11

Algorithm Recommendation for Data Streams

Summary. This chapter focuses on metalearning approaches that have been applied to data streams. This is an important area, as many real-world data arrive in the form of a stream of observations. We first review some important aspects of the data stream setting, which may involve online learning, non-stationarity, and concept drift. Then we focus on three types of approaches to algorithm recommendation that exploit metalearning. The first group splits the data stream into different parts and extracts metafeatures for each part. This information is used to decide which machine learning method (e.g., a classifier) should be applied for the next part. The approaches in the second group build an ensemble in an online manner. Their performance is monitored and the ones that have good performance on the recent part of the data are selected for the next part of the data stream. The third group of approaches aims to exploit recurring concepts in the data. Indeed, many data have a seasonality effect (e.g., data measured on weekdays vs data measured during weekends), and by applying metalearning we can reuse older models whenever appropriate. Finally, this chapter closes with some open research questions and directions for future work.

11.1 Introduction

Real-time analysis of data streams is a key area of data mining research. Many real-world collected data is in fact a stream, where observations come in one by one, and algorithms processing these observations are often subject to time and memory constraints. Some examples of data streams are:

- Stock prices and trade markets. The price of a stock constantly fluctuates, and is influenced by factors in the recent past. A commonly used dataset that embodies this concept is the electricity dataset, in which the goal is to predict whether the price will rise or fall. This dataset is available on OpenML[1] and UCI.
- Chemical compounds. The composition of chemical compounds is determined with the help of various sensors under strictly controlled conditions. Over time, the sensors can deteriorate or even fail, but the model still needs to determine the composition correctly. This dataset is available on OpenML[2] and UCI.

[1] `https://www.openml.org/d/151`
[2] `https://www.openml.org/d/1476`

P. Brazdil et al., *Metalearning*, Cognitive Technologies,
https://doi.org/10.1007/978-3-030-67024-5_11

- Human body measurements, such as electroencephalography (EEG) data. EEG is a monitoring method to record the electrical activity of the brain by attaching several sensors onto the head. The assumption with this kind of data is that, based on recent brain waves, it can be used to predict several events in the body. A common example of EEG data is the *eye-state database*, where the goal is to predict whether the human subject will have his eyes open or closed, based on the EEG data. This dataset is available on OpenML[3] and UCI.

In all these examples above, the goal is to predict something in the future (electricity price, chemical compound, and the action of blinking or not). A classifier can be trained to make these predictions, but later on, as the correct value is observed, the model can be retrained or suitably adapted.

There are several key differences between data streams and conventional types of data that have been explored in various other chapters of this book. The most important differences, as highlighted by various authors (see, e.g., Domingos and Hulten (2003); Gama et al. (2009); Bifet et al. (2010); Read et al. (2012)), are:

Non-stationarity The data is of non-stationary nature, that is, the order of the examples matters. This, for example, prohibits the use of cross-validation in evaluation, as some of the basic assumptions are violated.

Online learning Observations come at different points in time. This means that the algorithm has to handle the observations on a one-by-one basis, and hence should be updatable.

Infinity An algorithm should expect an infinite stream of observations, limiting the options in terms of computational complexity.

Concept drift A learned concept can change over time. When analyzing financial data, after a disrupting event occurs on the market, the existing models can become obsolete. This phenomenon is called *concept drift*. Data stream classifiers should be able to detect these events and act accordingly (e.g., by updating or replacing the model).

These aforementioned properties impose some requirements on algorithms that model data streams, as pointed out by, e.g., Bifet et al. (2010) and Read et al. (2012):

Data processing Process one observation at a time, and inspect it at most once. As data streams generally consist of large amounts of data, it is not feasible to keep all data in memory. Of course, in many applications a pre-defined number of observations can be stored for later inspection (e.g., k-NN), but the decision of whether to store or disregard an observation has to be made on the fly.

Resources Expect an infinite stream, but process it under finite resources. Ideally, the processing of a given observation (either for training or prediction) should require a constant amount of time. For large data streams, the memory requirements might also become an issue, so proper memory management needs to be applied.

Prediction Be ready to predict at any time. As observations come in one by one, the model might require a sufficient amount of data before it is capable of providing accurate predictions. This can occur especially at the beginning of a stream.

Formalization

Formally, a data stream is an ordered collection of n base-level observations, $\mathcal{D}^{base} = ((\mathbf{x}_i^{base}, y_i^{base}) \mid i = 1, \ldots, n)$, to map an input $\mathbf{x}_i^{base}$ to an output $f(\mathbf{x}_i^{base})$, which closely

[3] `https://www.openml.org/d/1471`

represents y_i^{base}. Models $f(\mathbf{x}_i^{base})$ that work well on some parts of the stream might become obsolete and outdated due to the aforementioned concept drift. The research community has developed a large number of machine learning algorithms capable of online modeling of general trends in stream data and providing accurate predictions for future observations.

11.1.1 Adapting batch classifiers to the data stream setting

Some batch classifiers can be adapted to a data stream setting. Examples are k nearest neighbor (Beringer and Hüllermeier, 2007; Zhang et al., 2011), stochastic gradient descent (SGD) (Bottou, 2004), and SPegasos (Stochastic Primal Estimated sub-GrAdient SOlver for SVMs) (Shalev-Shwartz et al., 2011). Both stochastic gradient descent and SPegasos are gradient descent methods capable of learning a variety of linear models, such as support vector machines and logistic regression, depending on the chosen loss function.

Other classifiers have been created specifically to operate on data streams. Most notably, Domingos and Hulten (2000) introduced the *Hoeffding tree* induction algorithm, which inspects every example only once, and stores per-leaf statistics to calculate the *information gain* on which the split criterion is determined. The *Hoeffding bound* states that the true mean of a random variable of a given range will not differ from the estimated mean by more than a certain value. This provides statistical evidence that a certain split is superior to others. As Hoeffding trees seem to work very well in practice, many variants have been proposed, such as Hoeffding option trees (Pfahringer et al., 2007), adaptive Hoeffding trees (Bifet and Gavaldà, 2009) and random Hoeffding trees (Bifet et al., 2012).

