detection, adversarial learning, semi-supervised learning, and novel class detection. Finally, some of the latest proposed algorithms for data stream anomaly detection are RS-Forest [125], Robust Random Cut Forest Based [59], Threaded Ensemble of Autoencoders [39], and OnCAD [28].

[3]Barreno et al. [12] presents a taxonomy of such attacks.

4. REASONING ABOUT THE LEARNING PROCESS

Learning from data streams is a continuous process. The learning systems that act in dynamic environments, where working conditions change and evolve, need to monitor their working conditions. They need to monitor the learning process for change detection, emergence of novel classes, changes in the relevance of features, changes in the *optimal* parameters settings, and others. The goal of this section is to discuss the design aspects of learning systems that can monitor their performance. In a general sense, these learning systems should be able of self-diagnosis when performance degrades, by identifying the possible causes of degradation and self-repairing or self-reconfiguring to recover to a stable status.

Much research has been devoted to characterizing concept drift [123], detecting and recovering from it [50], recurrent concepts [48; 49]. Since this is a frequent topic when discussing learning from data streams, we refrain from reviewing it entirely and focus mostly on current issues related to it, such as feature drifts and their relationship to feature selection; drift detection under unsupervised/semi-supervised and delayed labeled scenarios; and hyper-parameter tuning.

4.1 Concept drift and label availability

Novel concept drift detection algorithms are proposed each year, and their performance assessed using different methods (see section 5). Most of these algorithms are applied to the univariate stream of correct/incorrect predictions of a learner. To achieve detections in a timely fashion, this requires that the ground-truth be available almost immediately after the prediction is made. This 'immediate' setting can be characterized by the ground-truth y^t of instance x^t being available before the next instance x^{t+1} is available (see section 3.2). Algorithms such as ADWIN [17] and EDDM [7], were tested under the aforementioned assumption. However, if the ground-truth is not immediately available, then these algorithms' ability to detect drifts might be severely decreased.

Algorithms focusing on drift detection on delayed or partially labeled streams exists. Examples include SUN [126] and the method from Klinkenberg [69] based on support vector machines. The former uses a clustering algorithm to produce 'concept clusters' at leaves of an incremental decision tree, and drifts are identified according to the deviation between history concept clusters and the current clusters.

Žliobaite [134] presents an analytical view of the conditions that must be met to allow concept drift detection in a delayed labeled setting. Three types of concept drifts are analytically studied and two of them also empirically evaluated. Unfortunately, one of the least investigated cases, when the change occurs in the input data distribution, was not empirically investigated. Therefore, the proposed methods to detect changes in the input data, such as parametric and non-parametric multivariate two-sample tests, were not discussed in-depth. We further address the problem of identifying changes in the input data distribution in section 4.2.

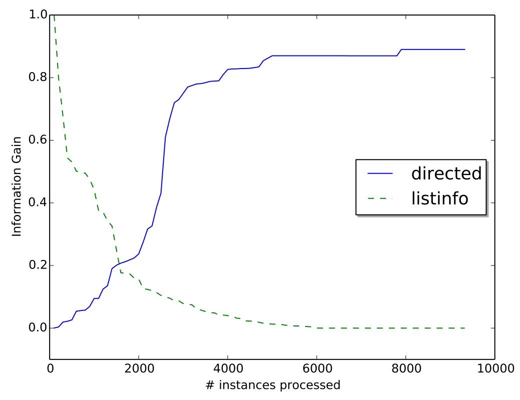

Figure 4: Two features IG variation over time for SPAM CORPUS. Adapted from [9].

4.2 Feature drift

Data streams are subject to different types of concept drifts. Examples include (i) changes in the values of a feature and their association with the class, (ii) changes in the domain of features, (iii) *changes in the subset of features that are used to label an instance*, and so on.

Despite considered in seminal works of the area [124], only recently works on the above-emphasized type of drift have gained traction. A feature drift occurs when a subset of features becomes, or ceases to be, relevant to the learning task [10]. Following the definition provided by Zhao et al. [132], a feature x_i is deemed relevant if $\exists S_i = X \setminus \{x_i\}, S'_i \subset S_i$, such that $P(Y|x_i, S'_i) > P(Y|S'_i)$ holds; and irrelevant otherwise. Given the previous definition, the removal of a relevant feature decreases the prediction power of a classifier. Also, there are two possibilities for a feature to be relevant: (i) it is strongly correlated with the class, or (ii) it forms a subset with other features, and this subset is correlated with the class [132]. An example of features importance varying over time can be visualized in Fig. 4, where the Information Gain for two features, w.r.t the target variable, is plotted over time for the SPAM CORPUS [68] dataset.

As in conventional drifts, changes in the relevant subset of features affect the class-conditional probabilities $P(Y|X)$ as the decision boundary across classes changes. Therefore, streaming algorithms should be able to detect these changes, enabling the learning algorithm to focus on the relevant features, leading to lighter-weighted and less overfit models.

To address feature drifts, few proposals have been presented in the literature. Barddal et al. [10] showed that Hoeffding Adaptive Trees [18] are the state-of-the-art learners for identifying changes in the relevance of features and adapting the model on the fly. Another important work that explicitly focuses on performing feature selection during the progress of data streams was HEFT-Stream [90], where new features are selected as new batches of arriving data become available for training.

Finally, the assessment of feature drifting scenarios should not only account for the accuracy rates of learners, but also whether the feature selection process correctly flags the changes in the relevant subset of features and if it identifies the features it should. Given that, the evaluation of feature selectors should also be dynamic, as the ground-truth subset of relevant features may drift over time.

4.3 Feature evolution

Another important trait of streaming scenarios regards the appearance and disappearance of features over time. In practice, if a new feature becomes available over time, and if it deemed relevant, one may argue that a feature drift has occurred, and then this feature could be incorporated into the learning process. Similarly, if a feature becomes unavailable, then all of its values might be treated as missing values, and then the learning model should ignore its existence. Most of the existing frameworks we will discuss in section 7, e.g., MOA [20], SAMOA [33], Scikit-multiflow [87], do not account for changes in the input vector of streaming data. Therefore, in dynamic scenarios where features may appear and disappear over time, the data stream computational representation in these frameworks will either remain static or require external updates. Developing an efficient dynamic input vector representation for streaming data is an important and difficult task. Given its relevance to some problem domains it deserves attention from machine learning framework developers.

