5 Data Streams

Real-time analysis of data streams is a key area of data mining research. Many real world collected data are in fact streams where observations come in one by one, and algorithms processing these are often subject to time and memory constraints. This chapter focusses on ensembles for data streams. Ensembles of classifiers are among the best performing classifiers available in many data mining applications, including the mining of data streams. Rather than training one classifier, multiple classifiers are trained, and their predictions are combined according to a given voting schedule. An important prerequisite for ensembles to be successful is that the individual models are diverse. One way to vastly increase the diversity among the models is to build an *heterogeneous* ensemble, comprised of fundamentally different model types. However, most ensembles developed specifically for the dynamic data stream setting rely on only one type of base-level classifier, most often Hoeffding Trees. This chapter focusses on meta-learning techniques to combine the intrinsically different models to heterogeneous ensembles.

OpenML played a vital role in the development of these techniques. In order to build a good heterogeneous ensemble technique, two questions need to be addressed: which classifiers to use and how to weight their individual votes. Having stored all experimental results of possible base-classifiers, it becomes a matter of database queries to test certain combinations. Classifiers that appear to complement each other well, can then be combined into a heterogeneous ensemble. Furthermore, most data stream research involves only few streams. Having a large collection of datasets and data streams, OpenML is an indispensable factor in scaling up this direction of research. Finally, all these results organised together foster new research questions and discoveries; Chapter 5.6 shows an example of this. OpenML is in this sense the open experiment database of data stream research.

5.1 Introduction

Modern society produces vast amounts of data coming, for instance, from sensor networks and various text sources on the internet. Much of this data is in fact a stream where observations come in one by one. Real-time analysis of such data streams is a key area of data mining research, typically the algorithms processing these are subject to time and memory constraints. The research community developed a large number of machine learning algorithms capable of online modelling general trends in stream data and make accurate predictions for future observations. In many applications, *ensembles* of classifiers are the most accurate classifiers available. Rather than building one model, a variety of models are generated that all vote for a certain class label.

One way to vastly improve the performance of ensembles is to build *heterogeneous* ensembles, consisting of models generated by different techniques, rather than *homogeneous* ensembles, in which all models are generated by the same technique. One technique to build such a heterogeneous ensemble is Stacking [53, 163]. It has been extensively analysed in classical batch data mining applications. As the underlying techniques upon which Stacking relies can not be trivially transferred to the data stream setting, there are currently no successful heterogeneous ensemble techniques in the data stream setting. State of the art heterogeneous ensembles in a data stream setting typically rely on meta-learning [125, 133]. These approaches both require the extraction of computationally expensive meta-features and yield marginal improvements.

In this work we introduce an elegant technique that natively combines heterogeneous models in the data stream setting. 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 5.1. We propose a way to measure the performance of ensemble members on recent observations and combine their votes based on this.

This chapter introduces the following concepts. First, we describe Online Performance Estimation, a framework that provides *dynamic* weighting of the votes of individual ensemble members across the stream [128]. Utilizing this framework, we introduce a new ensemble technique that combines heterogeneous models. The online performance estimation framework can also be used to generate traditional meta-features, which can also be used in a predictive manner. Finally, an experiment covering 60 data streams shows that both techniques are competitive with state of the art ensembles, while requiring significantly less run time.

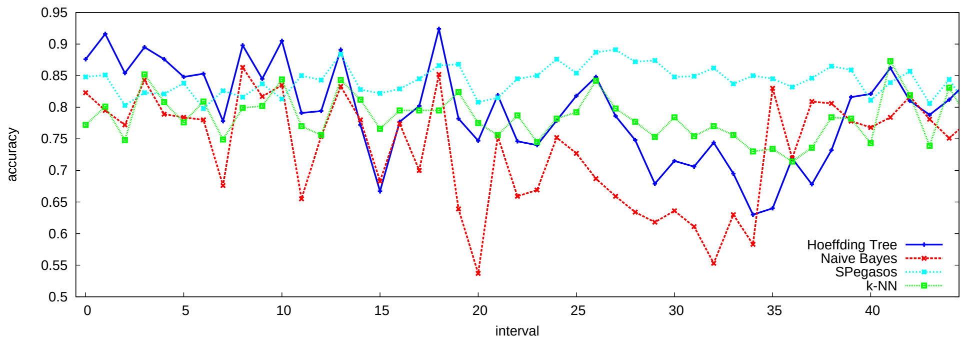


Figure 5.1: Performance of four classifiers on intervals (size $1{,}000$) of the 'electricity' dataset. Each data point represents the accuracy of a classifier on the most recent interval.

The remainder of this chapter is organised as follows. Chapter 5.2 surveys data streams, and Chapter 5.3 introduces the proposed methods. Chapter 5.4 and Chapter 5.5 describe various experiments, covering the performance of the proposed methods and the relevance of online performance estimation. Chapter 5.6 describes an interesting pattern that was discovered by mining the experiment repository. Chapter 5.7 concludes.

5.2 Related Work

It has been widely recognized that data stream mining differs significantly from conventional batch data mining [13, 14, 40, 57, 118]. In the conventional batch setting, a finite amount of stationary data is provided and the goal is to build a model that fits the data as well as possible. When working with data streams, we should expect an infinite amount of data, where observations come in one by one and are being processed in that order. Furthermore, the nature of the data can change over time, known as *concept drift*. Classifiers (and other modelling techniques) operating on streams of data should be able to detect when a learned model becomes obsolete and update it accordingly.

Common Approaches. Some batch classifiers can be trivially adapted to a data stream setting. Examples are k Nearest Neighbour [10, 165], Stochastic Gradient Descent [18] and SPegasos (Stochastic Primal Estimated sub-GrAdient SOlver for SVMs) [139]. 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, 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 [39]. 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 over others. As Hoeffding Trees seem to work very well in practice, many variants have been proposed, such as Hoeffding Option Trees [109], Adaptive Hoeffding Trees [12] and Random Hoeffding Trees [15].

Finally, 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, not before w new observations are made and the model is retrained. Read et al. [118] compare the performance of these *batch-incremental* classifiers with common data stream classifiers, and conclude that the overall performance is equivalent, although the batch-incremental classifiers generally use more resources (time and memory).

Ensembles. Ensemble techniques train multiple classifiers on a set of weighted training examples, and these weights can vary for different classifiers. In order to classify test examples, all individual models produce a prediction, also called a *vote*, and the final prediction is made according to a predefined voting schema, e.g., the class with the most votes is selected.

Bagging [23] exploits the instability of classifiers by training them on different *bootstrap replicates*: resamplings (with replacement) of the training set. Effectively, the training sets for various classifiers differ by the weights of their training examples. Online Bagging [101] operates on data streams by drawing the weight of each example from a $Poisson(1)$ distribution, which converges to the behaviour of the 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 for use in a Bagging scheme. However, empirical evaluation suggests that this yields good results nonetheless.

Boosting [135] is a technique that sequentially trains multiple classifiers, in which more weight is given to examples that where misclassified by earlier classifiers. Online Boosting [101] applies this technique on data streams by assigning more weight to training examples that were misclassified by previously trained classifiers in the ensemble. The Accuracy Weighted Ensemble (AWE) works well in combination with batch-incremental classifiers [161]. It maintains multiple recent models trained on different batches of the data stream, and weights the votes of each model based on the expected predictive accuracy.

Concept drift. One property of data streams is that the underlying concept that is being learned can change over time (e.g., [161]). This is called concept drift. Some of the aforementioned methods naturally deal with concept drift. For instance, k Nearest Neighbour maintains a number of w recent examples, substituting each example after w new examples have been observed. *Change detectors*, such as Drift Detection Method (DDM) [55] and Adaptive Sliding Window Algorithm (ADWIN) [11] are stand-alone techniques that detect concept drift and can be used in combination with any stream classifier. Both rely on the assumption that classifiers improve (or at least maintain) their accuracy when trained on more data. When the accuracy of a classifier drops in respect to a reference window, this could mean that the learned concept is outdated, and a new classifier should be built. The main difference between DDM and ADWIN is the way they select the reference window. Furthermore, classifiers can have built-in drift detectors. For instance, Ultra Fast Forest of Trees [56] are Hoeffding Trees with a built-in change detector for every node. When an earlier made split turns out to be obsolete, a new split can be generated.

It has been recognised that some classifiers recover faster from sudden changes of concepts than others. The *recovery analysis* framework measures the ability of classifiers to recover from concept drift [138]. It distinguishes instance-based classifiers that operate directly on the data (e.g., k-NN) and model-based classifiers, that build and maintain a model (e.g., tree algorithms, fuzzy systems). Their experimental results suggest, quite naturally, that instance-based classifiers generally have a higher capability to recover from concept drift than model-based classifiers.

Evaluation. As data from streams is non-stationary, the well-known cross-validation procedure for estimating model performance is not suitable. A commonly accepted estimation procedure is the *prequential method* [57], in which each example is first used to test the current model, and afterwards (either directly after testing or after a delay) becomes available for training. An advantage of this method is that it is tested on all data, and therefore no specific holdout set is needed.

Meta-Learning. Meta-learning approaches on data streams often train multiple base-classifiers and a meta-model decides, for each data point, which of the base-learners will make a prediction. Meta-knowledge can be obtained from two sources. The first option is to extract meta-knowledge obtained from earlier in the stream [133]. One advantage is that there is no representative meta-dataset required: we can assume that the data from earlier in the stream is adequate. The disadvantage is that it requires more run time, as a meta-model needs to be trained while processing the stream. The other option is to extract meta-knowledge from other data streams [125, 126, 128]. The advantage is that this is relatively fast: the meta-model can be trained a priori and does not require additional resources while processing the stream.

