Radboud University Nijmegen

Master Thesis

in Artificial Intelligence

Keeping up with Fraud

An Active Learning Approach for Imbalanced Non-Stationary Data Streams

---

Thesis submitted by:
Diede Kemper
s4056566

Supervisors:
Dr. George Kachergis*
Dr. habil. Georg Krempl**
Marcel de Wit***

July 10, 2017

* first reader, Radboud University Nijmegen
** second reader, Otto von Guericke Universität Magdeburg
*** Sogeti Netherlands BV

Abstract

Fraud detection is a difficult task in which multiple problems co-occur. The data often comes from a non-stationary stream. Moreover, correct labels are available for only a small part of the data and fraudulent cases are much more rare than non-fraudulent cases. A promising technique for solving this combination of problems is *active learning*, where instances are selected for labeling such that the classifier can learn the most. Previously, the critical sampling strategy has been proposed, that selects instances close to the decision boundary and oversamples fraudulent cases. The current project suggested an extension to this strategy that also explores full input space. These strategies were compared to state-of-the-art active learning strategies, using a new data stream sampled from the KDD'99 dataset, implemented in Massive Online Analysis (MOA). It was found that the original critical sampling algorithm does not perform better than random sampling, as has been found previously. An explanation could be that critical sampling induces a sampling bias, specifically if minority data comes from multiple dense and sparse areas in input space. In further research, this sampling bias could be overcome by combining critical sampling with a clustering- or diversity-based approach.

1 Introduction

New technologies such as mobile phones, the internet-of-things and networks of sensors have led to an increasing amount of data. Often, data are continuously recorded and models should be able to deal with these streams of data. The objective of the predictive model is to learn the distribution that the data are drawn from. In the field of data mining, it is often assumed that this distribution is stationary. In most real-world scenarios, however, this assumption is not true [10]. Data streams are often non-stationary and the distribution from which the data are drawn changes over time. This can be caused by, for example, changes in preferences of users, aging of sensors, and seasonal or periodic effects. This non-stationarity in data streams is called *concept drift*.

Applications in which non-stationary data streams are used are becoming more prevalent. Application areas can be divided into three categories [38]: (a) Information management, (b) analytics and diagnostics, and (c) monitoring and control. Information management involves personalization, for example via recommender systems or by customer profiling. Analytics and diagnostics include predictive analytics and diagnostic tasks, for example crime rate prediction or demand prediction. Monitoring and control mostly relates to detection tasks, where abnormal behavior needs to be signaled. Examples are bankruptcy prediction, controlling output quality in production industries and fraud detection.

Some of these applications have more problems to deal with than only concept drift. For example, it can be too costly to acquire labels for all data points. This gives rise to the partial labeling problem: How can predictive models deal with a data stream where only a part of the data points is labeled? Furthermore, class imbalance refers to the problem where some classes occur more

frequently than others, making it difficult for models to correctly classify the minority class. Fraud detection is an example where all three of these problems occur. Concept drift arises because of changing regulations or discovery of new fraudulent methods. Partial labeling occurs because there is not enough time and money to check every case for fraudulent behavior. Last, class imbalance arises because fraudulent cases are much more rare than non-fraudulent cases.

In the scientific community, the combination of these three problems has not received much attention. Nonetheless, it is important that techniques are developed that can deal with these problems, in order to better deal with fraud detection and similar real-world problems. Indeed, more research into this field has been called for [20]. This study contributes to this end, by studying the performance of several existing and one new active learning strategy on a drifting, partially labeled and imbalanced data stream. Moreover, the data stream generator that was developed during this study will be made available online to promote more research into this type of data streams. Before going into more details about the current study, first the existing literature on concept drift, class imbalance and partial labeling will be discussed.