4.4 Concept evolution

Concept evolution is a problem intrinsically related to others, such as anomaly detection for streaming data [43]. In general terms, concept evolution refers to the appearance or disappearance of class labels. This is natural in some domains, for example, the interest of users in news media change over time, with new topics appearing and older ones disappearing. Another example is Intrusion Detection Systems, where new threats appear as attackers evolve. Ideally, these threats must first be identified and then used for improving the model, however doing it automatically is a difficult task. Informally, the challenge is to discern between concept drifts, noise and the formation of a novel concept. Examples of algorithms that address concept evolution includes: ECSMiner [84], CLAM [5], and MINAS [32].

A major challenge here is the definition of evaluation setup and metrics to assess algorithms that detect concept evolution.

4.5 Hyperparameter tuning for evolving data streams

Hyperparameter tuning (or optimization) is often treated as a manual task where experienced users define a subset of hyperparameters and their corresponding range of possible values to be tested exhaustively (Grid Search), randomly (Random Search) or according to some other criteria [11]. The brute force approach of trying all possible combinations of hyperparameters and their values is time-consuming but can be efficiently executed in parallel in a batch setting. However, it can be difficult to emulate this approach in an evolving streaming scenario. A naive approach is to separate an initial set of instances from the first instances seen and perform an offline tuning of the model hyperparameters on them. Nevertheless, this makes a strong assumption that even if the concept drifts the selected hyperparameters' values will remain optimal. The challenge is to design an approach that incorporate the hyperparameter tuning as part of the continual learning process, which might involve data preprocessing, drift detection, drift recovery, and others.

Losing et al. [81] present a review and comparison of incremental learners including SVM variations, tree ensembles, instance-based models and others. Interestingly, this is one of the first works to benchmark incremental learners using a strategy to perform hyperparameter optimization. To perform the tuning a minimum of 20% (or 1,000 instances, whichever is reached first) of the training data was gathered. Assuming a stationary distribution, this approach performs reasonably well. Experiments with non-stationary streams are also briefly covered by the authors, but since the algorithms used were not designed for this setting, it was not possible to draw further conclusions about the efficiency of performing hyperparameter optimization on drifting streams.

A recent work, [120] formulates the problem of parameter tuning as an optimization problem. It uses the Nelder-Mead algorithm to exploit the space of the parameters. The NelderMead method [89] or downhill simplex method is a numerical method used to find the minimum or maximum of a function in a multidimensional space.

5. EVALUATION PROCEDURES AND DATA SOURCES

As the field evolves and practitioners, besides researchers, also start to apply the methods, it is critical to verify whether or not the currently established evaluation metrics and benchmark datasets fit the real world problems. The importance of selecting appropriate benchmark data is to avoid making assumptions about algorithms quality given empirical tests on data that might not reflect realistic scenarios.

Existing evaluation frameworks address issues such as imbalanced data, temporal dependences, cross-validation, and others [16]. For example, when the input data stream exhibit temporal dependences, a useful benchmark model is a naive *No Change* learner. This learner always predicts the next label as the previous label and, surprisingly, it may surpass advanced learning algorithms that build complex models from the input data. Žliobaite et al. [135] propose the κ-temporal statistic, which incorporates the *No Change* learner to the evaluation metric.

However, one crucial issue related to the evaluation of streaming algorithms is the lack of appropriate approaches to evaluate delayed labeled problems. As previously discussed (see section 3.2) in a delayed setting there is a non-negligible delay between the input data x and the ground-truth label y, which can vary from a few minutes/hours up to years depending on the application. A naive approach to evaluating the quality of such solutions is to record the learner prediction $\hat{y}$ when x is presented and then compare it against y once it is available. One issue with this approach is that in some applications, such as peer-to-peer lending and airlines delay prediction, the learner will be pooled several times with the same x before y is available, potentially improving its performance as time goes by as other labels are made available and used to update the model. Ideally, the learner should be capable of outputting better results since the first prediction when x is presented, however how to measure its ability to improve over time before y is made available? Despite works that address delayed labeling, evaluation frameworks have only recently been proposed [58].

5.1 Benchmark data

Data stream algorithms are usually assessed using a benchmark that is a combination of synthetic generators and real-world datasets. The synthetic data is used to allow showing how the method performs given specific problems (e.g., concept drifts, concept evolution, feature drifts, and so forth) in a controlled environment. The real world datasets are used to justify the application of the method beyond hypothetical situations; however, they are often used without guarantees that the issues addressed by the algorithm are present. For example, it is difficult to check if and when a concept drift takes place in a real dataset. The problem is that some of the synthetic streams can be considered too straightforward and perhaps outdated, e.g., STAGGER [111] and SEA [116] datasets.

When it comes to real-world data streams, some researchers use datasets that do not represent data streams or that are synthetic data masquerade as real datasets. An example that covers both concerns (not a stream and actually synthetic) is the dataset named Pokerhand[4], at some point in past it was used to assess the performance of streaming classifiers, probably because of its volume. However, it is neither "real" nor a representation of a stream of data. Until today it is still in use without any reasonable explanation. Even benchmark datasets that can be interpreted as actual data streams display some unrealistic characteristics that are often not discussed. Electricity [62] depicts the energy market from the Australian New South Wales Electricity Market, and even though the data was generated over time, often the dataset version used was preprocessed in an offline process to normalize the features, which might benefit some algorithms or at least 'leak' some future characteristics (see section 2.1.2).

Issues with evaluation frameworks are not limited to supervised learning in a streaming context. For instance, assessing concept drift detection algorithms is also subject to controversies. A standard approach to evaluate novel concept drift detection is to combine them with a classification algorithm and assess the detection capabilities of the concept drift method indirectly by observing the classification performance of the accompanying algorithm. The problem with this evaluation is that it is indirect; thus the actual characteristics of the drift detection algorithm, such as the lag between drift and detection, cannot be observed from it. This issue is detailed in a recent work [15].

Why are we not using real data streams to assess the performance of stream learning algorithms? One possible answer is the difficulty in preparing sensor data. Even though the data is abundant, it is still necessary to transform it to a suitable format, and often this means converting from a multivariate time series (see section 3.1) to a data stream. Another possible answer is that realistic data stream sources can be complicated to configure and to replicate.