Moreover, a commonly used technique to adapt traditional batch classifiers to the data stream setting is training them on a window of w recent examples: after w new examples have been observed, a new model is built. This approach has the advantage that old examples are ignored, providing natural protection against concept drift. A disadvantage is that it does not operate directly on the most recently observed data, until w new observations are made and the model is retrained. Read et al. (2012) compare the performance of these *batch-incremental* classifiers with common data stream classifiers, and conclude that the overall performance is equivalent, although batch-incremental classifiers generally use more resources.

Finally, Finn et al. (2019) propose an online version of MAML. MAML is a meta-learning approach to optimizing the initial parameters (rather than hyperparameters) of gradient-based models and will be explained in detail in Chapter 13. The basic idea is to set the parameters for the next time step equal to the best parameters in hindsight.

11.1.2 Adapting ensembles to the data stream setting

As shown in Chapter 9, ensemble techniques train multiple classifiers, which are then used to produce a prediction. Various schemes exist regarding how this is done. More details can be found in Chapter 9. In this section we review how some techniques, namely bagging and boosting, have been extended to the data stream setting.

Bagging (Breiman, 1996) exploits the instability of classifiers by training them on different *bootstrap replicates*, which are resampling samples (with replacement) of the training set. Online bagging (Oza, 2005) operates on data streams by drawing the weight of each example from a $Poisson(1)$ distribution, which converges to the behavior of the

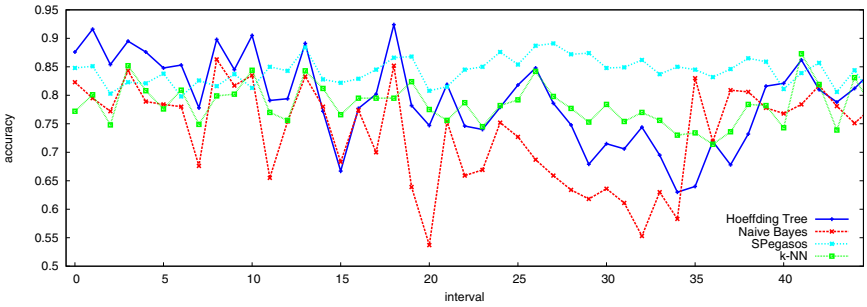

Fig. 11.1: Performance of four classifiers on intervals (size $1{,}000$) of the electricity dataset. Each data point represents the accuracy of a classifier on the most recent interval. Figure adapted from van Rijn (2016)

classical bagging algorithm if the number of examples is large. As the Hoeffding bound gives statistical evidence that a certain split criteria is optimal, this makes them more stable and hence less suitable to use in a bagging scheme. However, in practice this yields good results. Boosting (Schapire, 1990) is a technique that sequentially trains multiple classifiers, in which more weight is given to examples that were misclassified by earlier classifiers. Online boosting (Oza, 2005) applies this technique on data streams by assigning more weight to training examples that were misclassified by previously trained classifiers in the ensemble.

Chapter 10 gives more details on how metalearning and AutoML techniques can be used to design good ensembles for the the current dataset.

11.1.3 Motivation

As data streams are constantly subject to change over time, the most accurate classifier for a given interval of observations also changes frequently, as illustrated by Figure 11.1. The horizontal axis represents a certain chronological point in the stream, while the vertical axis represents the performance of various online classifiers. Due to the changing behaviour of the underlying data stream, at various points in the stream, different classifiers perform best. For example, in the beginning the Hoeffding tree is superior, while later SPegasos performs best. It would be logical to dynamically adapt which learner or portfolio of learners should be applied to a given part of the stream. Hence, we are dealing with a repeating algorithm selection problem. As such, we want to minimize the overall loss expressed by $\arg\min_{f_{meta}} \sum_i^n \mathcal{L}(f_{f_{meta}}(\mathbf{x}_i^{base}), y_i^{base})$. Here, f_{meta} represents some dynamic procedure that determines, for each observation, which models to use for prediction. Metalearning has been successfully used to achieve this, as described further on in this chapter.

The remainder of this chapter is organized as follows: Section 11.2 demonstrates how traditional metafeatures can be used to solve this problem. Section 11.3 shows several techniques that have been developed to build ensembles that adapt to the current state of the stream. As data streams are often subject to seasonality, it is likely that some

concepts might re-occur. Section 11.4 describes several works that leverage this property. Finally, in Section 11.5, we discuss general trends and challenges for future research.

11.2 Metafeature-Based Approaches

As shown in Figure 11.1, not only the performance of classifiers changes over the data stream, but also the relative ranking of the classifiers. Note that this figure shows how accurate the classifiers are in a window of $1{,}000$ independent observations. In this particular example, the Hoeffding tree performs best at the beginning of the stream, although it has several dips in performance (e.g., from interval 14 to 18 and from 46 to 41). After approximately interval 19 onwards, SPegasos is the best classifier until the end of the stream, where the Hoeffding tree, k-NN, and again SPegasos would lead to more accurate predictions.

One challenge for metalearning and algorithm selection in particular would be to dynamically predict which classifier would perform best for the next window of observations. In an ideal case, the algorithm selection system would always correctly identify which classifier to choose, leading to an accuracy that corresponds with the top boundary of the classifiers in Figure 11.1. This would yield performance gains in particular for data streams that are volatile, and for which at different intervals of observations different classifiers perform well.

Although it would be an interesting task to predict beforehand which classifier performs best on the metafeatures inferred over the complete data stream, this is not realistic; as was pointed out in Section 11.1, there is a requirement not to pass multiple times over the observations. One possibility is to take a small sample from the (beginning of the) data stream, calculate metafeatures based on this sample, and make a prediction on the basis of this (van Rijn et al., 2014). However, this is also not really practical. Due to possible concept drift, any conclusion based on the beginning of the stream might be outdated as time passes.

In order to dynamically select which classifier to use at any given point in time, we need the same components as in many other algorithm selection frameworks, that is: (i) a set of earlier observed data streams, (ii) performance values of the various classifiers (on intervals (portions) of) the data streams, and (iii) metafeatures that characterize (the intervals of) the data streams. There have been several works that attempt to do this. In this section we review some of these approaches.