Another technique uses meta-learning on time series with recurrent concepts: when concept drift is detected, a meta-learning algorithm decides whether a model trained previously on the same stream could be reused, or whether the data is so different from before that a new model must be trained [54]. Nguyen et al. [94] propose a method that combines feature selection and heterogeneous ensembles; members that perform poorly according to a drift detector can be replaced.

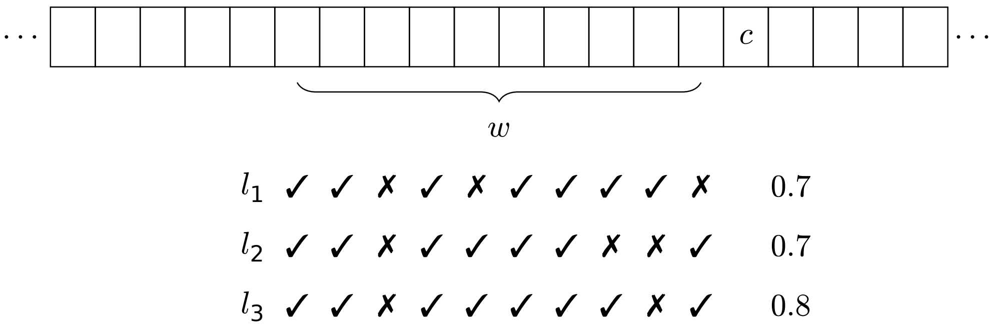
5.3 Methods

Traditional Machine Learning problems consist of a number of *examples* that are observed in arbitrary order. In this work we consider classification problems. Each example $e = (\mathbf{x}, l(\mathbf{x}))$ is a tuple of p predictive attributes $\mathbf{x} = (x_1, \ldots, x_p)$ and a target attribute $l(\mathbf{x})$. A data set is an (unordered) set of such examples. The goal is to approximate a labelling function $l : \mathbf{x} \rightarrow l(\mathbf{x})$. In the data stream setting the examples are observed in a given order, therefore each data stream S is a sequence of examples $S = (e_1, e_2, e_3, \ldots, e_n, \ldots)$, where n is the number of observations (possibly infinite). Consequently, S_i refers to the i^{th} example in data stream S. The set of predictive attributes of that example is denoted by PS_i, likewise $l(PS_i)$ maps to the corresponding label. Furthermore, the labelling function that needs to be learned can change over time due to concept drift. When applying an ensemble of classifiers, the most relevant variables are which base-classifiers (members) to use and how to weight their individual votes.

5.3.1 Online Performance Estimation

In most common ensemble approaches, all base-classifiers are given the same weight (as done in Bagging and Boosting) or their predictions are otherwise combined to optimise the overall performance of the ensemble (as done in Stacking). An important property of the data stream setting has not been taken into account: due to the possible occurrence of concept drift it is likely that in most cases recent examples are more relevant than older ones. Moreover, due to the fact that there is a temporal component in the data, we can actually measure how ensemble members have performed on recent examples, and adjust their weight in the voting accordingly. In order to estimate the performance of a classifier on recent data, van Rijn et al. [127] proposed:

$$P_{win}(l', c, w, L) = 1 - \sum_{i=max(1,c-w)}^{c-1} \frac{L(l'(PS_i), l(PS_i))}{min(w, c-w)} \tag{5.1}$$



. . .
c
. . .
w
l_1 ✓ ✓ ✗ ✓ ✗ ✓ ✓ ✓ ✓ ✗ 0.7
l_2 ✓ ✓ ✗ ✓ ✓ ✓ ✓ ✗ ✗ ✓ 0.7
l_3 ✓ ✓ ✗ ✓ ✓ ✓ ✓ ✓ ✗ ✓ 0.8

Figure 5.2: Schematic view of Windowed Performance Estimation. For all classifiers, w flags are stored, each flag indicating whether it predicted a recent observation correctly.

where l' is the learned labelling function of an ensemble member, c is the index of the example we want to predict and w is the number of training examples over which we want to estimate the performance of ensemble members. Finally, L is a loss function that compares the labels predicted by the ensemble member to the true labels. The most simple version is a zero/one loss function, which returns 0 when the predicted label is correct and 1 otherwise. More complicated loss functions can also be incorporated. The outcome of P_{win} is in the range $[0, 1]$, with better performing classifiers obtaining a higher score. The performance estimates for the ensemble members can be converted into a weight for their votes, at various points over the stream. For instance, the best performing members at that point could receive the highest weights. Figure 5.2 illustrates this.

There are a few drawbacks to this approach. Firstly, it requires the ensemble to store the $w \times n$ additional values, which is inconvenient in a data stream setting, where both time and memory are important factors. Secondly, it requires the user to tune a parameter which highly influences performance. Lastly, there is a hard cut-off point, i.e., an observation is either in or out of the window. What we would rather model is that the most recent observations are given most weight, and gradually lower this for less recent observations.

In order to address these issues, *fading factors* (further described in, e.g., Gama et al. [58]) can be used as an alternative. Fading factors give a high importance to recent predictions, whereas the importance fades away when they become older. This is illustrated by Figure 5.3. The red (solid) line shows a relatively fast fading factor, where the effect of a given prediction has already faded away almost completely after 500 new observations, whereas the blue (dashed) line shows a relatively slow fading

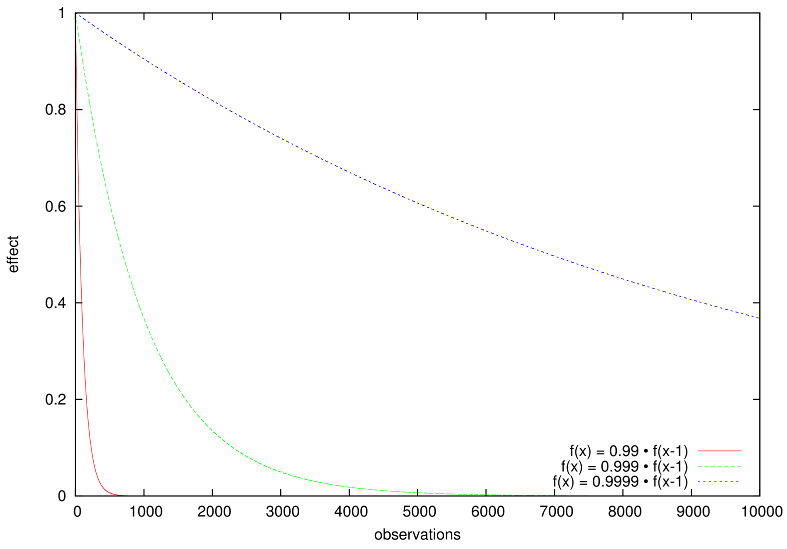
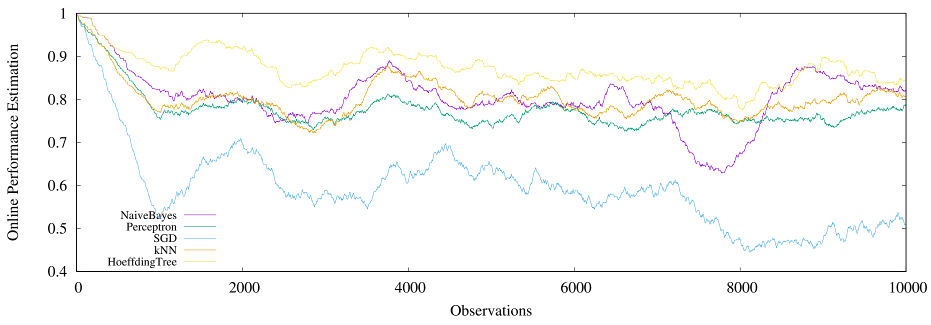


Figure 5.3: The effect of a prediction after a number of observations, relative to when it was first observed (for various values of α).

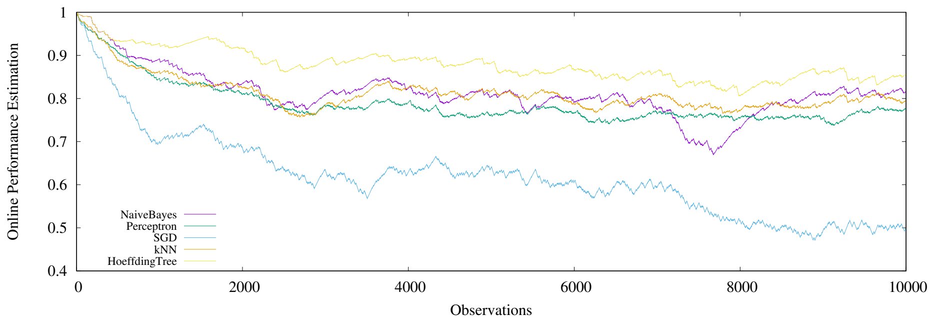
factor, where the effect of an observation is still considerably high, even when $10{,}000$ observations have passed in the meantime. Note that even though all these functions start at 1, in practise we need to scale this down to $1 - \alpha$, in order to constrict the complete function within the range $[0, 1]$. Putting this all together, we get:

$$P(l', c, \alpha, L) = \begin{cases} 1 & \text{iff } c = 0 \\ P(l', c-1, \alpha, L) \cdot \alpha + (1 - L(l'(PS_c), l(PS_c))) \cdot (1 - \alpha) & \text{otherwise} \end{cases} \tag{5.2}$$

where, similar to Eq. 5.1, l' is the learned labelling function of an ensemble member, c is the index of the last seen training example and L is a loss function that compares the labels predicted by the ensemble member to the true labels. Fading factor α (range $[0, 1]$) determines at what rate historic performance becomes irrelevant, and is to be tuned by the user. A value close to 0 will allow for rapid changes in estimated performance, whereas a value close to 1 will keep them rather stable. The outcome of P is in the range $[0, 1]$, with better performing classifiers obtaining a higher score. In Chapter 5.5 we will see that the fading factor parameter is more robust and easier to tune than the window size parameter. When building an ensemble based upon Online Performance Estimation, we can now choose between a Windowed approach (Eq. 5.1) and Fading Factors (Eq. 5.2). Figure 5.4 shows how the estimated performance for each base-classifier evolves on the start of the electricity data stream. Both



(a) Windowed, window size 1000



(b) Fading Factors, $\alpha = 0.999$

Figure 5.4: The estimated performance of each algorithm given previous examples, measured on the start of the electricity data stream.

figures expose similar trends: apparently, on this data stream the Hoeffding Tree classifier performs best and the Stochastic Gradient Descent algorithm performs worst. However, both approaches differ subtly in the way the performance of individual classifiers are measured. The Windowed approach contains many spikes, whereas the Fading Factor approach seems more stable.