5.2 Handling real streaming data

For actual implementations, an important aspect of streaming data is that the way the data is made available to the system is significant. High latency data sources will 'hold' the whole system, and there is nothing the learning algorithm can do to solve it. Different from batch learning, the

[4]`https://archive.ics.uci.edu/ml/datasets/Poker+Hand`

data source for streaming data is often harder to grasp for beginners. It is not merely a self-contained file or a well-defined database, and in fact, it has to allow the appearance of new data with low latency in a way that the learner is updated as soon as possible when new data is available. At the vanguard of stream processing there are frameworks, such as Apache Spark and Flink.

Recently, the team behind Apache Spark introduced a novel API to handle streaming data, namely the Structured Streaming API [6], which overshadows the previous Spark Streaming [130] API. Similar to its predecessor, Structured Streaming is mainly based on micro-batches, i.e., instead of immediately presenting new rows of data to the user code, the rows are combined into small logical batches. This facilitates manipulating the data as truly incremental systems can be both difficult to the user to handle and to the framework developer to come up with efficient implementations. Besides implementation details, the main difference between Spark Streaming and the new Structured Streaming API is that the latter assumes that there is a structure to the streaming data, which make it possible to manipulate data using SQL and uses the abstraction of an unbounded table.

6. STREAMING DATA IN ARTIFICIAL INTELLIGENCE

In this section, we look at stream mining in recent advanced topics in artificial intelligence, by which we mean tasks that fall outside of the traditional single-target classification or regression scenario.

6.1 Prediction of Structured Outputs

In structured output prediction, values for multiple target variables are predicted simultaneously (for each instance). A particular well-known special case is that of multi-label classification [106; 34; 131] where multiple labels are associated with each data point – a natural point of departure for many text and image labeling tasks.

Methods can be applied directly in a 'problem transformation' scenario or adapted in an 'algorithm adaptation' scheme [104], however, obtaining scalable models is inherently more challenging, since the output space is K^L for L label variables each taking K values, as opposed to K for a K-class (single-label) problem.

In other words: the output space may be of the same range of variety and dimensionality as an input space. As such we can consider the issues and techniques outlined in sections 4.2 and 4.3.

We can emphasize that in multi-output data streams there is an additional complication involving concept drift which now covers an additional dimension – models are inherently more complex and more difficult to learn and thus there is even greater motivation to adapt them as best as possible to the new concept when drift occurs, rather than resetting them. This is further encouraged under the consideration that supervised labeling is less likely to be complete under this scenario.

Structured-output learning is the case of multi-output learning where some structure is assumed in the problem domain. For example, in an image, segmentation local dependence is assumed among pixel variables, and in modeling sequences, it is often assumed temporal dependence among each of the output variables. However, essentially all multi-label and multi-output problems will have some underlying structure and thus are in fact structured-output problems in the strict sense. Indeed, many sequence prediction and time-series models can be applied practically as-is to multi-label problems and vice-versa [105]. This could include recurrent neural networks, as we review in section 6.2, or the methods mentioned already in section 3.1.

Therefore, the main challenges are dealing with the inherently more complex models and drift patterns streams dealing with structured outputs. Complex structured output prediction tasks such as captioning have yet to be approached in a data-stream context.

6.2 Recurrent Neural Networks

Many structured-ouput approaches can be approached with recurrent neural networks (RNNs). These are inherently robust and well suited to dealing with sequential data, particularly text (natural language) and signals with high temporal dependence. See, e.g., Du and Swamy [41] present a detailed overview.

RNNs are notoriously difficult to train. Obtaining good results on batch data can already require exhaustive experimentation of parameter settings, not easily affordable in the streaming context. Long Short-Term Memory neural networks (LSTMs) have gained recent popularity, but are still challenging to train on many tasks.

There are simplified RNNs which have been very effective, such as Time Delay Neural Networks, which simply include a window of input as individual instances; considered, for example, in Žliobaite et al. [121] in the context of streams.

Another useful variety of RNN more suited to data streams is the Echo State Networks (ESNs). The weights of the hidden layer of an ESN are randomly initialized and not trained. Only the output layer (usually a linear one) is trained; and stochastic gradient descent will suffice in many contexts.

ESNs are an interesting way to embed signals into vectors – making them a good starting point for converting a time series into an i.i.d. data stream which can be processed by traditional methods (see also discussion in section 3.1).

RNNs are naturally deployed in a streaming scenario for prediction, but training them under the context of concept drift has, to the best of our knowledge, not been widely approached.

Finally, we could remark that neuro-evolution is popular as a training method for RNNs in some areas, such as reinforcement learning, in particular in policy search approaches (where the policy map is represented as a neural network); see, for example, Stanley and Miikkulainen [114]. The structure of the network is evolved over time (rather than backward propagation of errors), and hence is arguably a more intuitive option in online tasks.

As training options become easier, we expect RNNs to be a more common option as a data-streams method.

6.3 Reinforcement learning

Reinforcement learning is inherently a task of learning from data streams. Observations arrive on a time-step basis, in a stream, and are typically treated either on an episode basis (here we can make an analogy with *batch-incremental* methods) or on a time-step basis (i.e., *instance-incremental* streaming). For a complete introduction to the topic, see, for example, Sutton and Barto [117].

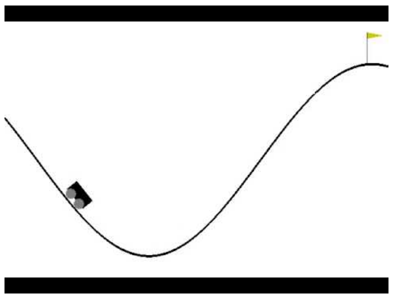

Figure 5: The Mountain Car problem is a typical benchmark in reinforcement learning. The goal is to drive the car to the top. It can be treated as a streaming problem.

The goal in reinforcement learning is to learn a policy, which is essentially a mapping from inputs (i.e., observations) to outputs (i.e., actions). This mapping is conceptually similar to that desired from a machine learning model (mapping inputs to outputs). However, the peculiarity is that ground-truth training pairs are never presented to the model, unlike in the typical supervised learning scenario. Rather a reward signal is given instead of true labels. The reward signal is often sparse across time and – the most significant challenge – is that the reward at a particular time step may correspond to an action taken many time steps ago, and it is thus difficult to break down into a time-step basis. Nevertheless, in certain environments, it is possible to consider training pairs on an episode level.