11.2.1 Methods

Figure 11.2 shows the general framework for dynamic algorithm selection for data streams. The top of the image displays the current data stream. Each block here represents an observation in this data stream. The index of the current observation, for which a prediction is required, is indicated by c. We have seen all observations before index c and have not seen any of the observations from index c on. We are interested in selecting a classifier for the observations that are indicated by α, starting with the current observation c. This can be done by extracting metafeatures based on the most recent part of the stream, indicated by the window w.

After using the selected classifier for making predictions for all the observations in interval α, both the window w and interval α shift by $|\alpha|$ observations, and the algorithm selection system repeats this procedure. If the meta-model allows for incremental updates, it can be trained on the recently generated meta-observation.

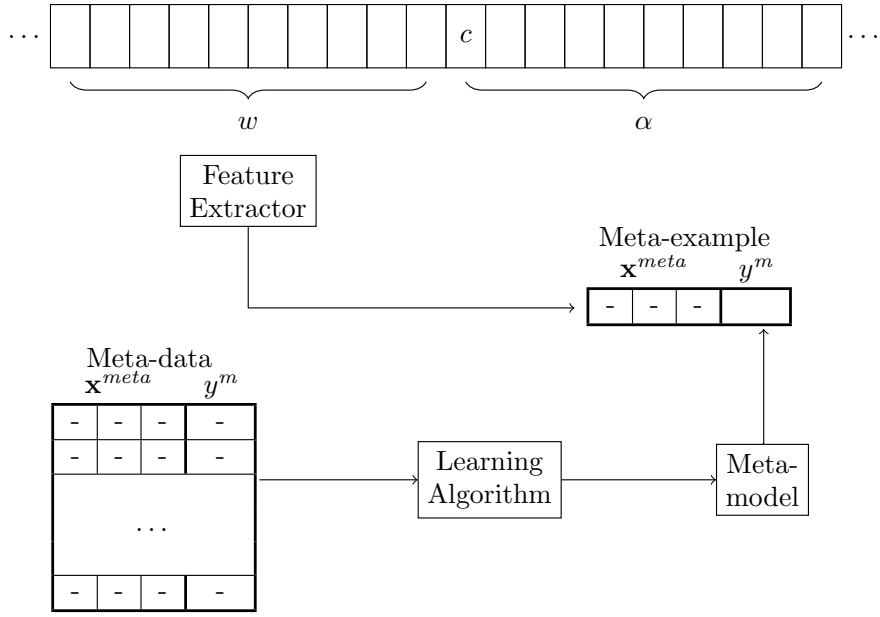

Fig. 11.2: Architecture of an algorithm selection system that dynamically selects a classifier for an interval α of observations. y^{meta} is abbreviated as y^m. Figure adapted from Rossi et al. (2014)

11.2.2 Training the meta-model

The meta-model can be trained on metadata from the current stream (Rossi et al., 2014), metadata from previously seen streams (van Rijn et al., 2014), or metadata from both. In either case, the meta-model is trained on the same form of metadata. Generally, the meta-models are induced based on a dataset $\mathcal{D}^{meta} = \{(\mathbf{x}_i^{meta}, y_i^{meta}) \mid i = 1, \ldots, m\}$ (with $|\mathcal{D}^{meta}| = m$) to map an input $\mathbf{x}^{meta}$ to an output $f(\mathbf{x}^{meta})$, which closely represents y^{meta}. Here, $\mathbf{x}^{meta}$ comprises the metafeatures calculated on some interval (section) of observations from the base data streams, and y^{meta} represents the classifier that performed best on that given interval.

The question regarding how to define the intervals (start position and length) needs to be considered. In some approaches, the size of the interval is fixed to w and the start of intervals is defined by either using a *sliding window* or *shifting window* scheme.

If the sliding window scheme is used, an interval can start at any given observation, splitting a data stream into $n - w$ equal intervals (i.e., the intervals starting with observations $\{1, 2, \ldots, n-w\}$). Note that the data stream can be potentially of infinite length. As such, we consider n as the number of observations in the part of the data that we analyze. Each observation in the stream is involved in several intervals, yielding a large amount of metadata.

Alternatively, when the shifting window is used, an interval can only start any observation that is indexed by a divisor of w (i.e., the intervals starting with observations $\{1, w+1, 2 \times w+1, \ldots, \lceil \frac{n}{w} \rceil \times w + 1\}$). This way, each observation is used exactly in

one interval. This yields less training instances for the meta-model, and presumably decreases the training time and the amount of redundant information. Both Rossi et al. (2014) and van Rijn et al. (2014) opted for this setup.

11.2.3 Metafeatures

We can use most of the metafeatures defined in Chapter 4 on the intervals of the data streams. In the data stream literature, the following subset of metafeatures has been used: *simple* (number of examples, number of attributes), *statistical* (mean standard deviation of attributes, mean skewness of attributes), *information theoretic* (class entropy, mean mutual information), and *landmarkers* (Pfahringer et al., 2000), representing performance evaluations of simple classifiers on the data stream. These are more expensive to measure but often yield good results.

Additionally, van Rijn et al. (2014) introduced the concept of *drift detection metafeatures*. These are produced by running a drift detector (in particular DDM (Gama et al., 2004) and ADWIN (Bifet and Gavaldà, 2007)) on each interval and recording the number of warnings and alarms raised. This informs the meta-algorithm that there was a change of concept and that some corrective action should be taken, such as retraining some classifier or substituting it by another one.

Finally, van Rijn et al. (2015) introduced the concept of *stream landmarkers*. For each interval, the performance of all classifiers is measured and the best classifier identified. So the classifier that performed best on the previous interval can be used as an option to fall back on in the current interval.

11.2.4 Considerations regarding hyperparameters

The algorithm selection systems described in this chapter contain several considerations and hyperparameters that influence performance. We describe the most important ones.

Set of base-classifiers: The performance of any algorithm selection system is highly dependent on the algorithms that it can choose from. In general, it is good to have a diverse set of good-performing classifiers. Having more classifiers potentially makes the learning task harder, as the base classifiers might be less represented in the meta-dataset. Chapter 8 elaborates on a suitable choice of base-classifiers; further in this chapter we will also address this issue specific to data stream classifiers.