1.1 Concept Drift

1.1.1 Characterizing concept drift

The data-generating process of a data stream provides a sequence of tuples (X_t, y_t) sampled from an unknown probability distribution $p_t(X, y) = p_t(y|X)p_t(X)$, where $p_t(y|X)$ and $p_t(X)$ denote the posterior and evidence distributions, respectively, at some arbitrary time point t [10]. In non-stationary data streams, we are trying to predict a concept that changes over time. This change is commonly referred to as *concept drift*. Formally, concept drift between time points t_0 and t_1 can be defined as

$$\exists X : p_{t_0}(X, y) \neq p_{t_1}(X, y)$$

where p_{t_n} denotes the joint distribution at time t_n between the set of input variables X and the target variable y [14]. This change in the joint distribution can have two causes, giving rise to two types of concept drift:

1. Real concept drift. The posterior probability $p_t(y|X)$ changes over time.
2. Virtual drift. The evidence $p_t(X)$ changes over time, without affecting $p_t(y|X)$.

Note that virtual drift can be detected even from looking solely at the input data distribution [14]. In other words, the true labels y are not needed to make predictive models adaptive to virtual drift. However, in order to deal with real concept drift, the true labels y are necessary such that the classifier can shift its decision boundary. It is expected that, in the case of fraud detection, the

decision boundary between 'fraud' and 'no fraud' changes over time [1]. Therefore, in this study the focus will be on dealing with real concept drift.

Concept drift is often characterized by its speed. Ditzler et al. [10] discriminate between sudden and gradual concept drift. Sudden concept drifts refer to abrupt changes in data patterns, caused by for example faults affecting a sensor. Gradual concept drifts refer to slowly evolving distributions over time, caused by for example aging effects of sensors. As noted by Gama et al. [14], two types of gradual drifts exist. There is drift where the concept itself gradually changes. In other words, there are intermediate concepts. For example, when sensors age, the concept halfway through the drift is different from both the initial concept and the final concept. Moreover, there is drift from one concept to another where the probability of observing the concepts gradually changes over time. For example, a reader may change his or her interest in news topics, but initially keeps going back to the previous interest for some time. In this study, we are interested in sudden drift and both types of gradual drift[2].

In the next section, existing techniques that can deal with concept drift are described.

1.1.2 Dealing with concept drift

There are two types of methods that can deal with concept drift [10]: active and passive methods. Active methods monitor whether a change in the data patterns occurs. The model is adapted only if such a change is detected. Passive methods do not actively monitor changes, but nevertheless continuously update the model. On the one hand, active methods are better at coping with sudden drifts than passive methods. Also, active methods perform better in an instance-incremental setting, where instances are processed one at a time. On the other hand, passive methods are better at coping with gradual and reoccurring drifts than active methods. Furthermore, passive methods work better in a batch-incremental setting, where processing happens per set of instances.

Active methods use change detectors to monitor the data stream [10]. After a change has been detected, the learner needs to adapt to the change by learning from the up-to-date samples and discarding obsolete ones. The challenge is to distinguish both types of samples from one another. Three main types of adaptation mechanisms are used. In the windowing mechanism, a sliding window is used to select the most recent samples, while all older samples are discarded, and the learner is retrained on the selected samples. The optimal length of the window can be estimated using a variety of methods. Second, when using the method of weighting, all available samples are kept in the buffer, but samples are given a weight that represents how relevant it is to the new (changed) concept.

[1] For instance, because of changing laws, things that were allowed previously can become fraudulent later, and vice versa. Also, fraudsters will try to hide their fraud and make it seem like normal observations. Across time, it can be expected that fraudsters get better in this. This results in observations that were not suspicious previously but are suspicious now.

[2] But, as will be noted, in our experiments we are only capable of simulating the gradual drift of the probability of observing the old and new concepts.

The main drawback of this approach is that many samples need to be stored which does not meet the minimal memory requirements of most data stream applications. Third, in the sampling approach, samples have a certain probability p to be stored in the reservoir and a probability $(1 - p)$ to be discarded. The formula that is used to compute this probability p takes into account the recency and quality of the sample. A new model is then trained on the samples stored in the reservoir.