Despite the similarities with data-streams, there has been relatively little overlap in the literature. It is not difficult to conceive of scenarios where a reinforcement-learning agent needs to detect concept drift in its environment, just as any classification or regression model.

Reinforcement learning is still in its infancy relative to tasks such as supervised classification – especially in terms of industrial applications, which may explain the lack of consideration of additional complications typically considered in data-streams, as concept drift. Nevertheless, we would expect such overlap to increase as a wider variety of real-world application domains are considered.

7. SOFTWARE PACKAGES AND FRAMEWORKS

In this section, we present existing tools for applying machine learning to data streams for both research and practical applications. Initially, frameworks were designed to facilitate collaboration among research groups and allow researchers to test their ideas directly. Nowadays, tools such as the Massive Online Analysis (MOA) [20] can be adapted to deploy models in practice depending on the problem requirements.

Massive Online Analysis (MOA)[5] [20]. The MOA framework includes several algorithms for multiple tasks concerning data stream analysis. It features a larger community of researchers that continuously add new algorithms and tasks to it. The current tasks included in MOA are classification, regression, multi-label, multi-target, clustering, outlier detection, concept drift detection, active learning, and others. Besides learning algorithms, MOA also provides: data generators (e.g., AGRAWAL, Random Tree Generator, and SEA); evaluation methods (e.g., periodic holdout, test-then-train, prequential); and statistics (CPU time, RAM-hours, Kappa). MOA can be used through a GUI (Graphical User Interface) or via command line, which facilitates running batches of tests. The implementation is in Java and shares many characteristics with the WEKA framework [60], such as allowing users to extend the framework by inheriting abstract classes. Very often researchers make their source code available as an MOA extension[6].

Advanced Data mining And Machine learning System (ADAMS)[7] [107]. ADAMS is a workflow engine designed to prototype and maintain complex knowledge workflows. ADAMS is not a data stream learning tool *per se*, but it can be combined with MOA, and other frameworks like SAMOA and WEKA, to perform data stream analysis.

Scalable Advanced Massive Online Analysis (SAMOA)[8] [33]. SAMOA combines stream mining and distributed computing (i.e., MapReduce), and is described as a framework as well as a library. As a framework, SAMOA allows users to abstract the underlying stream processing execution engine and focus on the learning problem at hand. Currently, it is possible to use Storm (`http://storm.apache.org`) and Samza (`http://samza.incubator.apache.org`). SAMOA provides adapted versions of stream learners for distributed processing, including the Vertical Hoeffding Tree algorithm [72], bagging and boosting.

Vowpal Wabbit (VW)[9]. VW is an open source machine learning library with an efficient and scalable implementation that includes several learning algorithms. VW has been used to learn from a terafeature dataset using 1000 nodes in approximately an hour [2].

StreamDM[10]. StreamDM is an open source framework for big data stream mining that uses the Spark Streaming [130] extension of the core Spark API. One advantage of StreamDM in comparison to existing frameworks is that it directly benefits from the Spark Streaming API, which handles much of the complex problems of the underlying data sources, such as out of order data and recovery from failures.

Scikit-multiflow[11] [87]. Scikit-multiflow is an extension to the popular scikit-learn [98] inspired by the MOA framework. It is designed to accommodate multi-label, multi-output and single output data stream mining algorithms. One advantage of scikit-multiflow is its API similarity to scikit-learn, which is widely used worldwide.

Ray RLlib[12] [79]. RLlib is a reinforcement learning library that features reference algorithms' implementations and facilitates the creation of new algorithms through a set of scalable primitives. RLlib is part of the open source Ray project. Ray is a high-performance distributed execution framework that allows Python tasks to be distributed across larger clusters.

[5]`http://moa.cms.waikato.ac.nz`

[6]`http://moa.cms.waikato.ac.nz/moa-extensions/`

[7]`https://adams.cms.waikato.ac.nz`

[8]`http://samoa.incubator.apache.org`

[9]`https://github.com/VowpalWabbit/vowpal_wabbit`

[10]`http://huawei-noah.github.io/streamDM/`

[11]`https://github.com/scikit-multiflow/scikit-multiflow`

[12]`https://ray.readthedocs.io/en/latest/rllib.html`

8. CONCLUSIONS

We have discussed several challenges that pertain machine learning for streaming data. In some cases, these challenges have been addressed (often partially) by existing research, which we discuss and point out the shortcomings. All the topics covered in this work are important, but some have a broader impact or have been less investigated. Further developing these in the near future will help the development of the field:

- Explore the relationships between other AI developments (e.g., recurrent neural networks, reinforcement learning, etc.) and adaptive stream mining algorithms;
- Characterize and detect drifts in the absence of immediately labeled data;
- Develop adaptive learning methods that take into account verification latency;
- Incorporate pre-processing techniques that continuously transform the raw data;

It is also important to develop software that allows the application of data stream mining techniques in practice. In recent years, many frameworks were proposed, and they are constantly being updated and maintained by the community. Finally, it is unfeasible to cover all topics related to machine learning and streaming data in a single paper. Therefore, we were able to only scratch the surface for some topics that deserve further analysis in the future, such as regression; unsupervised learning; evolving graph data; image, text and other non-structured data sources; and pattern mining.

9. REFERENCES

[1] Z. S. Abdallah, M. M. Gaber, B. Srinivasan, and S. Krishnaswamy. Activity recognition with evolving data streams: A review. *ACM Computing Surveys (CSUR)*, 51(4):71, 2018.

[2] A. Agarwal, O. Chapelle, M. Dudík, and J. Langford. A reliable effective terascale linear learning system. *The Journal of Machine Learning Research*, 15(1):1111–1133, 2014.

[3] C. C. Aggarwal, J. Han, J. Wang, and P. S. Yu. A framework for clustering evolving data streams. In *International Conference on Very Large Data Bases (VLDB)*, pages 81–92, 2003.

[4] C. C. Aggarwal and P. S. Yu. On classification of high-cardinality data streams. In *SIAM International Conference on Data Mining*, pages 802–813, 2010.