Set of metafeatures: Like most metalearning systems, the performance heavily relies on metafeatures. Many metafeatures that have been discussed in Chapter 4 can be used on the interval of the data streams. However, metafeatures such as *number of features* and *number of observations* in the interval will obviously be constant throughout the stream and will therefore not be able to trigger a dynamic switch of classifiers.

Meta-model: The meta-model that is trained on the meta-dataset that includes metafeatures as attributes. The quality of the model will depend on the setup, and hence also metafeatures. An accurate model will make the right choices of base-classifiers, whereas an inaccurate model might do the opposite. Both stream-based algorithms (e.g., Hoeffding trees) and batch algorithms (e.g., random forest) have been applied in the past.

Metafeature window size: The size of the window on which metafeatures are calculated (denoted as α in Fig. 11.2). Small windows will not be able to capture trends in the data, whereas large windows will not allow for a fast adaptation of classifiers.

Table 11.1: Comparisons between two works on metalearning with data streams

	Rossi et al. (2014)	van Rijn et al. (2014)
Metadata	Generated on current stream	Collected from other streams
Meta-model	Stream model	Batch learner
Learning paradigm	Regression	Classification
$\|A\|$ (algorithms)	2	5

Prediction window size: This determines the number of observations for which the same base-classifier will be used (denoted as w in Fig. 11.2). Setting this number low will result in a potentially too frequent switching of the active classifier, while setting this number high will have the opposite effect. van Rijn et al. (2014) suggested that the window size w should be set to be a multiple of the α size. For example, when α is set to 100, then w can be set to either of the values $\{100, 200, 300, \ldots\}$. This way, all previously seen observations will have an influence on an equal number of prediction windows.

11.2.5 Meta-model

The most important component of metalearning systems is a meta-model. The online algorithm recommendation framework, as outlined in this chapter, has been studied in two works referred to in Table 11.1. This table outlines the main differences between both experimental setups.

The work of van Rijn et al. (2014) requires a meta-dataset built over other data streams, following the usual metalearning framework. The work of Rossi et al. (2014) did not require such a meta-database. The metadata was generated from earlier intervals on the current stream, following the style that is common to AutoML. Although it requires some time for the system to build up a sufficient amount of metadata to be able to construct a good model, this alleviates two problems: (i) the burden of collecting a set of representative metadata on other datasets, and (ii) statistical metafeatures, such as *standard deviation*, can be generated on a per column basis.

Another prominent difference is regarding the meta-model: Rossi et al. (2014) used a stream-based meta-model, which is updated whenever new metadata is gathered; van Rijn et al. (2014) used a batch version of random forests, which is trained once on the metadata and does not require updates. Other differences are the learning paradigm used in the evaluation (regression streams versus classification data streams) and the number of algorithms that have been considered.

11.2.6 Evaluation of metalearning systems for data streams

Evaluation of metalearning and AutoML systems is discussed in Chapter 3. Many of the concepts and methods can be reused for the task of evaluating metalearning and AutoML systems oriented to data streams.

As pointed out in Chapter 3, it is important to distinguish between *base-level performance* and *meta-level performance*. Base-level performance represents the performance

that would be obtained per dataset (or data stream) if the recommendation of the meta-algorithm was followed. Meta-level performance is the performance of the meta-model at the task of selecting a good base-level algorithm (e.g., a classifier if the task is classification).

When dealing with data-streaming tasks which involve intervals (sections of the data stream), it is necessary to introduce the concepts of base-level and meta-level performance. The *base-level performance per interval* determines the performance of the system relative to a specific interval. The *base-level performance across n intervals* (or simply *base-level performance*) returns the average of the above measures across the larger portion of the data stream of interest, given the performance of the base-classifiers that were selected per interval.

The value of the *meta-level performance per interval* is 1 if the correct base-level algorithm was selected, and 0 otherwise. The *meta-level performance across n intervals* (or simply *meta-level performance*) returns the average of the above measure across n intervals.

11.2.7 Baselines

As in other areas of machine learning and metalearning, we need good baselines against which the proposed systems can be compared. In the following we describe some baselines that have been proposed in the past:

Average best classifier (AvBest) is the classifier that obtained the highest base-level accuracy on all the data streams in the training set.

Most frequent best classifier (FreqBest) is the classifier that was the best classifier on most intervals in the meta-dataset.

Best classifier on last interval (Blast) selects, for each interval in the stream, the classifier that performed best on the previous interval. It will be explained formally in Section 11.3.

Oracle is the classifier that always selects the best classifier for each interval, i.e., its meta-performance is 1 (it is not an existing metalearning system). The oracle shows what the base-level performance would be if the meta-model always performed perfectly.

According to some experiments carried out by van Rijn et al. (2015), the variants AvBest and FreqBest have very similar performance. In general, the AvBest variant has a higher base-level accuracy, whereas FreqBest has a better meta-level performance. The Blast baseline basically embodies the no change classifier on the meta-level (Bifet et al., 2013) and is generally a very simple but yet rather strong baseline. Later work shows, somewhat surprisingly, that metalearning systems do not necessarily outperform the Blast baseline (van Rijn et al., 2015).

The oracle is a very useful concept for analyzing the performance of the meta-algorithm. If the gap between the algorithm selection system and the oracle is small, this means that the algorithm selection system performs well.

Besides, the oracle can be used to assess the adequacy of the pre-selected set of base-classifiers. If the oracle performs close to perfect prediction, this means that an adequate set of classifiers has been chosen. If this is not the case, an improvement can be made by adding other appropriate base-level classifiers to the base-level set.

11.2.8 Discussion

Metafeatures have been successfully applied for algorithm selection in data streams. Although there has been a lot of work on metafeatures and on data streams, to the best of our knowledge, only the aforementioned works have studied the interaction between the two.

There is a lot of progress to be made. As pointed out by the authors, not all of the defined baselines have been convincingly beaten by other variants. In particular, the Blast baseline turns out to be a very competitive baseline (more in the next section).

It would be interesting to see how metafeature approaches could be improved by taking advantage of recent developments in *batch metalearning*.