5.3.2 Ensemble Composition

In order for an ensemble to be successful, the individual classifiers should be both accurate and diverse [64]. When employing a homogeneous ensemble, choosing an appropriate base-learner is an important decision. For heterogeneous ensembles this is even more true, as we have to choose a set of base-learners.

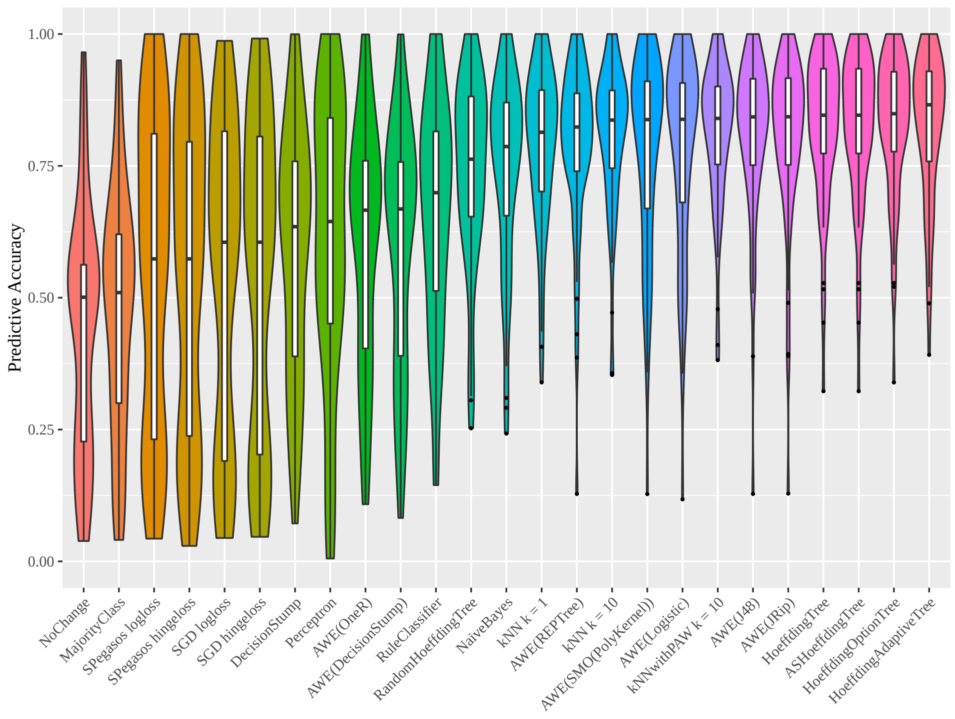
Figure 5.5 shows some basic performance results of the classifiers on 60 data streams. Figure 5.5a shows a violin plot of the performance of all classifiers, with a box plot in the middle. The classifiers are sorted by median accuracy. Two common Data Stream baseline methods, the No Change classifier and the Majority Class classifier, end at the bottom of the ranking based on accuracy. This indicates that most of the selected data streams are both balanced (in terms of class labels) and do not have high auto-correlation. In general, tree-based methods seem to perform best.

Figure 5.5b shows the result of a statistical test on the base-classifiers. Classifiers are sorted by their average rank (lower is better). Classifiers that are connected by a horizontal line are statistically equivalent. The results confirm some of the observations made based on the violin plots, e.g., the baseline models (Majority Class and No Change) perform worst; also other simple models such as the Decision Stumps and OneRule (which is essentially a Decision Stump) are inferiour to the tree-based models. Oddly enough, the instance incremental Rule Classifier does not compete at all with the Batch-incremental counterpart (AWE(JRIP)).

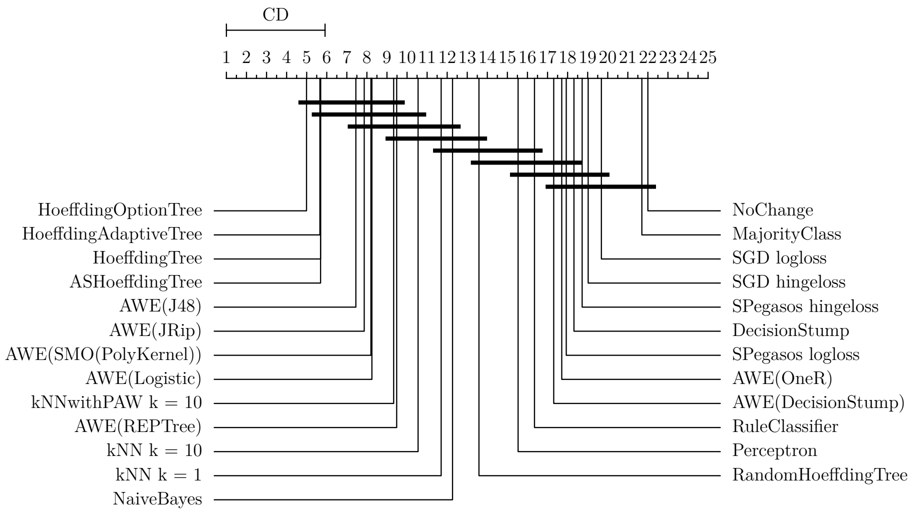
A dendrogram like the one in Figure 4.14 (page 69) can serve to select a collection of diverse and well performing ensemble members. Classifier Output Difference (COD) is a metric which measures the number of observations on which a pair of classifiers yields a different prediction [106]. This can be used to ensure diversity among the ensemble members [85]. A threshold on the Classifier Output Difference needs to be determined, selecting representative classifiers from all pairs of clusters with a distance higher than this threshold. A higher threshold would result in a smaller set of classifiers, and vice versa. For example, if we set the threshold to 0.2, we end up with an ensemble consisting of classifiers from 11 clusters. The ensemble will consist of one representative classifier from each cluster, which can be chosen based on accuracy, run time, a combination of the two (see, e.g., [21]), or any arbitrary criterion. Which exact criterion to use is outside the scope of this thesis, however in the experiments we will use a combination of accuracy and run time. Clearly, when using this technique in experiments, the dendrogram should be constructed in a leave-one-out setting: it can be created based on all data streams except for the one that is being tested.

5.3.3 BLAST

BLAST (short for **b**est **last**) is an ensemble embodying the performance estimation framework. Ideally, it consists of a group of diverse base-classifiers. These are all trained using the full set of available training observations. For every test example, it selects one of its members to make the prediction. This member is referred to as the *active classifier*. The active classifier is selected based on Online Performance Estimation: the classifier that performed best over the set of w previous trainings examples is selected as the active classifier (i.e., it gets 100% of the weight), hence the name.



(a) Performance of base-classifiers



(b) Nemenyi test, $\alpha = 0.05$

Figure 5.5: Performance of 25 data stream classifiers based on 60 data streams.

Algorithm 5.1 BLAST (Learning)

```
Require: Set of ensemble members $M$, Loss function $L$ and Fading Factor $\alpha$
1: Initialise ensemble members $m_j$, with $j \in \{1, 2, 3, \ldots, |M|\}$
2: Set $p_j = 1$ for all $j$
3: for all training examples $e = (\mathbf{x}, l(\mathbf{x}))$ do
4:     for all $m_j \in M$ do
5:         $l'_j(\mathbf{x}) \leftarrow$ predicted label of $m_j$ on attributes $\mathbf{x}$ of current example $e$
6:         $p_j \leftarrow p_j \cdot \alpha + (1 - L(l'_j(\mathbf{x}), l(\mathbf{x}))) \cdot (1 - \alpha)$
7:         Update $m_j$ with current example $e$
8:     end for
9: end for
```

Formally,

$$AC_c = \underset{m_j \in M}{\arg\max}\, P(m_j, c-1, \alpha, L) \tag{5.3}$$

where M 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) and L is a zero/one loss function, giving a penalty of 1 to all misclassified examples. Note that the performance estimation function P can be replaced by any measure. For example, if we would replace it with Equation 5.1, we would get the exact same predictions as reported by van Rijn et al. [127]. When multiple classifiers obtain the same estimated performance, any arbitrary classifier can be selected as active classifier. The details of this method are summarised in Algorithm 5.1.

Line 2 initialises a variable that keeps track of the estimated performance for each base-classifier. Everything that happens from lines 5–7 can be seen as an internal prequential evaluation method. At line 5 each training example is first used to test all individual ensemble members on. The predicted label is compared against the true label $l(\mathbf{x})$ on line 7. If it predicts the correct label then the estimated performance for this base-classifier will increase; if it predicts the wrong label, the estimated performance for this base-classifier will decrease (line 6). After this, base-classifier m_j can be trained with the example (line 7). When, at any time, a test example needs to be classified, the ensemble looks up the highest value of p_j and lets the corresponding ensemble member make the prediction.