[5] T. Al-Khateeb, M. M. Masud, L. Khan, C. C. Aggarwal, J. Han, and B. M. Thuraisingham. Stream classification with recurring and novel class detection using class-based ensemble. In *ICDM*, pages 31–40, 2012.

[6] M. Armbrust, T. Das, J. Torres, B. Yavuz, S. Zhu, R. Xin, A. Ghodsi, I. Stoica, and M. Zaharia. Structured streaming: A declarative api for real-time applications in apache spark. In *International Conference on Management of Data*, pages 601–613, 2018.

[7] M. Baena-García, J. del Campo-Ávila, R. Fidalgo, A. Bifet, R. Gavaldà, and R. Morales-Bueno. Early drift detection method. 2006.

[8] D. Barber. *Bayesian Reasoning and Machine Learning*. Cambridge University Press, 2012.

[9] J. P. Barddal, H. M. Gomes, and F. Enembreck. Analyzing the impact of feature drifts in streaming learning. In *International Conference on Neural Information Processing*, pages 21–28. Springer, 2015.

[10] J. P. Barddal, H. M. Gomes, F. Enembreck, and B. Pfahringer. A survey on feature drift adaptation: Definition, benchmark, challenges and future directions. *Journal of Systems and Software*, 127:278 – 294, 2017.

[11] R. Bardenet, M. Brendel, B. Kégl, and M. Sebag. Collaborative hyperparameter tuning. In *International Conference on Machine Learning*, pages 199–207, 2013.

[12] M. Barreno, B. Nelson, R. Sears, A. D. Joseph, and J. D. Tygar. Can machine learning be secure? In *ACM Symposium on Information, computer and communications security*, pages 16–25, 2006.

[13] L. E. Baum, T. Petrie, G. Soules, and N. Weiss. A maximization technique occurring in the statistical analysis of probabilistic functions of markov chains. *The annals of mathematical statistics*, 41(1):164–171, 1970.

[14] Y. Ben-Haim and E. Tom-Tov. A streaming parallel decision tree algorithm. *The Journal of Machine Learning Research*, 11:849–872, 2010.

[15] A. Bifet. Classifier concept drift detection and the illusion of progress. In *International Conference on Artificial Intelligence and Soft Computing*, pages 715–725. Springer, 2017.

[16] A. Bifet, G. de Francisci Morales, J. Read, G. Holmes, and B. Pfahringer. Efficient online evaluation of big data stream classifiers. In *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 59–68, 2015.

[17] A. Bifet and R. Gavalda. Learning from time-changing data with adaptive windowing. In *SIAM international conference on data mining*, pages 443–448, 2007.

[18] A. Bifet and R. Gavaldà. Adaptive learning from evolving data streams. In *International Symposium on Intelligent Data Analysis*, pages 249–260. Springer, 2009.

[19] A. Bifet, R. Gavalda, G. Holmes, and B. Pfahringer. *Machine Learning for Data Streams: with Practical Examples in MOA*. Adaptive Computation and Machine Learning series. MIT Press, 2018.

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

[21] A. Bifet, G. Holmes, and B. Pfahringer. Leveraging bagging for evolving data streams. In *PKDD*, pages 135–150, 2010.

[22] B. H. Bloom. Space/time trade-offs in hash coding with allowable errors. *Communications of the ACM*, 13(7):422–426, 1970.

[23] A. Blum and T. Mitchell. Combining labeled and unlabeled data with co-training. In *Conference on Computational learning theory*, pages 92–100, 1998.

[24] L. Breiman. Random forests. *Machine Learning*, 45(1):5–32, 2001.

[25] P. Carbone, A. Katsifodimos, S. Ewen, V. Markl, S. Haridi, and K. Tzoumas. Apache flink: Stream and batch processing in a single engine. *Bulletin of the IEEE Computer Society Technical Committee on Data Engineering*, 36(4), 2015.

[26] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer. Smote: synthetic minority over-sampling technique. *Journal of artificial intelligence research*, 16:321–357, 2002.

[27] S. Chen and H. He. Towards incremental learning of nonstationary imbalanced data stream: a multiple selectively recursive approach. *Evolving Systems*, 2(1):35–50, 2011.

[28] M. Chenaghlou, M. Moshtaghi, C. Leckie, and M. Salehi. Online clustering for evolving data streams with online anomaly detection. In *Pacific-Asia Conference on Knowledge Discovery and Data Mining*, pages 508–521. Springer, 2018.

[29] W. Chu, M. Zinkevich, L. Li, A. Thomas, and B. Tseng. Unbiased online active learning in data streams. In *ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 195–203, 2011.

[30] 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.

[31] M. Datar, A. Gionis, P. Indyk, and R. Motwani. Maintaining stream statistics over sliding windows. *SIAM journal on computing*, 31(6):1794–1813, 2002.

[32] E. R. de Faria, A. C. P. de Leon Ferreira de Carvalho, and J. Gama. MINAS: multiclass learning algorithm for novelty detection in data streams. *Data Mining Knowledge Discovery*, 30(3):640–680, 2016.

[33] G. De Francisci Morales and A. Bifet. Samoa: Scalable advanced massive online analysis. *Journal of Machine Learning Research*, 16:149–153, 2015.

[34] K. Dembczyński, W. Waegeman, W. Cheng, and E. Hüllermeier. On label dependence and loss minimization in multi-label classification. *Mach. Learn.*, 88(1-2):5–45, July 2012.

[35] G. Ditzler, M. D. Muhlbaier, and R. Polikar. Incremental learning of new classes in unbalanced datasets: Learn++.udnc. In *International Workshop on Multiple Classifier Systems*, pages 33–42, 2010.

[36] G. Ditzler, R. Polikar, and N. Chawla. An incremental learning algorithm for non-stationary environments and class imbalance. In *International Conference on Pattern Recognition*, pages 2997–3000, 2010.

[37] P. Domingos and G. Hulten. Mining high-speed data streams. In *ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 71–80, 2000.

[38] A. R. T. Donders, G. J. Van Der Heijden, T. Stijnen, and K. G. Moons. A gentle introduction to imputation of missing values. *Journal of clinical epidemiology*, 59(10):1087–1091, 2006.