Passive methods perform a continuous adaptation of model parameters every time new data arrive. In this way, these methods do not have the potential pitfall of failing to detect a change or falsely detecting a non-existent change. There are two main approaches of passive methods, those that use a single classifier and those that use an ensemble of classifiers. Single classifiers have a lower computational cost than ensemble-based approaches, making them attractive solutions for massive data streams. However, it has been shown that ensembles of classifiers can provide more stable results than single classifiers in non-stationary settings [10].

One of the most popular single classifiers for data stream mining is the Very-Fast Decision Tree (VFDT) or Hoeffding Tree [11]. This classifier builds a decision tree while going through the data stream. A separate part of the data stream is used to decide the attribute to split on for each node. The first data points will be used to choose the attribute for the root node, the subsequent data points will be passed down to the corresponding leaves and are used to choose the splitting attributes there, etcetera. The Concept-adapting VFDT (CVFDT) [17] can also deal with concept drifts. It uses a sliding window of data to keep up with the most recent concept. Periodically, it is verified whether each node of the tree would be split on the same attribute based on the current window of data. If not, a parallel subtree is built for the node. Once this alternative subtree has an higher accuracy than the original subtree, it replaces the original. This way, the decision tree is adapted along the way and can deal with drifts.

Ensemble methods can handle concept drift very well [21]. An ensemble is a group of classifiers where the prediction of the ensemble is based on the predictions of its members. There are three main approaches to deal with concept drift in an ensemble [24]:

1. Dynamic combinations of members. The changes in the environment are tracked by changing the combination rule of the members. Most often, classifiers are combined by giving them weights based on their individual performance. The prediction of the ensemble is a weighted sum of the predictions by the individual classifiers.

2. Continuous updates. The classifiers can be updated incrementally based on the new training examples.

3. Structural updates. The structure of the ensemble can be changed by adding and/or removing classifiers. Often, either the oldest or the worst-

performing member is replaced by a classifier trained on a new part of the data.

Some ensemble methods work in an instance-incremental setting. For example, in Online Bagging, an ensemble of multiple classifiers is built, where each classifier is built on a different set of input instances [28]. This is realized by training each classifier K times on the current input instance, where K is drawn from a Poisson distribution with parameter $\lambda = 1$. This means that there is a high probability that the classifier uses the current input instance 0 or 1 time, but also quite some probability that it is used more often.

Most ensemble methods, however, work in a batch-incremental setting. For instance, the Streaming Ensemble Algorithm (SEA) [30] was one of the first streaming ensemble methods. It trains a new classifier for each incoming batch of data. The ensemble has a fixed length. When the ensemble is full, the classifier that has the lowest quality value is replaced by the new one. This quality value can be decided using an heuristic, such as the age of the classifier, or using a more direct measurement such as the accuracy of predictions [10]. A more recent example of a streaming ensemble is Learn++.NSE [12], that also trains a new classifier for each batch of data. The predictions of the ensemble members are combined by using weighted majority voting. The weights of ensemble members are determined based on the accuracy of the members. Members are never removed from the ensemble, but their weights can become zero.

In the next section, methods that can deal with class imbalance in evolving data streams are discussed.

1.2 Dealing with Class Imbalance

Classic machine learning algorithms often assume that training data contain roughly the same number of instances from each class. However, in practice this assumption is not always met. Class imbalance refers to the situation where some classes appear much more frequent than others. This results in learning algorithms that are biased towards correctly classifying the majority classes and misclassifying minority classes. In real-life settings, this behavior of models can be destructive, especially since the cost of misclassifying minority classes is often higher than the cost of misclassifying majority classes. Therefore it is important that approaches are developed that can deal with class imbalance.

In static data mining, this problem has been studied in much detail. There exist three general approaches to deal with imbalanced data [20]:

1. Data-level methods use normal machine learning algorithms, but change the class distribution of the data beforehand. In practice, this often results in oversampling of the minority class and undersampling of the majority class instances.