11.3 Data Stream Ensembles

As is well known, ensembles of classifiers have, in general, superior performance to individual classifiers. It thus makes sense to see how ensemble methods can be adapted to data stream settings.

Various types of ensemble methods were discussed in Chapter 9. In this section we will consider only some of those (e.g., bagging) and describe how this architecture can be adapted to the data stream setting. In this setting the most relevant decision is which base-classifiers (ensemble members) should be involved and how to weight their individual votes. However, due to the possible occurrence of concept drift, it is likely that more recent examples will be more relevant than older ones. Moreover, due to the fact that there is a temporal component in the data, we can measure how ensemble members have performed on recent examples and adjust their weight accordingly.

In this section we review several ensemble-based methods adapted for data streams that utilize these observations.

11.3.1 Best classifier on last interval (Blast)

In Section 11.2.7 we already briefly introduced the Blast baseline. It trains an array of diverse classifiers, and measures which of these performed best in the last interval. van Rijn et al. (2015) formalized this as follows: At every new observation in the stream, each ensemble member is assigned a certain score based on its performance on previous observations:

$$P_{win}(f_j, c, w, \mathcal{L}) = 1 - \sum_{i=max(1,c-w)}^{c-1} \frac{\mathcal{L}(f_j(\mathbf{x}_i^{base}), y_i)}{min(w, c-1)}, \tag{11.1}$$

where f_j is the learned labeling function of this specific ensemble member, c is the index of the last seen training example, and w is the number of training examples used to estimate the performance. This method has a certain start-up time (i.e., when w is larger than or equal to c) during which we have fewer observations than we would like to have (i.e., fewer than w observations). Also note that it can only be performed after several labels have been observed (i.e., the index of the current observation $c > 1$).

Finally, $\mathcal{L}$ is a loss function that compares the labels predicted by the ensemble member with the true labels. The simplest version is a zero/one loss function, which returns 0 when the predicted label is correct and 1 otherwise. More complicated loss functions can

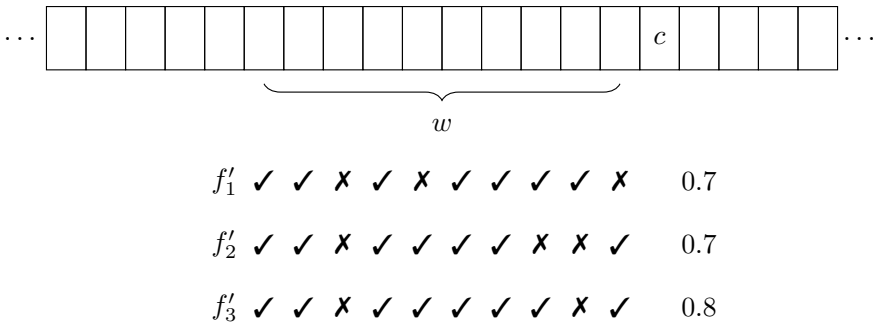

Fig. 11.3: Schematic view of window-based performance estimation. For all classifiers, w flags are stored, each indicating whether it predicted a recent observation correctly

also be incorporated. The outcome of P_{win} is in the range $[0, 1]$, with better-performing classifiers obtaining a higher score.

The classifier that performed best over the last interval of training examples is selected as the *active classifier* (i.e., it gets 100% of the weight). This is illustrated in Figure 11.3, where f'_3 would be selected as the active classifier (AC_c) with respect to example c. Formally, this can be expressed by

$$AC_c = \arg\max_{j \in J} P_{win}(f_j, c-1, \alpha, \mathcal{L}), \tag{11.2}$$

where J is the set of models generated by the ensemble members, c is the index of the current example, α is a parameter to be set by the user (fading factor (discussed in the next subsection)), and $\mathcal{L}$ is a zero/one loss function, giving a penalty of 1 to all misclassified examples. As the authors have shown, this simple baseline outperforms the metafeature-based approaches.

11.3.2 Fading factors

Considering the above observation, a question arises as to whether metalearning approaches can be improved. It is clear that Blast has several drawbacks: (i) it requires the ensemble to store the $w \times |J|$ additional values (recall that J is the set of classifiers that are being considered), which is inconvenient in a data stream setting, where both time and memory are important factors; (ii) it requires the user to tune parameter α, which highly influences performance; and (iii) there is a hard cut-off point, i.e., an observation is either in or out of the window.

These problems can be alleviated using so-called *fading factors*, described by Gama et al. (2013). Fading factors give a high importance to recent predictions, but their importance decreases when they become older. This is illustrated in Figure 11.4. The red (solid) line corresponds to a relatively fast fading factor, and consequently the effect of a given prediction has faded away almost completely after 500 predictions. On the other hand, the blue (dashed) line corresponds to a relatively slow fading factor, where the

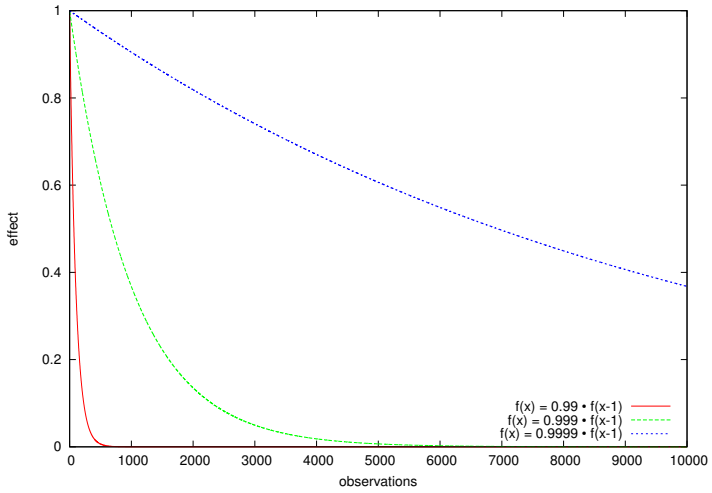

Fig. 11.4: The effect of a prediction after a number of observations, relative to when it was first observed (for various values of α)