The concept of an active classifier can also be extended to multiple active classifiers. Rather than selecting the best classifier on recent predictions, we can select the best k classifiers, whose votes for the specified class-label are all weighted according to some weighting schedule. First, we can weight them all equally. Indeed, when using this voting schedule and setting $k = |M|$, we would get the same behaviour as the Majority Vote Ensemble, as described by van Rijn et al. [127], which performed only averagely. Alternatively, we can use Online Performance Estimation to weight the votes. This way, the best performing classifier obtains the highest weight, the second best performing classifier a bit less, and so on. Formally, for each $y \in Y$ (with Y being the set of all class labels):

$$votes_y = \sum_{m_j \in M} P(m_j, i, \alpha, L) \cdot B(l'_j(PS_i), y) \tag{5.4}$$

where M is the set of all models, l'_j is the labelling function produced by model m_j and B is an indicator function, returning 1 iff l'_j voted for class label y and 0 otherwise. Other functions regulating the voting process can also be incorporated, but are beyond the scope of this research. The label y that obtained the highest value $votes_y$ is then predicted.

BLAST has been implemented in the MOA framework, and can be obtained in the appropriate OpenML MOA package[1].

5.3.4 Meta-Feature Ensemble

Alternatively, we can use meta-learning to dynamically select between various classifiers. Several notable techniques for heterogeneous model combination in the data stream setting rely on meta-learning [125, 126, 133]. These techniques divide the data stream in windows of size w. Over each of these windows, a set of meta-features is calculated, and this is repeated for all previously known data streams. Using these meta-features, a meta-classifier is trained to predict the best classifier for the *next* window given the meta-features of the current window. Any batch classifier can be used as the meta-classifier. Usually, Random Forest are used, since these have proven to work well in the context of algorithm selection [144].

Table 5.1 shows the meta-features per category that are used in this work. The first four categories are the commonly used 'SSIL' features. The 'Drift detection' features were first introduced in [125]. These are obtained by running a drift detector (Adwin or DDM) on the dataset, and measure the number of changes or warnings in a given interval. The motivation behind this is that for volatile streams, where many changes are detected, simple classifiers might be preferred, whereas in steady streams, more complex classifiers have the opportunity to fit a good model.

The Stream Landmarkers represent the output of Online Performance Estimation as meta-feature. After each window, Eq. 5.1 is used to give an estimation on how well each base-classifier performed on that window. These performance estimations are then used as meta-features. In fact, the Windowed version of BLAST has the same information, when the grace period is equal to the window size.

[1]Available on Maven Central: `http://search.maven.org/` (artifact id: openmlmoa)

Table 5.1: Meta-features used by the Meta-Feature Ensemble.

Category	Meta-features
Simple	# Instances, # Attributes, # Classes, Dimensionality, Default Accuracy, # Observations with Missing Values, # Missing Values, % Observations With Missing Values, % Missing Values, # Numeric Attributes, # Nominal Attributes, # Binary Attributes, % Numeric Attributes, % Nominal Attributes, % Binary Attributes, Majority Class Size, % Majority Class, Minority Class Size, % Minority Class
Statistical	Mean of Means of Numeric Attributes, Mean Standard Deviation of Numeric Attributes, Mean Kurtosis of Numeric Attributes, Mean Skewness of Numeric Attributes
Information Theoretic	Class Entropy, Mean Attribute Entropy, Mean Mutual Information, Equivalent Number Of Attributes, Noise to Signal Ratio
Landmarkers [108]	Accuracy, Kappa and Area under the ROC Curve of the following classifiers: Decision Stump, J48 (confidence factor: 0.01), k-NN, NaiveBayes, REP Tree (maximum depth: 3)
Drift detection [125]	Changes by Adwin (Hoeffding Tree), Warnings by Adwin (Hoeffding Tree), Changes by DDM (Hoeffding Tree), Warnings by DDM (Hoeffding Tree), Changes by Adwin (Naive Bayes), Warnings by Adwin (Naive Bayes), Changes by DDM (Naive Bayes), Warnings by DDM (Naive Bayes)
Stream Landmarkers	Accuracy Naive Bayes on previous window, Accuracy k-NN on previous window, ...

5.4 Experimental Setup

In order to establish the utility of Online Performance Estimation and the two proposed techniques, we conduct experiments using a large set of data streams. The data streams and results of all experiments are made publicly available in OpenML for the purposes of verifiability, reproducibility and generalizability.

5.4.1 Data Streams

The data streams are a combination of real world data streams and synthetically generated data commonly used in data stream research (e.g., [10, 13, 125]). Many contain a natural drift of concept. Opposed to batch classification, in data stream classification it is quite natural to test techniques on synthetically generated data, as real world data is hard to obtain. We estimate the performance of the methods using the prequential method: each observation is used as a test example first and as a training example afterwards [57]. In total, a set of 60 different data streams is used.

Table 5.2: Data streams used for the Data Stream experiment.

name	obs.	atts.	cls.
SEA(50)	1,000,000	4	2
SEA(50000)	1,000,000	4	2
Stagger1	1,000,000	4	2
Stagger2	1,000,000	4	2
Stagger3	1,000,000	4	2
airlines	539,383	8	2
codrnaNorm	488,565	9	2
electricity	45,312	9	2
Agrawal1	1,000,000	10	2
BNG(tic-tac-toe)	39,366	10	2
BNG(cmc)	55,296	10	3
BNG(page-blocks)	295,245	11	5
Hyperplane(10;0001)	1,000,000	11	5
Hyperplane(10;001)	1,000,000	11	5
RandomRBF(0;0)	1,000,000	11	5
RandomRBF(10;0001)	1,000,000	11	5
RandomRBF(10;001)	1,000,000	11	5
RandomRBF(50;0001)	1,000,000	11	5
RandomRBF(50;001)	1,000,000	11	5
pokerhand	829,201	11	10
BNG(solar-flare)	1,000,000	13	3
BNG(bridges_version1)	1,000,000	13	6
BNG(heart-statlog)	1,000,000	14	2
BNG(wine)	1,000,000	14	3
BNG(heart-c)	1,000,000	14	5
BNG(vowel)	1,000,000	14	11
adult	48,842	15	2
BNG(JapaneseVowels)	1,000,000	15	9
BNG(credit-a)	1,000,000	16	2
BNG(labor)	1,000,000	17	2
BNG(vote)	131,072	17	2
BNG(pendigits)	1,000,000	17	10
BNG(letter)	1,000,000	17	26
BNG(zoo)	1,000,000	18	7
BNG(lymph)	1,000,000	19	4
BNG(vehicle)	1,000,000	19	4
BNG(hepatitis)	1,000,000	20	2
BNG(segment)	1,000,000	20	7
BNG(credit-g)	1,000,000	21	2
BNG(mushroom)	1,000,000	23	2
BNG(SPECT)	1,000,000	23	2
LED(50000)	1,000,000	25	10
AirlinesCodrnaAdult	1,076,790	30	2
BNG(trains)	1,000,000	33	2
BNG(ionosphere)	1,000,000	35	2
BNG(dermatology)	1,000,000	35	6
BNG(soybean)	1,000,000	36	19
BNG(kr-vs-kp)	1,000,000	37	2
BNG(satimage)	1,000,000	37	6
BNG(anneal)	1,000,000	39	6
BNG(waveform-5000)	1,000,000	41	3
covertype	581,012	55	7
BNG(spambase)	1,000,000	58	2
BNG(sonar)	1,000,000	61	2
BNG(optdigits)	1,000,000	65	10
CovPokElec	1,455,525	73	10
BNG(mfeat-fourier)	1,000,000	77	10
vehicleNorm	98,528	101	2
20_newsgroups.drift	399,940	1,001	2
IMDB.drama	120,919	1,002	2

Table 5.2 shows all datasets used in this experiment. Some of the used data streams and data generators are described below.

SEA Concepts The SEA Concepts Generator [143] generates three numeric attributes from a given distribution, of which only the first two are relevant. The class that needs to be predicted is whether these values exceed a certain threshold. Several SEA Concept generated data streams based on different data distributions can be joined together in order to simulate concept drift.

STAGGER The STAGGER Concepts Generator [137] generates descriptions of geometrical objects. Each instance describes the size, shape and colour of such an object. A STAGGER concept is a binary classification rule distinguishing between the two classes, e.g., all blue rectangles belong to the positive class.

Rotating Hyperplanes The Rotating Hyperplane Generator [73] generates a high-dimensional hyperplane. Instances represent a point in this high-dimensional space. The task is to predict whether such a point is below the hyperplane. Concept drift can be introduced by rotating and moving the hyperplane.

LED The LED Generator [25] generates instances based on a LED display. Attributes represent the various LED lights, and the task is to predict which digit is represented. In order to add noise, attributes can display the wrong value with a certain probability. Furthermore, additional (irrelevant) attributes can be added.

Random RBF The Random RBF Generator generates a number of centroids. Each has a random position in Euclidean space, standard deviation, weight and class label. Each example is defined by its coordinates in Euclidean Space and a class label referring to a centroid close by. Centroids move at a certain speed, generating gradual concept drift.

Bayesian Network The Bayesian Network Generator [125] takes a batch data set as input, builds a Bayesian Network over it, and generates instances based on the probability tables. As the Bayesian Network does not change over time, it is unlikely that a native form of concept drift occurs in the data stream.

Composed Data Streams Common batch datasets can be appended to each other, forming one combined dataset covering all observations and attributes, containing small periods or abrupt concept drift. This is commonly done with the 'Covertype', 'Pokerhand' and 'Electricity' dataset [14, 118]. We applied a similar merge to the 'Airlines', 'CodeRNA' and 'Adult' dataset, forming 'AirlinesCodernaAdult'. The original datasets are normalized before this operation is applied.

IMDB.drama The 'IMDB' dataset contains 120,919 movie plots. Each movie is represented by a bag of words of the globally 1,000 most occurring words. Originally, it is a multi-label dataset, with binary attributes indicating whether a movie belongs to a given genre. We predict whether it is in the drama genre, which is the most frequently occurring [118].