[39] Y. Dong and N. Japkowicz. Threaded ensembles of autoencoders for stream learning. *Computational Intelligence*, 34(1):261–281, 2018.

[40] D. M. dos Reis, P. Flach, S. Matwin, and G. Batista. Fast unsupervised online drift detection using incremental kolmogorov-smirnov test. In *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1545–1554, 2016.

[41] K.-L. Du and M. N. Swamy. *Neural Networks and Statistical Learning*. Springer Publishing Company, Incorporated, 2013.

[42] R. Elwell and R. Polikar. Incremental learning of concept drift in nonstationary environments. *IEEE Transactions on Neural Networks*, 22(10):1517–1531, 2011.

[43] M. A. Faisal, Z. Aung, J. R. Williams, and A. Sanchez. Data-stream-based intrusion detection system for advanced metering infrastructure in smart grid: A feasibility study. *IEEE Systems journal*, 9(1):31–44, 2015.

[44] W. Fan and A. Bifet. Mining big data: current status, and forecast to the future. *ACM SIGKDD Explorations Newsletter*, 14(2):1–5, 2013.

[45] M. M. Gaber, A. Zaslavsky, and S. Krishnaswamy. Mining data streams: a review. *ACM Sigmod Record*, 34(2):18–26, 2005.

[46] M. Galar, A. Fernandez, E. Barrenechea, H. Bustince, and F. Herrera. A review on ensembles for the class imbalance problem: bagging-, boosting-, and hybrid-based approaches. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 42(4):463–484, 2012.

[47] S. Galelli, G. B. Humphrey, H. R. Maier, A. Castelletti, G. C. Dandy, and M. S. Gibbs. An evaluation framework for input variable selection algorithms for environmental data-driven models. *Environmental Modelling & Software*, 62:33 – 51, 2014.

[48] J. Gama and P. Kosina. Learning about the learning process. In *International Symposium on Intelligent Data Analysis*, pages 162–172, 2011.

[49] J. Gama and P. Kosina. Recurrent concepts in data streams classification. *Knowledge and Information Systems*, 40(3):489–507, 2014.

[50] J. Gama, I. Žliobaitė, A. Bifet, M. Pechenizkiy, and A. Bouchachia. A survey on concept drift adaptation. *ACM computing surveys (CSUR)*, 46(4):44, 2014.

[51] S. García, S. Ramírez-Gallego, J. Luengo, J. M. Benítez, and F. Herrera. Big data preprocessing: methods and prospects. *Big Data Analytics*, 1(1):9, 2016.

[52] A. Ghazikhani, R. Monsefi, and H. S. Yazdi. Ensemble of online neural networks for non-stationary and imbalanced data streams. *Neurocomputing*, 122:535–544, 2013.

[53] H. M. Gomes, J. P. Barddal, F. Enembreck, and A. Bifet. A survey on ensemble learning for data stream classification. *ACM Computing Surveys*, 50(2):23:1–23:36, 2017.

[54] H. M. Gomes, A. Bifet, J. Read, J. P. Barddal, F. Enembreck, B. Pfharinger, G. Holmes, and T. Abdessalem. Adaptive random forests for evolving data stream classification. *Machine Learning*, 106(9-10):1469–1495, 2017.

[55] H. M. Gomes and F. Enembreck. Sae: Social adaptive ensemble classifier for data streams. In *IEEE Symposium on Computational Intelligence and Data Mining*, pages 199–206, April 2013.

[56] H. M. Gomes and F. Enembreck. Sae2: Advances on the social adaptive ensemble classifier for data streams. In *Proceedings of the 29th Annual ACM Symposium on Applied Computing (SAC)*, SAC 2014, pages 199–206, March 2014.

[57] H. M. Gomes, J. Read, and A. Bifet. Streaming random patches for evolving data stream classification. In *IEEE International Conference on Data Mining*. IEEE, 2019.

[58] M. Grzenda, H. M. Gomes, and A. Bifet. Delayed labelling evaluation for data streams. *Data Mining and Knowledge Discovery*, to appear.

[59] S. Guha, N. Mishra, G. Roy, and O. Schrijvers. Robust random cut forest based anomaly detection on streams. In *International Conference on Machine Learning*, pages 2712–2721, 2016.

[60] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten. The weka data mining software: an update. *ACM SIGKDD Explorations newsletter*, 11(1):10–18, 2009.

[61] A. Haque, B. Parker, L. Khan, and B. Thuraisingham. Evolving big data stream classification with mapreduce. In *International Conference on Cloud Computing (CLOUD)*, pages 570–577, 2014.

[62] M. Harries and N. S. Wales. Splice-2 comparative evaluation: Electricity pricing. 1999.

[63] S. Hashemi, Y. Yang, Z. Mirzamomen, and M. Kangavari. Adapted one-versus-all decision trees for data stream classification. *IEEE Transactions on Knowledge and Data Engineering*, 21(5):624–637, 2009.

[64] M. J. Hosseini, A. Gholipour, and H. Beigy. An ensemble of cluster-based classifiers for semi-supervised classification of non-stationary data streams. *Knowledge and Information Systems*, 46(3):567–597, 2016.

[65] L. Huang, A. D. Joseph, B. Nelson, B. I. Rubinstein, and J. Tygar. Adversarial machine learning. In *ACM workshop on Security and artificial intelligence*, pages 43–58, 2011.

[66] N. Jiang and L. Gruenwald. Research issues in data stream association rule mining. *ACM Sigmod Record*, 35(1):14–19, 2006.

[67] T. Joachims. Transductive inference for text classification using support vector machines. In *ICML*, volume 99, pages 200–209, 1999.

[68] I. Katakis, G. Tsoumakas, E. Banos, N. Bassiliades, and I. Vlahavas. An adaptive personalized news dissemination system. *Journal of Intelligent Information Systems*, 32(2):191–212, 2009.

[69] R. Klinkenberg. Using labeled and unlabeled data to learn drifting concepts. In *IJCAI Workshop on Learning from Temporal and Spatial Data*, pages 16–24, 2001.

[70] J. Z. Kolter, M. Maloof, et al. Dynamic weighted majority: A new ensemble method for tracking concept drift. In *ICDM*, pages 123–130, 2003.