effect is still considerably high, even when $10{,}000$ observations have passed. Note that, even though all these functions start at 1, in practice we need to scale this down to $1-\alpha$ in order to constrain the function within the range $[0,1]$. Formally, the weighting function becomes

$$P_{ff}(f_j, c, \alpha, \mathcal{L}) = \begin{cases} 1 & \text{iff } c = 1 \\ P_{ff}(f_j, c-2, \alpha, \mathcal{L}) \cdot \alpha + (1 - \mathcal{L}(f_j(\mathbf{x}_{c-1}^{base}), y_{c-1}^{base})) \cdot (1-\alpha) & \text{otherwise} \end{cases} \tag{11.3}$$

where the meaning of the symbols is similar to Eq. 11.1. The fading factor α (range $[0,1]$) determines at what rate the historical performance becomes irrelevant, and its value can be set (or tuned) by the user. A value close to 0 will result in rapid changes in estimated performance, whereas a value close to 1 will keep them more stable. The outcome of P_{ff} is in the range $[0,1]$, with better-performing classifiers obtaining a higher score.

This notion has been used in the following works: Kolter and Maloof (2007) describe this weighting strategy and use it to build *dynamic ensembles*, that is, ensembles that grow in size whenever the current set of ensemble members make a mistake. They also introduce methods to prune the ensemble again.

van Rijn et al. (2018) used this weighting strategy to create heterogeneous ensembles of fixed size that includes different modeltypes. However, their experiments have shown that Blast achieved better performance.

Cerqueira et al. (2019) developed a method that operates in the regression setting. The method uses a meta-model that predicts the performance of each base-level model for the next observation. Based on these predictions, the weights of base-level models are determined and suitably normalized. It would be interesting to see whether the system could be improved by incorporating metafeatures.

11.3.3 Heterogeneous ensembles for feature drift

Nguyen et al. (2012) introduce the notion of feature drift. This arises when the set of most important features used for predicting the class changes. Besides, they show: (i) concept drift may give rise to feature drift, (ii) concept drift does not necessarily lead to feature drift, and (iii) feature drift occurs at a slower rate than concept drift.

So one could argue that systems that utilize stream ensembles should incorporate drift detection and feature selection. The method proposed by the authors employs a method for feature selection (i.e., the fast correlation-based filter algorithm by Yu and Liu (2003)). If a new set of features is detected, the model that performs badly is dropped from the ensemble and a new model is trained.

11.3.4 Considerations regarding the choice of base-classifiers

In the previous section we focused on how to weight the votes of individual classifiers based on recent data. Another important question is which base-level models should be considered in the first place. This issue was addressed in Chapter 8 (Section 8.3), where we discussed which base-level algorithms should be considered for inclusion in a given portfolio. Many of the concepts presented there are relevant also for the constitution of ensembles.

One such concept was *classifier output difference (COD)* (Peterson and Martinez, 2005), which can be used to detect whether two classifiers generate similar predictions. Lee and Giraud-Carrier (2011) used this function to build a hierarchical clustering of classifiers. Classifiers that generate similar predictions appear to be clustered together, and vice versa.

This idea was followed by van Rijn et al. (2018) to create a clustering of various stream classifiers from the MOA framework. The resulting dendrogram is shown in Figure 11.5. The assumption is that classifiers that are clustered far away from each other are *diverse* and hence would work well together in an ensemble.

11.3.5 Discussion

A popular approach to the repeated algorithm selection problem is the use of ensembles that either change (i) their set of base-learners, or (ii) the way the votes of the individual base-learners are weighted. These decisions are often based on how the classifiers perform on recent examples. The performance is often better than competitive ensembles that do not utilize this property. Unfortunately, there has been no general comparison between the various ensemble methods discussed in this chapter.

11.4 Recurring Meta-level Models

Another way to leverage properties of data streams is to store the meta-level models generated on old parts of the stream and learn when to reuse them. Gama and Kosina (2014) used this scheme in a sensor network consisting of several geographically distributed sensors, each producing a high-speed data stream. They measure some quantity of interest, for example, the electricity demand in a particular geographic region. Companies are interested in predicting the demand of electricity for a certain time horizon,

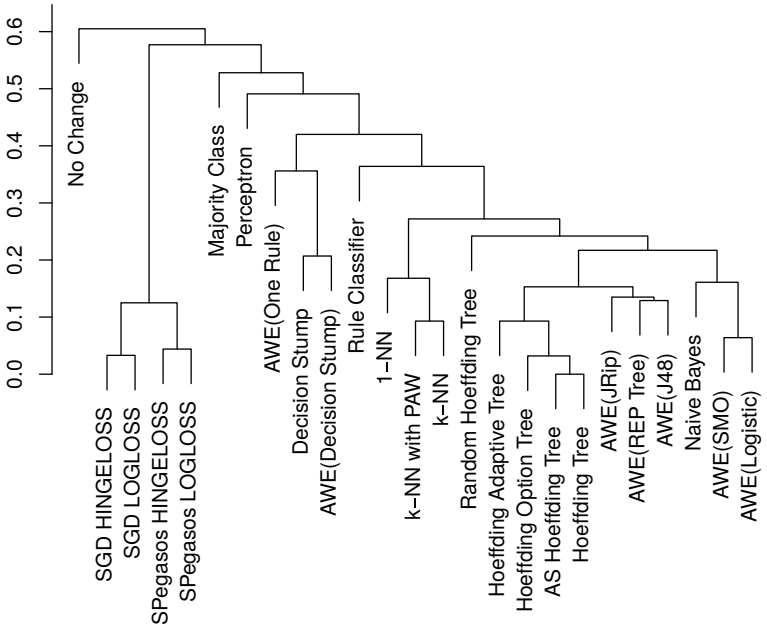

Fig. 11.5: Hierarchical clustering of stream classifiers

e.g., an hour later. At a given point in time, the model makes a prediction for the electricity demand an hour later. When this time point occurs, the sensor measures the actual electricity demand and can thus determine the loss obtained by the model. Consumption patterns change (e.g., from winter to summer) and, due to seasonality, might reoccur.

Several authors have used ideas of metalearning for this scenario. We will review two of these works here.