20 **Newsgroups** The original 20 Newsgroup dataset contains 19,300 newsgroup messages, each represented as a bag of words of the 1,000 most occurring words. Each instance is part of at least one newsgroup. This data set is commonly converted into

Table 5.3: Classifiers used in the experiments. All as implemented in MOA 2016.04 by Bifet et al. [13], default parameter settings are used unless stated otherwise.

Classifier	Model type	Parameters
Naive Bayes	Bayesian	
Stochastic Gradient Descent	SVM	Loss function: Hinge
k Nearest Neighbour	Lazy	$k = 10$, $w = 1{,}000$
Hoeffding Option Tree	Option Tree	
Perceptron	Neural Network	
Random Hoeffding Tree	Decision Tree	
Rule Classifier	Decision Rules	

20 binary classification problems, with the task to determine whether an instance belongs to a given newsgroup. We append these data sets to each other, resulting in one large binary-class dataset containing $386{,}000$ records with 19 shifts in concept [118].

5.4.2 Parameter Settings

The heterogeneous ensembles contain the seven classifiers from Table 5.3 as base-classifiers. The table also states the model type, as described in Chapter 2. The classifiers are selected using the dendrogram of Figure 4.14 (page 69), loosely omitting some simple models such as No Change, Majority Class and Decision Stumps. BLAST is tested with both Fading Factors ($\alpha = 0.999$) and the Windowed approach ($w = 1{,}000$). For both versions $k = 1$, thus a single best classifier is chosen for every prediction. We explore the effect of larger values for k in Chapter 5.5.2. The Meta-Feature Ensemble also uses windows of size $1{,}000$.

5.4.3 Baselines

We compare the results of the defined methods with the *Best Single Classifier*. Each heterogeneous ensemble consists of n base-classifiers. The one that performs best on average over all data streams is considered the best single classifier. Following Figure 5.5, the Hoeffding Option Tree is the best classifier (this is also confirmed by [127]). This baseline enables us to measure the potential accuracy gains of adding more classifiers (at the cost of using more computational resources).

Furthermore, we compare against the *Majority Vote Ensemble* (same base-classifiers as in Table 5.3), which is a heterogeneous ensemble that predicts the label that is predicted by most ensemble members. This enables us to measure the potential accuracy gain of using Online Performance Estimation over just naively counting the votes of base-classifiers. Finally, we also compare the techniques to state of the art homogeneous ensembles, such as Online Bagging, Leveraging Bagging, and Accuracy Weighted Ensemble. In order to understand these a bit better, we provide some results.

Figure 5.6 shows violin plots of the performance of Accuracy Weighted Ensemble (left bars, red), Leveraging Bagging (middle bars, green) and Online Bagging (right bars, blue), with an increasing number of ensemble members. Accuracy Weighted Ensebmble (AWE) uses J48 trees as ensemble members, both Bagging schemas use Hoeffding Trees. Naturally, as the number of members increases, both accuracy and run time increase, however Leveraging Bagging performs eminently better than the others. Leveraging Bagging using 16 ensemble members already outperforms both AWE and Online Bagging using 128 ensemble members, based on median accuracy. This performance also comes at a cost, as it uses considerably more run time than both other techniques, even when containing the same number of members. Accuracy Weighted Ensemble performs pretty constant, regardless of the number of ensemble members. As the ensemble size grows, both accuracy and run time slightly increase. We will compare BLAST against the homogeneous ensembles consisting of 128 members.

5.5 Results

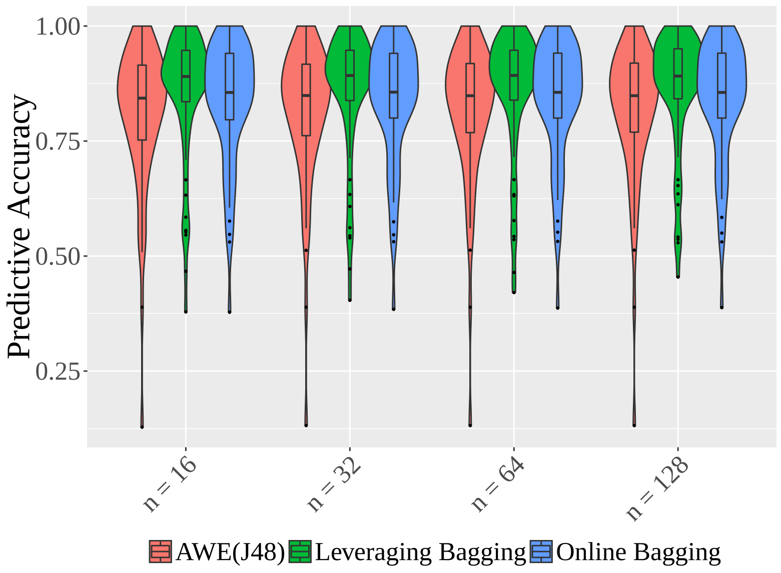
In this chapter we present and discuss the experimental results.

5.5.1 Ensemble Performance

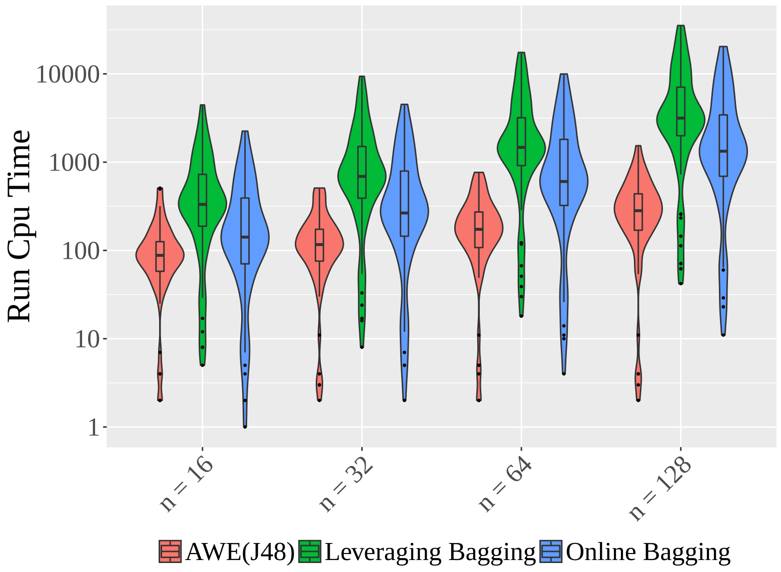
We run all techniques on the set of 60 data streams. The results are shown in Figure 5.7. As the Meta-Feature ensemble is a virtual classifier, there are no run time results available. However, as it builds the same set of base-classifiers as the Majority Vote Ensemble and both versions of BLAST, it is plausible that the run time is in the same order of magnitude.

An important observation is that both versions of BLAST are competitive with state of the art ensembles. These results suggest that Online Performance Estimation is a useful technique when applied to data stream ensembles. Also note that BLAST requires far fewer run time than the ensemble techniques. A peculiar observation is that BLAST also outperforms the Meta-Feature Ensemble, which has access to the same information and thus is supposed to perform at least equally good.

Figure 5.8 shows the accuracy of four heterogeneous ensemble techniques per data stream. In order to not overload the figure, we only show BLAST (with fading

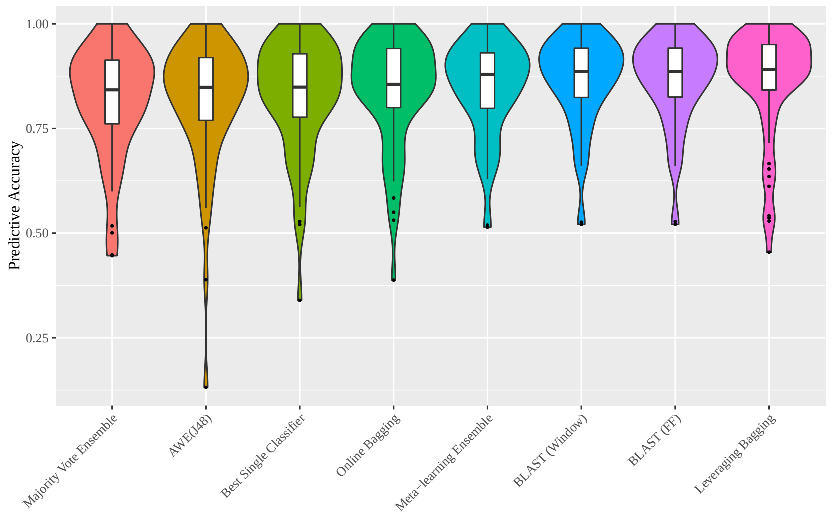


(a) Accuracy

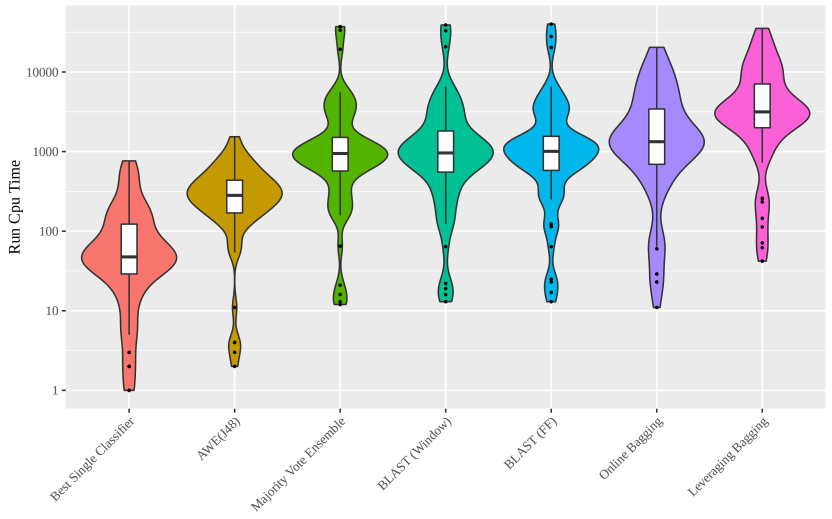


(b) Run time

Figure 5.6: Effect of the ensemble size parameter.