2. Algorithm-level methods actually change machine learning algorithms such that they can deal with imbalanced data. Cost-sensitive algorithms are

often used, where misclassifying a minority instance has an higher cost than misclassifying a majority instance.

3. Hybrid methods combine data-level and algorithm-level methods.

Only in recent years, class imbalance in data streams has been explored [20]. Often, data-level methods are used that change the class distribution of input data per batch. For example, one approach oversamples minority instances and undersamples majority instances per batch, where misclassified majority instances have a higher probability to remain than correctly classified instances [25]. Moreover, the algorithm of Uncorrelated Bagging (UCB) trains classifiers on a subset of the majority class instances and all minority class instances seen so far [15]. This approach implicitly assumes that the minority class does not experience drift, which is not necessarily true. This problem has been solved in the Selective Recursive Approach (SERA) [8]. In this approach, only the previous minority examples that are most similar to recent minority examples are selected. Furthermore, in another approach [9], Learn++.NSE has been combined with SMOTE [7]. SMOTE stands for Synthetic Minority Oversampling Technique. As the name implies, it adds synthetically created minority instances to the dataset. SMOTE is applied to each batch of data before Learn++.NSE is applied to classify the data and to train its ensemble members.

Instance-incremental methods have also been proposed. For example, a cost-sensitive neural network classifier has been proposed [16]. Furthermore, two variants to Online Bagging have been developed that can deal with class imbalance [32]: Oversampling Online Bagging (OOB) and Undersampling Online Bagging (UOB). In these variants, the parameter λ of the Poisson distribution is dynamically determined by the degree of class imbalance and by the class of the instance. In OOB, this results in oversampling of the minority class, whereas in UOB, this results in undersampling of the majority class. An ensemble combining both OOB and UOB has also been proposed [32].

All methods discussed in the previous paragraphs assume that it is known whether an instance is a minority instance. In real-world applications, however, this is often not the case. Therefore, an open challenge in the field of class imbalance in evolving data streams is its combination with the problem of partial labeling [20]. This challenge will be explored in the current study. In the next section, literature that deals with the partial labeling problem is discussed, with a focus on active learning.

1.3 Dealing with Partial Labeling: Active Learning

In data stream literature, it is often assumed that all data are labeled. In reality, however, this assumption does not always hold true. Often, only a fraction of data can be labeled. Active learning deals with the variant of this problem where the learner itself can choose which instances should be labeled. This has the advantage that the learner can choose those instances from which it expects to learn the most.

Multiple active learning methods have been developed. Two families of active learning methods are very popular [22]. First, in uncertainty sampling the instances that are close to the decision boundary of the classifier are selected for labeling. These are the instances with the lowest maximum posterior estimates of the classifier and thus the classifier is most uncertain about their predicted labels. The idea is that classifiers can learn the most from these instances. Uncertainty sampling has two shortcomings [22]. First, it does not always take into account the uncertainty of the posterior estimate itself. This causes unreliable uncertainty measures at the start of classifier training. At later stages, uncertainty measures tend to sample from regions with high Bayesian error, where classification accuracy is not improvable. These shortcomings cause uncertainty sampling to perform worse than random sampling quite often. It is nevertheless one of the most used active learning baselines because of its efficiency and simplicity. The second family of methods is query by committee [22]. This family is used by ensemble classifiers. All ensemble members predict the class label of input instances. The instance with the highest disagreement in its predicted label is selected for labeling.

The application of active learning to evolving data streams has only recently been explored. Two types of methods have been proposed, instance-incremental and batch-incremental methods. These will now be discussed.