11.4.1 Accuracy-weighted ensemble

The approach proposed by Wang et al. (2003) is to separate the stream into windows of a given fixed size, and train a new classifier on each of these windows. During prediction, each classifier votes for a certain class label. The votes are weighted by their accuracy.

In addition to convincing empirical results, they provide a formal proof regarding the error rate of an ensemble of k classifiers, each trained on the data points from different windows, compared with the classifier that is trained on all data points from the last k windows. Their proof shows that the error rate of the ensemble will always be equal to or lower than the error rate of the individual classifier.

One of the most important advantages of this approach is that it does not come at the expense of additional training time, as each observation is used as training input to only one model, which is a cheap way of turning a single classifier into an ensemble. Read et al. (2012) noted that this is an opportunity to gain additional performance from batch classifiers, which must be trained on windows of data.

There are several practical considerations that need to be taken into account when employing this method. First, there is a question of the ensemble size, that is, how many members should be used. This is obviously a hyperparameter under the control of the user. Normally, higher values would be preferred, as these often lead to better performance. However, memory usage is another criterion to take into consideration when setting this hyperparameter.

Second, it is important to determine which members should be kept in memory. Once the ensemble has generated more members than it is possible to keep in memory, it is necessary to determine which members should be dropped. These can, for example, be the oldest members in memory, or the members with the worst performance on the current window (or on all windows).

Finally, the window size should be set appropriately. When the individual ensemble members are trained on a window that is too small, the models will not be able to accurately classify the data. When, on the other hand, the window is too large, the individual ensemble members will not be able to respond well to the dynamics of changing concepts.

11.4.2 Two-layered architecture

Gama and Kosina (2014) proposed a two-layered architecture to handle the recurrence of concepts in the data. The first layer (*induction layer*) is responsible for doing the actual classification, whereas the second layer (*control layer*) is responsible for checking the applicability of the old, stored models. Let us see the role of each layer in more detail:

Induction layer This layer consists of a classifier that is specialized in the actual classification task. Whenever a new measurement is done, the classifier is trained on the most recent data.

Control layer This layer consists of a classifier that is specialized in determining whether the classifier from the induction layer will make the correct prediction. As such, it is involved in a binary classification task.

The classifier in the control layer can determine how well the corresponding classifier in the induction layer will work on a given window of data points.

Besides, a change detector is also used to estimate whether a concept drift occurred. The authors state that, although any change detector could be used, they suggest SPC, as it is also able to give a warning signal (Gama et al., 2004). Whenever a change in concept is detected, the meta-model has to choose between the following options: (i) train a new model, or (ii) activate one of the previously learned models.

In order to distinguish between the two options, all previously trained models will make predictions for the recent set of data points. This set can be of fixed size or include every data point since the drift detector signaled a warning. Depending on the prediction horizon, some of these data points might not yet have the associated label.[4] For these data points, the classifier in the control layer can be used to estimate whether they would have been classified correctly.

The classifier that performed best on this recent set of data points will be used to classify the new observations, provided its performance is better than a given threshold. So one of the stored models can be reused here. If none of the previously trained models performs better than this threshold, a new model is trained.

[4] In the most optimistic case, this is only one data point; in the most pessimistic case, this can be all the data points.

11.5 Challenges for Future Research

There remain many avenues for further research, and many questions to be answered. First, as discussed before and summarized in Table 11.1, we can generate and leverage metadata from earlier in the current stream alone, or we can do so from a larger set of previously seen data streams, when these are available. These two methods are complementary, but currently there does not yet exist clear analyses or comparisons between these approaches that explain when and how to use which approach, or whether both can be combined.

Such an analysis could elucidate several key questions: (i) which properties of a given setup contribute to its success (e.g., which types of learners to use), (ii) when a given setup works well (i.e., on what types of data), and (iii) why a given setup works well on a certain type of data.

Indeed, one of the most surprising results of the metafeature-based approaches is that they did not seem to be able to consistently outperform a simple baseline Blast. Although it is clear that this baseline would not work in highly volatile streams, there is currently no convincing argument for using metafeature-based approaches instead of Blast. Still, an interesting question is whether the metafeatures could still enhance the performance in some way, or whether better metafeatures could be included.

A second key question is whether the meta-problem should be modeled by stream learning or batch learning. In other words: should we adapt existing meta-learning and AutoML methods that use batch learners to work in the stream setting, or should we focus on techniques that do meta-learning and AutoML on stream learning algorithms directly? This should also include the tuning of hyperparameters and the inclusion of preprocessing in pipelines, which was not yet done in the studies discussed earlier in this chapter.

While, to the best of our knowledge, there do not yet exist AutoML tools that operate over stream learning algorithms, Celik and Vanschoren (2020) provide an analysis of how existing AutoML methods can be adapted to the stream setting, for instance, by restarting the AutoML process after concept drift is detected (with or without a warm start from the previous best configurations), or simply by retraining the best model(s) on the latest batches of data. This analysis shows that both Bayesian optimization and evolutionary approaches can handle concept drift well, given an appropriate adaptation strategy and forgetting mechanism. It also finds that different drift characteristics (e.g., gradual versus sudden drift) affect learning algorithms in different ways, and that different adaptation strategies may be needed to optimally deal with them. This shows that there is ample room to improve existing AutoML systems, and even to design entirely new AutoML systems that naturally adapt to concept drift.

Finally, many tools have been developed to facilitate the task of conducting meta-learning and AutoML experiments, such as OpenML (Vanschoren et al., 2014), discussed in Chapter 16. OpenML also contains several stream datasets, with and without concept drift, and it is integrated with the MOA stream mining library. As such, it provides a great starting point for novel research into metalearning and AutoML on data streams, which would certainly have a positive impact on the field, especially if more stream mining datasets can be made publicly available and further stream mining libraries are integrated to facilitate experimentation.

References

Beringer, J. and Hüllermeier, E. (2007). Efficient instance-based learning on data streams. *Intelligent Data Analysis*, 11(6):627–650.

Bifet, A., Frank, E., Holmes, G., and Pfahringer, B. (2012). Ensembles of restricted Hoeffding trees. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 3(2):30.