(a) Accuracy



(b) Run time

Figure 5.7: Performance of the various meta-learning techniques averaged over 60 data streams.

factors), the Meta-Feature Ensemble, Leveraging Bagging and the Best Single Classifier.

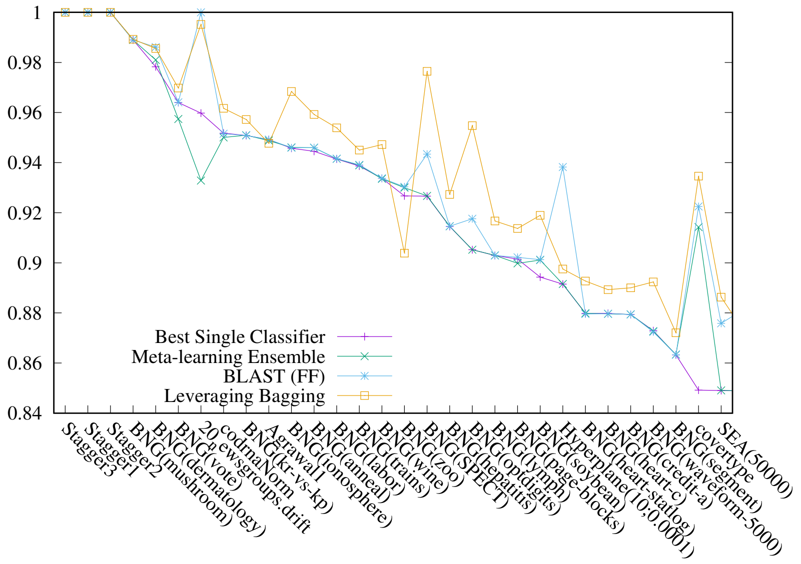
All techniques outperform the Best Single Classifier consistently. Especially on data streams where the performance of the Best Single Classifier is mediocre (Figure 5.8 bottom part), accuracy gains are eminent. The difference between Leveraging Bagging and BLAST is harder to assess. Although Leveraging Bagging seems to be slightly better in many cases, there are some clear cases where there is a big difference in favour of BLAST (see, e.g., the difference in data stream 'IMDB.drama'). Despite Leveraging Bagging is best on most data streams, other techniques are better on average.

To assess statistical significance of the results, we use the Friedman test with post-hoc Nemenyi test to establish the statistical relevance of our results. These tests are considered the state of the art when it comes to comparing multiple classifiers [36]. The Friedman test checks whether there is a statistical significant difference between the classifiers; when this is the case the Nemenyi post-hoc test can be used to determine which classifiers are significantly better than others.

The results of the Nemenyi test are shown in Figure 5.9. It plots the average rank of all methods and the critical difference. Classifiers for which not a statistically significant difference was found are connected by a black line. For all other cases, there was a significant difference in performance, in favour of the classifier with the better (lower) average rank. We performed the test based on Accuracy and run time. Again, no run time results are recorded for the Meta-Feature Ensemble.

Figure 5.9a shows that there is no statistically significant difference in terms of accuracy between BLAST (using Fading Factors) and two of the homogeneous ensembles (i.e., Leveraging Bagging and Online Bagging using 128 Hoeffding Trees). BLAST (Window) does perform significantly worse than Leveraging Bagging. Meta-Feature Ensemble is significantly worse than Leveraging Bagging and BLAST (with Fading Factors). As expected, the Best Single Classifier and the Majority Vote Ensemble perform significantly worse than most other techniques. Clearly, combining heterogeneous ensemble members by simply counting votes does not work in this setup. It seems that poor results from some ensemble members outweigh the diversity.

When comparing the techniques on computational efficiency, the overall outcome is different, as reflected by Figure 5.9b. The best single classifier (Hoeffding Option Tree) requires fewest resources. There is no significant difference in resources between BLAST (FF), BLAST (Window), Majority Vote Ensemble and Online Bagging. This makes sense, as these the first three operate on the same set of base-classifiers. The Accuracy Weighted Ensemble performs really well in terms of run time, even though it uses 128 ensemble members. Due to its efficient trainings technique, adding more classifiers does not necessarily result in more run time. Altogether, BLAST (FF) performs equivalent to both Bagging schemas in terms of accuracy, while using signi-



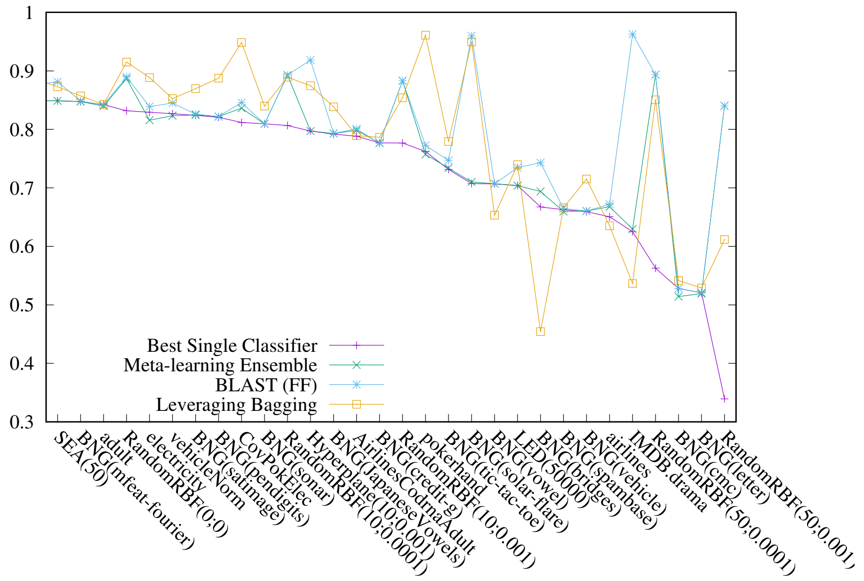
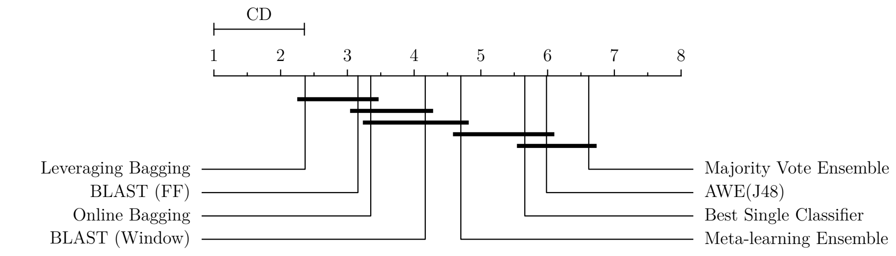
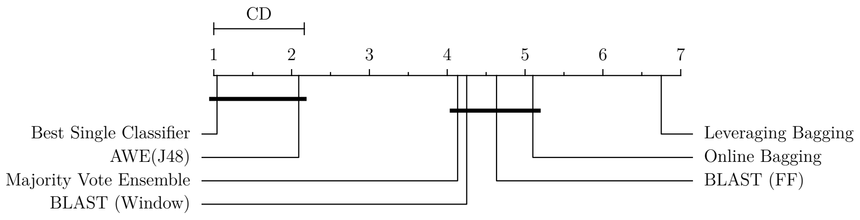


Figure 5.8: Accuracy per data stream, sorted by accuracy of the best single classifier.



(a) Accuracy



(b) Run time

Figure 5.9: Results of Nemenyi test. Classifiers are sorted by their average rank (lower is better). Classifiers that are connected by a horizontal line are statistically equivalent.

ficantly fewer resources.

5.5.2 Effect of Parameters

In this chapter we study the effect of the user-defined parameters of BLAST.

5.5.2.1 Window size and decay rate

First, for both version of BLAST, there is a parameter that controls the rate of dismissal of old observations. For BLAST (FF) this is the α parameter (the fading factor); for BLAST (Windowed) this is the w parameter (the window size). The α parameter is always in the range $[0, 1]$, and has no effect on the use of resources. The window parameter can be in the range $[0, n]$, where n is the size of the data stream. Setting this value higher results in bigger memory requirements, although these are typically neglectable compared to the memory usage of the base-classifiers.

Figure 5.10 shows violin plots and box plots of the effect of varying these parameters. The effect of the α (a) value on BLAST (FF) is displayed in the left (red)

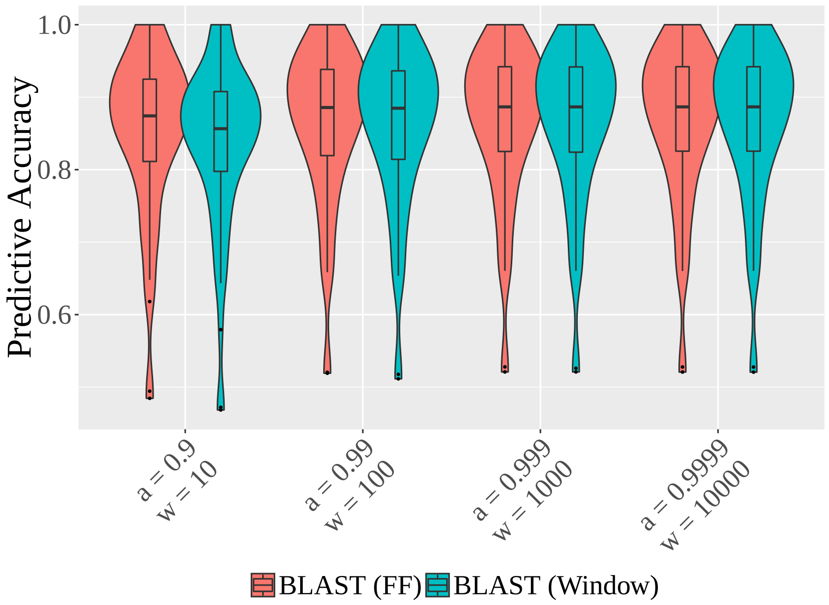


Figure 5.10: Effect of the decay rate and window parameter.

violin plots; the effect of the window (w) value on BLAST (Window) is displayed in the right (blue) violin plots. Even though it is hard to draw general conclusions from this, the trend over 60 data streams seems to be that setting this parameter too low results in suboptimal accuracy. Arguably, this is good in highly volatile streams when concepts change rapidly, but in general we do not want to dismiss old information too quickly. Altogether, BLAST (FF) seems to be slightly more robust, as the investigated values of the α parameter does not perceptibly influence the performance.