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

[72] N. Kourtellis, G. D. F. Morales, A. Bifet, and A. Murdopo. Vht: Vertical hoeffding tree. In *IEEE International Conference on Big Data*, pages 915–922, 2016.

[73] G. Krempl, I. Žliobaite, D. Brzeziński, E. Hüllermeier, M. Last, V. Lemaire, T. Noack, A. Shaker, S. Sievi, M. Spiliopoulou, et al. Open challenges for data stream mining research. *ACM SIGKDD Explorations newsletter*, 16(1):1–10, 2014.

[74] L. I. Kuncheva. A stability index for feature selection. In *International Multi-Conference: Artificial Intelligence and Applications*, AIAP'07, pages 390–395, 2007.

[75] B. Kveton, H. H. Bui, M. Ghavamzadeh, G. Theocharous, S. Muthukrishnan, and S. Sun. Graphical model sketch. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 81–97, 2016.

[76] J. Langford, L. Li, and A. Strehl. Vowpal Wabbit, 2007.

[77] P. Lehtinen, M. Saarela, and T. Elomaa. Online chimerge algorithm. In *Data Mining: Foundations and Intelligent Paradigms*, pages 199–216. 2012.

[78] M. Li, M. Liu, L. Ding, E. A. Rundensteiner, and M. Mani. Event stream processing with out-of-order data arrival. In *International Conference on Distributed Computing Systems Workshops*, pages 67–67, 2007.

[79] E. Liang, R. Liaw, R. Nishihara, P. Moritz, R. Fox, J. Gonzalez, K. Goldberg, and I. Stoica. Ray rllib: A composable and scalable reinforcement learning library. *arXiv preprint arXiv:1712.09381*, 2017.

[80] V. López, A. Fernández, S. García, V. Palade, and F. Herrera. An insight into classification with imbalanced data: Empirical results and current trends on using data intrinsic characteristics. *Information Sciences*, 250:113–141, 2013.

[81] V. Losing, B. Hammer, and H. Wersing. Incremental on-line learning: A review and comparison of state of the art algorithms. *Neurocomputing*, 275:1261–1274, 2018.

[82] G. Louppe and P. Geurts. Ensembles on random patches. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 346–361. Springer, 2012.

[83] D. Marron, J. Read, A. Bifet, T. Abdessalem, E. Ayguade, and J. Herrero. Echo state hoeffding tree learning. In R. J. Durrant and K.-E. Kim, editors, *Asian Conference on Machine Learning*, volume 63, pages 382–397, 2016.

[84] M. Masud, J. Gao, L. Khan, J. Han, and B. M. Thuraisingham. Classification and novel class detection in concept-drifting data streams under time constraints. *IEEE Transactions on Knowledge and Data Engineering*, 23(6):859–874, 2011.

[85] M. 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. In *ICDM*, pages 929–934. IEEE, 2008.

[86] I. Mitliagkas, C. Caramanis, and P. Jain. Memory limited, streaming pca. In *Advances in Neural Information Processing Systems*, pages 2886–2894, 2013.

[87] J. Montiel, J. Read, A. Bifet, and T. Abdessalem. Scikit-multiflow: A multi-output streaming framework. *Journal of Machine Learning Research*, 19(72), 2018.

[88] M. D. Muhlbaier, A. Topalis, and R. Polikar. Learn++.nc: Combining ensemble of classifiers with dynamically weighted consult-and-vote for efficient incremental learning of new classes. *IEEE transactions on neural networks*, 20(1):152–168, 2009.

[89] J. Nelder and R. Mead. A simplex method for function minimization. *The Computer Journal*, 7:308–313, 1965.

[90] H.-L. Nguyen, Y.-K. Woon, W.-K. Ng, and L. Wan. Heterogeneous ensemble for feature drifts in data streams. In P.-N. Tan, S. Chawla, C. K. Ho, and J. Bailey, editors, *Advances in Knowledge Discovery and Data Mining*, pages 1–12, 2012.

[91] S. Nogueira and G. Brown. Measuring the stability of feature selection. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 442–457. Springer, 2016.

[92] A. Oliver, A. Odena, C. A. Raffel, E. D. Cubuk, and I. Goodfellow. Realistic evaluation of deep semi-supervised learning algorithms. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, pages 3238–3249. 2018.

[93] N. Oza. Online bagging and boosting. In *IEEE International Conference on Systems, Man and Cybernetics*, volume 3, pages 2340–2345 Vol. 3, Oct 2005.

[94] S. J. Pan, Q. Yang, et al. A survey on transfer learning. *IEEE Transactions on knowledge and data engineering*, 22(10):1345–1359, 2010.

[95] B. Parker and L. Khan. Rapidly labeling and tracking dynamically evolving concepts in data streams. *IEEE International Conference on Data Mining Workshops*, 0:1161–1164, 2013.

[96] B. Parker, A. M. Mustafa, and L. Khan. Novel class detection and feature via a tiered ensemble approach for stream mining. In *IEEE International Conference on Tools with Artificial Intelligence*, volume 1, pages 1171–1178, 2012.

[97] B. S. Parker and L. Khan. Detecting and tracking concept class drift and emergence in non-stationary fast data streams. In *AAAI Conference on Artificial Intelligence*, 2015.

[98] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, et al. Scikit-learn: Machine learning in python. *Journal of machine learning research*, 12(Oct):2825–2830, 2011.

[99] B. Pfahringer, G. Holmes, and R. Kirkby. Handling numeric attributes in hoeffding trees. In *Pacific-Asia Conference on Advances in Knowledge Discovery and Data Mining*, pages 296–307, 2008.

[100] X. C. Pham, M. T. Dang, S. V. Dinh, S. Hoang, T. T. Nguyen, and A. W. C. Liew. Learning from data stream based on random projection and hoeffding tree classifier. In *International Conference on Digital Image Computing: Techniques and Applications*, pages 1–8, 2017.

[101] C. Pinto and J. Gama. Partition incremental discretization. In *Portuguese conference on artificial intelligence*, pages 168–174, 2005.

[102] J. Plasse and N. Adams. Handling delayed labels in temporally evolving data streams. In *IEEE ICBD*, pages 2416–2424, 2016.

[103] S. Ramírez-Gallego, B. Krawczyk, S. García, M. Woźniak, and F. Herrera. A survey on data preprocessing for data stream mining: current status and future directions. *Neurocomputing*, 239:39–57, 2017.