Bifet, A. and Gavaldà, R. (2007). Learning from Time-Changing Data with Adaptive Windowing. In *SDM*, volume 7, pages 139–148. SIAM.

Bifet, A. and Gavaldà, R. (2009). Adaptive learning from evolving data streams. In *Advances in Intelligent Data Analysis VIII*, pages 249–260. Springer.

Bifet, A., Holmes, G., Kirkby, R., and Pfahringer, B. (2010). MOA: Massive Online Analysis. *J. Mach. Learn. Res.*, 11:1601–1604.

Bifet, A., Read, J., Žliobaitė, I., Pfahringer, B., and Holmes, G. (2013). Pitfalls in benchmarking data stream classification and how to avoid them. In *Machine Learning and Knowledge Discovery in Databases*, pages 465–479. Springer.

Bottou, L. (2004). Stochastic learning. In *Advanced Lectures on Machine Learning*, pages 146–168. Springer.

Breiman, L. (1996). Bagging predictors. *Machine Learning*, 24(2):123–140.

Celik, B. and Vanschoren, J. (2020). Adaptation strategies for automated machine learning on evolving data. *arXiv preprint arXiv:2006.06480*.

Cerqueira, V., Torgo, L., Pinto, F., and Soares, C. (2019). Arbitrage of forecasting experts. *Machine Learning*, 108(6):913–944.

Domingos, P. and Hulten, G. (2000). Mining High-Speed Data Streams. In *Proceedings of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 71–80.

Domingos, P. and Hulten, G. (2003). A general framework for mining massive data streams. *Journal of Computational and Graphical Statistics*, 12(4):945–949.

Finn, C., Rajeswaran, A., Kakade, S., and Levine, S. (2019). Online meta-learning. In Chaudhuri, K. and Salakhutdinov, R., editors, *Proceedings of the 36th International Conference on Machine Learning*, ICML'19, pages 1920–1930. JMLR.org.

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

Gama, J., Medas, P., Castillo, G., and Rodrigues, P. (2004). Learning with drift detection. In *SBIA Brazilian Symposium on Artificial Intelligence*, volume 3171 of *Lecture Notes in Computer Science*, pages 286–295. Springer.

Gama, J., Sebastião, R., and Rodrigues, P. P. (2009). Issues in evaluation of stream learning algorithms. In *Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 329–338. ACM.

Gama, J., Sebastião, R., and Rodrigues, P. P. (2013). On evaluating stream learning algorithms. *Machine Learning*, 90(3):317–346.

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

Lee, J. W. and Giraud-Carrier, C. (2011). A metric for unsupervised metalearning. *Intelligent Data Analysis*, 15(6):827–841.

Nguyen, H.-L., Woon, Y.-K., Ng, W.-K., and Wan, L. (2012). Heterogeneous Ensemble for Feature Drifts in Data Streams. In *Advances in Knowledge Discovery and Data Mining*, pages 1–12. Springer.

Oza, N. C. (2005). Online bagging and boosting. In *Systems, Man and Cybernetics, 2005 IEEE International Conference*, volume 3, pages 2340–2345. IEEE.

Peterson, A. H. and Martinez, T. (2005). Estimating the potential for combining learning models. In *Proc. of the ICML Workshop on Meta-Learning*, pages 68–75.

Pfahringer, B., Bensusan, H., and Giraud-Carrier, C. (2000). Meta-learning by landmarking various learning algorithms. In Langley, P., editor, *Proceedings of the 17th International Conference on Machine Learning*, ICML'00, pages 743–750.

Pfahringer, B., Holmes, G., and Kirkby, R. (2007). New options for Hoeffding trees. In *AI 2007: Advances in Artificial Intelligence*, pages 90–99. Springer.

Read, J., Bifet, A., Pfahringer, B., and Holmes, G. (2012). Batch-Incremental versus Instance-Incremental Learning in Dynamic and Evolving Data. In *Advances in Intelligent Data Analysis XI*, pages 313–323. Springer.

Rossi, A. L. D., de Leon Ferreira, A. C. P., Soares, C., and De Souza, B. F. (2014). MetaStream: A meta-learning based method for periodic algorithm selection in time-changing data. *Neurocomputing*, 127:52–64.

Schapire, R. (1990). The strength of weak learnability. *Machine Learning*, 5(2):197–227.

Shalev-Shwartz, S., Singer, Y., Srebro, N., and Cotter, A. (2011). Pegasos: primal estimated sub-gradient solver for SVM. *Mathematical Programming*, 127(1):3–30.

van Rijn, J. N. (2016). *Massively collaborative machine learning*. PhD thesis, Leiden University.

van Rijn, J. N., Holmes, G., Pfahringer, B., and Vanschoren, J. (2014). Algorithm Selection on Data Streams. In *Discovery Science*, volume 8777 of *LNCS*, pages 325–336. Springer.

van Rijn, J. N., Holmes, G., Pfahringer, B., and Vanschoren, J. (2015). Having a Blast: Meta-Learning and Heterogeneous Ensembles for Data Streams. In *2015 IEEE International Conference on Data Mining (ICDM)*, pages 1003–1008. IEEE.

van Rijn, J. N., Holmes, G., Pfahringer, B., and Vanschoren, J. (2018). The online performance estimation framework: heterogeneous ensemble learning for data streams. *Machine Learning*, 107(1):149–167.

Vanschoren, J., van Rijn, J. N., Bischl, B., and Torgo, L. (2014). OpenML: networked science in machine learning. *ACM SIGKDD Explorations Newsletter*, 15(2):49–60.

Wang, H., Fan, W., Yu, P. S., and Han, J. (2003). Mining Concept-Drifting Data Streams using Ensemble Classifiers. In *KDD*, pages 226–235.

Yu, L. and Liu, H. (2003). Feature selection for high-dimensional data: A fast correlation-based filter solution. In *Proceedings of the 20th International Conference on Machine Learning*, ICML'03, pages 856–863.

Zhang, P., Gao, B. J., Zhu, X., and Guo, L. (2011). Enabling fast lazy learning for data streams. In *2011 IEEE 11th International Conference on Data Mining (ICDM)*, pages 932–941. IEEE.