5.5.2.2 Grace parameter

Prior work by Rossi et al. [133] and van Rijn et al. [127] introduced a grace parameter that controls the number of observations for which the active classifier was not changed. This potentially saves resources, especially when the algorithm selection process involves time consuming operations such as the calculation of meta-features. On the other hand, it can be seen that in a data stream setting where concept drift occurs, in terms of performance it is always optimal to act on changed behaviour as fast as possible. Although we have omitted this parameter from the formal definition of BLAST in Chapter 5.3, similar behaviour can be obtained by updating the active

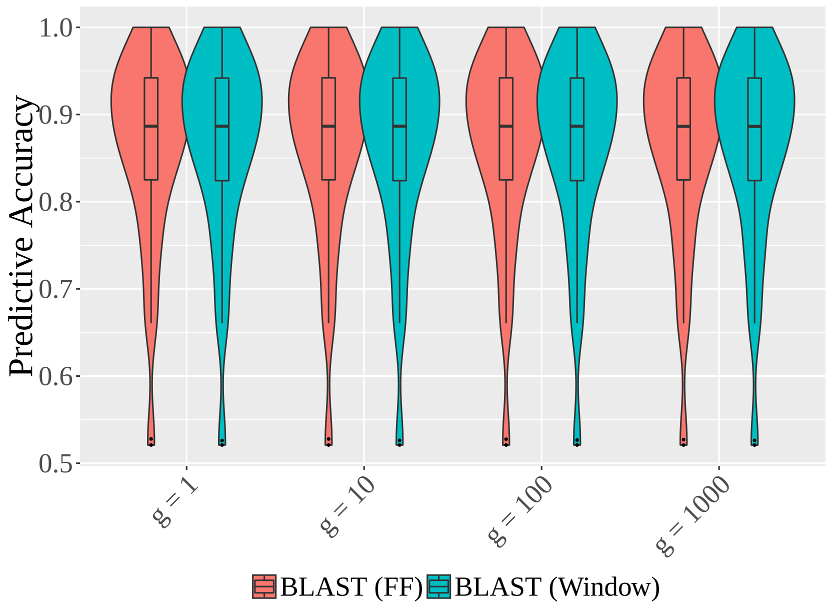


Figure 5.11: Effect of the grace parameter on accuracy. The x-axis denotes the grace, the y-axes the performance. BLAST (Window) was ran with $w = 1{,}000$; BLAST (FF) was ran with $\alpha = 0.999$.

classifier only at certain invervals. Formally, a grace period can be enforced by only executing Eq. 5.3 when $c \mod g = 0$, where (following earlier definitions) c is the index of the current observation, and g is a grace period defined by the user.

Figure 5.11 shows the effect of the (hypothetical) grace parameter on performance, averaged over 60 data streams. The plots do not indicate that there is much difference in performance. Moreover, the algorithm selection phase of BLAST simply depends on finding the maximum element in an array. Therefore, the grace period would not have any influence on the required resources.

5.5.2.3 Number of active classifiers

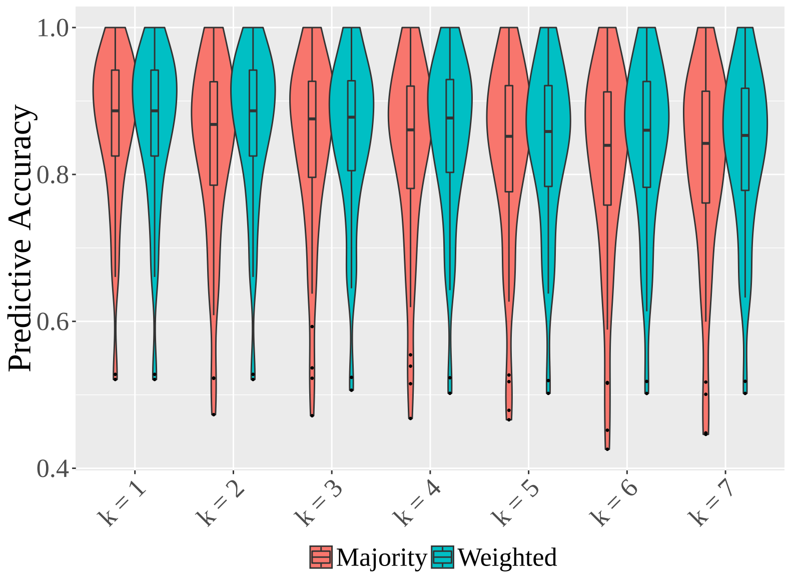
Rather than selecting one active classifier, multiple active classifiers can be selected that all vote for a class label. The votes of these classifiers can either contribute equally to the final vote, or be weighted accouring to their estimated performance. We used BLAST (FF) to explore the effect of this parameter. We varied the number of active classifiers k from one to five, and measured the performace according to both voting schemas. Figure 5.12 shows the results.

Figure 5.12a shows how the various strategies perform when evaluated using predictive accuracy. We can make several observations to verify the correctness of the results. First, the results of both strategies are equal when $k = 1$, as the algorithm selects only one classifier, weights are obsolete. Second, the result of the Majority weighting schema for $k = 7$ is equal to the score of the Majority Weight Ensemble (see Figure 5.7a), which is also correct, as these are the same by definition. Finally, when using the weighted strategy, setting $k = 2$ yields exactly the same scores for accuracy as setting $k = 1$. This also makes sense, as it is guaranteed that the second best base-classifier always has a lower weight as the best base-classifier, and thus it is incapable of changing any prediction.

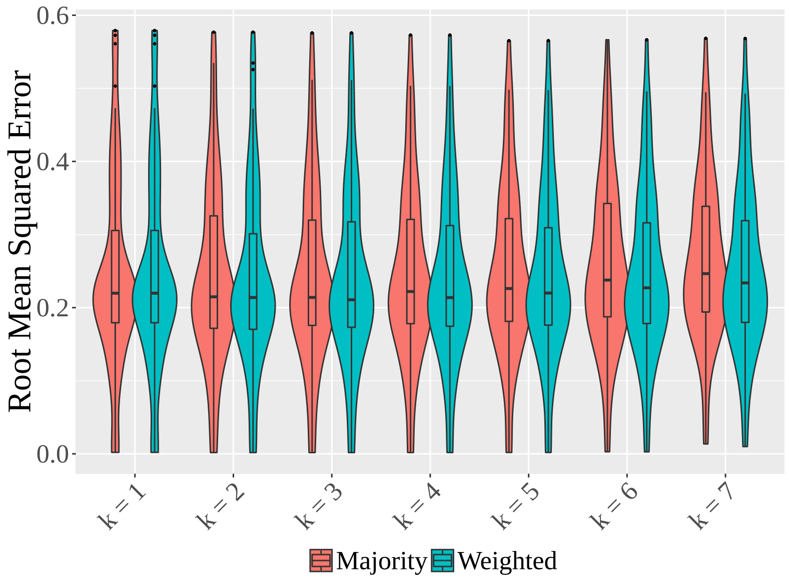
In all, it seems that increasing the number of active classifiers is not beneficial for accuracy. Note that this is different from adding more classifiers in general, which clearly would not decrease performance results. This behaviour is different from the classical approach, where adding more classifiers (which inherently are all active) yield better results up to a certain point [30]. However, in the data stream setting we deal with a time component, and we can actually measure which classifiers performed well on recent intervals. By increasing the number of active classifiers, we would add classifiers to the vote of which we have empirical evidence that they performed worse on recent observations.

Similarly, Figure 5.12b shows the Root Mean Squared Error (RMSE). RSME is typically used as an evaluation measure for regression, but can be used in classification problems to assess the quality of class confidences. For every prediction, the error is considered to be the difference between the class confidence for the correct label and 1. This means that if the classifier had a confidence close to 1 for the particular class label, a low error is recorded, and vice versa. The violin plots indicate that adding more active classifiers can lead to a lower median error. This also makes sense, as the Root Mean Squared Error tends to punish classification errors harder when these are made with a high confidence. We observe this effect until $k = 3$, after which adding more active classifiers starts to lead to a higher RMSE. It is unclear why this effect holds until this particular value.

All together, from this experiment we conclude that adding more active classifiers in the context of Online Performance Estimation does not necessarily yields better results beyond selecting the expected best classifier at that point in the stream. This might be different when using more base-classifiers, as we would expect to have more similarly performing classifiers on each interval. As we expect to measure this effect when using orders of magnitude more classifiers, this is considered future work. Clearly, when using multiple active classifiers, weighting their votes using online performance estimation seems beneficial.