1.3.1 Instance-incremental

Conventional active learning strategies are often certainty-based, where only the uncertain instances close to the decision boundary are labeled. For instance-incremental active learning this is challenging, because the decision to label an instance or to refrain from labeling has to be made immediately when the instance comes available. Setting a fixed certainty threshold for labeling is problematic, because either too many instances are selected from the start, leaving little labeling budget for the other instances, or the classifier becomes more certain across time causing no instances to be labeled anymore. Zliobaite et al. [36] [37] have proposed the variable uncertainty strategy to overcome this problem. This strategy increases the certainty threshold when the classifier is certain about its judgments and decreases it when the classifier is uncertain. It cannot detect all concept drifts in the data stream, however, as it only selects instances close to the decision boundary. Concept drifts can take place everywhere in input space. Therefore, the authors proposed the variable uncertainty strategy with randomization, where the certainty threshold is randomized by multiplication with a normally distributed random variable. This way, the strategy does cover the full input space.

Several improvements of these instance-incremental strategies have been proposed. For example, DBALStream [19] is an extension to the variable uncertainty strategy, with two main changes. First, an instance can only be labeled if it is present in a dense area. This change is built on the assumption that an instance is more important if it lies in a dense area. Second, instead of the maximum posterior, the margin is used to measure classifier uncertainty. The margin

is the difference in probability between the two most probable classes. DBAL-Stream is shown to outperform random selection and the variable uncertainty strategy with randomization. The Active and Adaptive Incremental Learning method (AAIL) has been proposed as another competing algorithm [29]. For each incoming instance, the probability that concept drift happens is estimated by testing how much the instance deviates from the current probability distribution in the data. The probability that an instance is selected for labeling depends on the amount of deviation: When the instance does not deviate much, it is not likely to be selected, whereas a deviating instance is very likely to be selected. It is shown that AAIL performs better than variable uncertainty strategy.

1.3.2 Batch-incremental

Several certainty-based batch-incremental methods have been proposed as well. For instance, one approach selects instances within the batch that are close to the separating hyperplane of an SVM classifier [26]. These labeled instances are then added to a sliding window of training data. Every time new labeled training data is added to the window, the SVM is rebuilt. It is shown that the approach outperforms random sampling on several textual datasets. I think this result is quite surprising, since the approach does not explore full input space in its selections, in order to detect concept drifts that happen further away from the decision boundary. Apparently, the drifts in the textual datasets took place close to the decision boundary.

Other batch-incremental methods use the query by committee strategy that was explained previously [22]. For example, a maximum variance method has been proposed that selects the instances that have the highest ensemble variance [35]. A similar method is Actminer [27]. This method selects instances that are difficult to classify by checking whether the ensemble members agreed on its classification. Also, it selects outliers for labeling in order to detect new classes in the data.

A different type of method uses clustering techniques to select instances for labeling. For example, in ACLStream [18] k-means clustering is applied to the batch of unlabeled instances. An instance is most likely to be labeled when it is representative of its cluster, when the classifier is uncertain about its label and when its cluster contains multiple uncertain instances. The authors show that ACLStream outperforms random sampling, variable uncertainty strategy with randomization [37] and the maximum variance method [35]. However, in another experiment that uses the same datasets and budget values, ACLStream performed worse than random sampling [19], so results are inconsistent. Another clustering-based method is COPAL [23]. This approach also starts with k-means clustering of the unlabeled instances of the batch. It repeatedly uses a probabilistic active learning method to select the cluster that will profit most from an additional label and selects an instance from this cluster such that the diversity among labeled instances within the cluster is maximized. When the requested number of instances is labeled, an incremental classifier is trained using the labeled instances. It has been shown that COPAL outperforms ACLStream,

and is also slightly better than the instance-incremental DBALStream method.

Active learning methods that can deal with class imbalanced and non-stationary data streams have not been proposed very often. In the next section, such a method is discussed.

1.4 Combining the problems

Recently, the combination of problems of concept drift, class imbalance and partial labeling has been considered for one of the first times, by suggesting the batch-wise Reduced Labeled Samples (RLS) framework [3]. In the RLS framework, a static base classifier is first trained on a fully labeled batch of data, where SMOTE is applied to deal with class imbalance. The minority samples of this first batch are saved to memory. Also the samples that the classifier is uncertain about are saved to memory. Together these saved samples are the borrowed samples. For subsequent batches, the same model is applied until concept drift is detected. Only then is the base classifier retrained. Concept drift is detected when the classifier has a performance drop from one batch to the next. This performance drop is measured using a selected subset of labeled samples for each batch. These critical samples have the lowest Euclidean distance to the borrowed samples from the previous batch [2]. The labels of the critical samples are requested, and the performance of the model can be evaluated. If concept drift occurs, the model is retrained on the subset of critical samples. An ensemble version of RLS has also been proposed [4].