[104] J. Read, A. Bifet, G. Holmes, and B. Pfahringer. Scalable and efficient multi-label classification for evolving data streams. *Machine Learning*, 88(1-2):243–272, 2012.

[105] J. Read, L. Martino, and J. Hollmén. Multi-label methods for prediction with sequential data. *Pattern Recognition*, 63(March):45–55, 2017.

[106] J. Read, B. Pfahringer, G. Holmes, and E. Frank. Classifier chains for multi-label classification. *Machine Learning*, 85(3):333–359, 2011.

[107] P. Reutemann and J. Vanschoren. Scientific workflow management with adams. In *Machine Learning and Knowledge Discovery in Databases*, pages 833–837. Springer, 2012.

[108] P. Roy, A. Khan, and G. Alonso. Augmented sketch: Faster and more accurate stream processing. In *International Conference on Management of Data*, pages 1449–1463, 2016.

[109] J. Rushing, S. Graves, E. Criswell, and A. Lin. A coverage based ensemble algorithm (cbea) for streaming data. In *IEEE International Conference on Tools with Artificial Intelligence*, pages 106–112, 2004.

[110] M. Salehi and L. Rashidi. A survey on anomaly detection in evolving data:[with application to forest fire risk prediction]. *ACM SIGKDD Explorations Newsletter*, 20(1):13–23, 2018.

[111] J. C. Schlimmer and R. H. Granger. Incremental learning from noisy data. *Machine learning*, 1(3):317–354, 1986.

[112] A. Shrivastava, A. C. Konig, and M. Bilenko. Time adaptive sketches (ada-sketches) for summarizing data streams. In *International Conference on Management of Data*, pages 1417–1432, 2016.

[113] V. Sindhwani, P. Niyogi, and M. Belkin. Beyond the point cloud: from transductive to semi-supervised learning. In *ICML*, pages 824–831, 2005.

[114] K. O. Stanley and R. Miikkulainen. Efficient reinforcement learning through evolving neural network topologies. In *Genetic and Evolutionary Computation Conference*, page 9, San Francisco, 2002.

[115] I. Stoica, D. Song, R. A. Popa, D. Patterson, M. W. Mahoney, R. Katz, A. D. Joseph, M. Jordan, J. M. Hellerstein, J. E. Gonzalez, et al. A berkeley view of systems challenges for ai. *arXiv preprint arXiv:1712.05855*, 2017.

[116] W. N. Street and Y. Kim. A streaming ensemble algorithm (sea) for large-scale classification. In *ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 377–382, 2001.

[117] R. S. Sutton and A. G. Barto. *Introduction to Reinforcement Learning*. MIT Press, 1st edition, 1998.

[118] L. Torgo, R. P. Ribeiro, B. Pfahringer, and P. Branco. Smote for regression. In *Portuguese conference on artificial intelligence*, pages 378–389. Springer, 2013.

[119] A. Tsymbal. The problem of concept drift: definitions and related work. Technical report, 2004.

[120] B. Veloso, J. Gama, and B. Malheiro. Self hyper-parameter tuning for data streams. In *International Conference on Discovery Science*, page to appear, 2018.

[121] I. Žliobaitė, A. Bifet, J. Read, B. Pfahringer, and G. Holmes. Evaluation methods and decision theory for classification of streaming data with temporal dependence. *Machine Learning*, 98(3):455–482, 2014.

[122] G. I. Webb. Contrary to popular belief incremental discretization can be sound, computationally efficient and extremely useful for streaming data. In *ICDM*, pages 1031–1036. IEEE, 2014.

[123] G. I. Webb, R. Hyde, H. Cao, H. L. Nguyen, and F. Petitjean. Characterizing concept drift. *Data Mining and Knowledge Discovery*, 30(4):964–994, 2016.

[124] G. Widmer and M. Kubat. Learning in the presence of concept drift and hidden contexts. *Machine Learning*, 23(1):69–101, Apr. 1996.

[125] K. Wu, K. Zhang, W. Fan, A. Edwards, and S. Y. Philip. Rs-forest: A rapid density estimator for streaming anomaly detection. In *ICDM*, pages 600–609. IEEE, 2014.

[126] X. Wu, P. Li, and X. Hu. Learning from concept drifting data streams with unlabeled data. *Neurocomputing*, 92:145–155, 2012.

[127] X. Wu, X. Zhu, G.-Q. Wu, and W. Ding. Data mining with big data. *IEEE transactions on knowledge and data engineering*, 26(1):97–107, 2014.

[128] T. Yang, L. Liu, Y. Yan, M. Shahzad, Y. Shen, X. Li, B. Cui, and G. Xie. Sf-sketch: A fast, accurate, and memory efficient data structure to store frequencies of data items. In *ICDE*, pages 103–106, 2017.

[129] W. Yu, Y. Gu, and J. Li. Single-pass pca of large high-dimensional data. In *International Joint Conference on Artificial Intelligence*, pages 3350–3356, 2017.

[130] M. Zaharia, T. Das, H. Li, T. Hunter, S. Shenker, and I. Stoica. Discretized streams: Fault-tolerant streaming computation at scale. In *ACM Symposium on Operating Systems Principles*, pages 423–438, 2013.

[131] M.-L. Zhang and Z.-H. Zhou. A review on multi-label learning algorithms. *IEEE Transactions on Knowledge and Data Engineering*, 26(8):1819–1837, 2014.

[132] Z. Zhao, F. Morstatter, S. Sharma, S. Alelyani, A. Anand, and H. Liu. Advancing feature selection research. *ASU feature selection repository*, pages 1–28, 2010.

[133] G. Zhou, K. Sohn, and H. Lee. Online incremental feature learning with denoising autoencoders. In *Artificial intelligence and statistics*, pages 1453–1461, 2012.

[134] I. Žliobaite. Change with delayed labeling: When is it detectable? In *IEEE International Conference on Data Mining Workshops*, pages 843–850, 2010.

[135] I. Žliobaitė, A. Bifet, J. Read, B. Pfahringer, and G. Holmes. Evaluation methods and decision theory for classification of streaming data with temporal dependence. *Machine Learning*, 98(3):455–482, 2015.