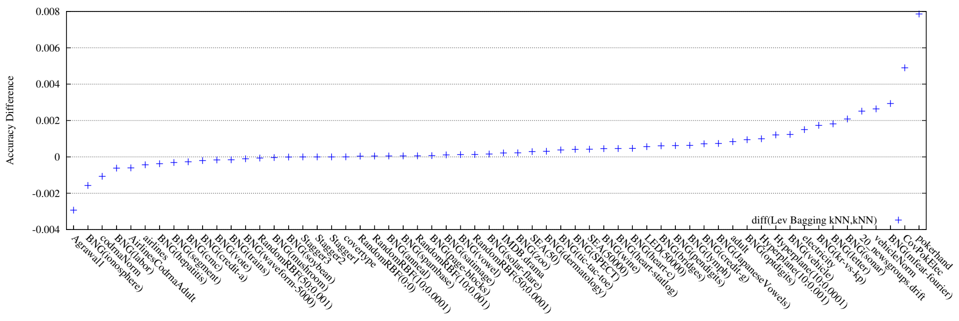


(a) Accuracy

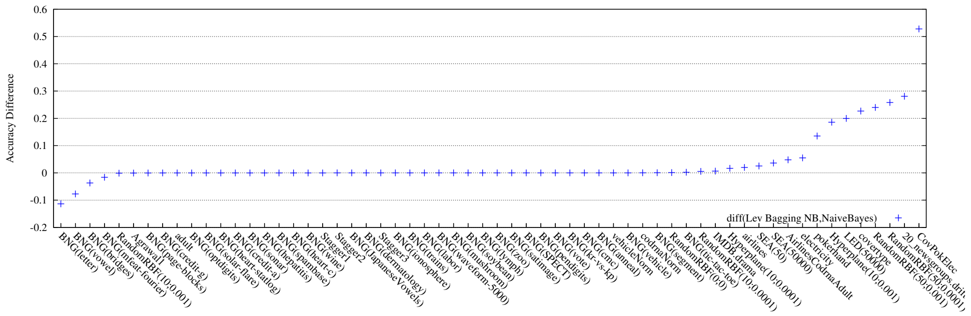


(b) Root Mean Squared Error

Figure 5.12: Performance for various values of k, weighted votes versus unweighted votes.



(a) Leveraged Bagging k-NN Vs. k-NN



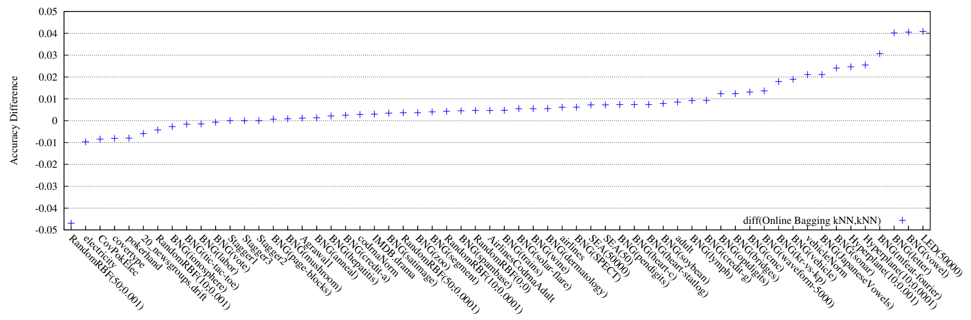
(b) Leveraged Bagging Naive Bayes Vs. Naive Bayes

Figure 5.13: Performance differences between Leveraging Bagging ensembles and single classifiers.

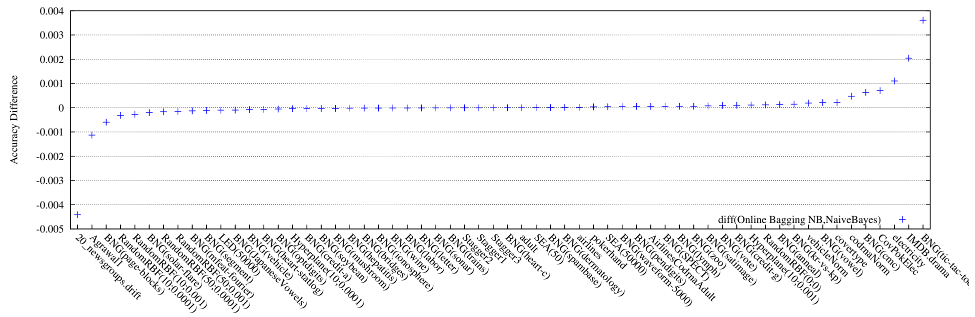
5.6 Designed Serendipity

The philosophy of openly sharing results and working on massive amounts of data streams leads to unexpected results that would possibly remain undiscovered otherwise. In prior work was already observed that applying a Bagging technique like Leveraged Bagging improves classification performance for k Nearest Neighbour [125]. This is a counter-intuitive result, as Breiman [23] already conjectured that Bagging will not affect stable classifiers in the batch setting; k Nearest Neighbour with $k = 10$ is considered a stable classifier. A follow-up study has been conducted for the data stream setting, covering both k Nearest Neighbour and Naive Bayes, and comparing the performance over two Bagging Schema's, i.e., Online Bagging and Leveraging Bagging [129].

Figure 5.13 and Figure 5.14 show the results of the experiments. The x-axis shows



(a) Online Bagging k-NN Vs. k-NN



(b) Online Bagging Naive Bayes Vs. Naive Bayes

Figure 5.14: Performance differences between Online Bagging ensembles and single classifiers.

the dataset, the y-axis shows the difference in performance between the bagging schema and the single classifier. Note that the single classifier performed better on datasets where the difference is below zero, and vice versa for datasets where the difference is above zero. Datasets are ordered by this difference.

Figure 5.13a shows a similar trend as seen in the figure presented in [125]. There are few data streams on which k Nearest Neighbour performs best, but many data streams on which Leveraging Bagging k Nearest Neighbour performs best. One notable observation is the difference in scale between Figure 5.13b describing the effect of using Naive Bayes in a Leveraging Bagging schema, and the other figures. In most cases, the effect of a Bagging schema can attribute to performance gains of few percentages. However in the case of Leveraged Bagging Naive Bayes, the performance gain can lead up to 50% in the 'CovPokElec' dataset, but also other data sets show eminent improvements.

Among the 15 data streams on which Leveraged Bagging improves upon Naive Bayes the most, many presumably contain concept drift. Compared to k Nearest Neighbour, Naive Bayes' performance is quite poor on these data streams[2]. Apparently, k Nearest Neighbour's natural protection against concept drift (it removes old instances as new ones come in) makes it perform quite well. Note that Leveraging Bagging is a Bagging technique combined with a change detector and leveraging more randomness. When using Naive Bayes in a Leveraging Bagging schema, the change detector ensures that Naive Bayes also obtains this performance increase.

Figure 5.14b shows what we would expect to see when applying Online Bagging, which is a pure form of Bagging described in Chapter 3, to a stable classifier. The differences in accuracy are small and the performance gains are equally divided between the single classifier and the bagging schema.

From this we learn three things.

- First, when applied in a normal Bagging schema, Naive Bayes classifiers exhibit similar behaviour as in the batch setting (Figure 5.14b).
- Second, this result does not hold for Leveraging Bagging, which also embodies a change detector. This can have a big influence on classification results (Figure 5.13b).
- Finally, concept drift has a big influence on the applicability of bagging schemas.

5.7 Conclusions

This chapter covered Data Streams and various Machine Learning techniques that operate on them. We surveyed the Online Performance Estimation framework, which can be used in data stream ensembles to weight the votes of ensemble members, in particular when using fundamentally different model types. Online Performance Estimation measures the performance of all base-classifiers on recent training examples. Using two different performance estimation functions, it can be used to built heterogeneous ensembles.

BLAST is a heterogeneous ensemble technique based on Online Performance Estimation that selects the single best classifier on recent predictions to classify new observations. We have integrated both performance estimation functions into BLAST. The Meta-Feature Ensemble uses a variety of traditional meta-features and meta-features obtained from the Online Performance Estimation framework. The introduced techniques were developed using experimental results stored in OpenML. Em-

[2]This information can not be deduced from the figures, therefore the reader is referred to OpenML or Table 3 of [129]

pirical evaluation shows that BLAST with fading factors outperforms the other methods. This is most likely because Fading Factors are better able to capture typical data stream properties, such as changes of concepts. When this occurs, there will also be changes in the performances of ensemble members, and the fading factors adapt to this relatively fast. Based on an empirical evaluation covering 60 data streams, we observe that BLAST is statistically equivalent to current state of the art ensembles while using significantly fewer resources.

We also evaluated the effect of the method's parameters on the performance. The most important parameter proves to be the one controlling the performance estimation function: α for the fading factor, controlling the decay rate, and w for the windowed approach, determining the window size. Our results show that the optimal value for these parameters is dependent on the given dataset, although setting this too low turns out to have a worse effect on accuracy than setting it too high.

To select the classifiers included in the heterogeneous ensemble, we used the hierarchical clustering of 25 commonly used data stream classifiers that was presented in Figure 4.14 (page 69). We used this clustering to gain methodological justification for which classifiers to use, although the clustering is mainly a guideline. A human expert can still determine to deviate from the resulting set of algorithms, in order to save resources. The resulting dendrogram also has scientific value in itself. It confirms some well-established assumptions regarding the typically used classifier taxonomy in data streams, that have never been tested before. Many of the classifiers that were suspected to be similar were also clustered together, for example the various decision trees, support vector machines and gradient descent models all formed their own clusters. Moreover, some interesting observations were made that can be investigated in future work. For instance, the Rule Classifier used turns out to perform averagely, and was rather far removed from the decision trees, whereas we would expect it to perform better and be clustered closer to the decision trees.

Utilizing the Online Performance Estimation framework opens up a whole new line of data stream research. Rather than creating more data stream classifiers, combining them in a suitable way can elegantly lead to highly improved results that effortlessly adapt to changes in the data stream. More than in the classical batch setting, memory and time are of crucial importance. Experiments suggest that the selected set of base-classifiers has a substantial influence on the performance of the ensemble. Research should be conducted to explore what model types best complement each other, and which work well together given a constraint on resources. Combining data stream research with the open approach of OpenML led to new knowledge and techniques. We believe that by exploring these possibilities we can further push the state of the art in data stream ensembles.