Thus, a part of the RLS framework is an active learning strategy that selects critical samples. To my current knowledge, this active learning strategy does not have a name, therefore I will call the strategy 'critical sampling'. Critical sampling has not been compared to other active learning strategies besides as a part of the RLS framework. In this study, I compare critical sampling to other strategies and try to improve its performance.

In critical sampling, instances are labeled when they are close to the minority- and uncertain instances from the previous batch. In most scenarios, this will give a good performance. Uncertain instances in the current batch are likely to be similar to uncertain instances in the previous batch, so this approach tends to select uncertain instances. Also, minority instances in the current batch are likely to be similar to the minority instances in the previous batch, so this approach oversamples minority instances and undersamples majority instances. I think, however, that the method misses some exploration of input space that ensures all concept drifts to be detectable. When the concept of the minority class changes gradually, this could still be noticed by this method. When the concept of the minority class experiences a big sudden change, however, it can be the case that new minority instances will not resemble old minority instances, causing the current method to fail in detecting this change.

1.4.1 Research question and approach

As we have seen so far, there are not many active learning strategies that can deal with class imbalance and concept drift. One of the only methods that deals with these problems, critical sampling, cannot deal with all types of concept drift, because it does not explore full input space. In the current project, it is tested whether exploration of input space will indeed improve the performance of critical sampling. A hypothetical fraud detection scenario is used as inspiration. This scenario is described in the methodology section.

In general, this study compares several active learning strategies within the fraud detection scenario. There are two research questions: Does adding randomization to critical sampling improve its performance when dealing with an imbalanced non-stationary data stream? And does randomized critical sampling outperform the other active learning strategies when applied to an imbalanced non-stationary data stream? Both questions are expected to be answers with 'yes'. Adding randomization to critical sampling should improve its performance, because it will cause exploration of full input space and therefore should allow critical sampling to detect sudden concept drifts. Also, randomized critical sampling should outperform other active learning strategies, because it is the only strategy that selects most informative (uncertain) instances, that oversamples minority instances in order to deal with class imbalance and that explores full input space in order to detect all concept drifts.

These research questions are answered by assuming a fixed data stream framework in which different active learning strategies are compared: Random sampling, uncertainty sampling [26], randomized uncertainty sampling [37] and critical sampling [3]. A new strategy is proposed that combines random sampling and critical sampling: Randomized critical sampling. A data stream generator is implemented that creates a data stream similar to the data stream in the fraud detection scenario. The study is implemented using Massive Online Analysis (MOA) [5].

2 Methodology

2.1 Hypothetical fraud detection scenario

The fraud detection scenario takes the perspective of a governmental organization, e.g. one that is responsible for providing subsidies or student's loans. This organization wants to make sure that people who receive such a subsidy are also entitled to receive it, and therefore the organization wants to create a predictive model that can recognize fraudulent cases. Periodically (e.g. once in a month), a batch of data becomes available. Only a part of the instances from this batch can be verified for fraud, because of capacity and time restrictions. This is the problem of partial labeling. The predictive model indicates which of the instances should be verified for fraud in order to know their truth labels. At the same time, the model should take into account class imbalance: Fraudulent cases are rare compared to non-fraudulent cases. Furthermore, it should deal

with concept drift. Fraud patterns change over time, due to the discovery of new fraud methods and changes in regulations and circumstances. These changes cause the predictive model to become outdated. Therefore, new labeled data should be fed into the predictive model from time to time, in order to keep it up-to-date.

Fraud detection is a very complicated task, with many other co-occurring problems. For example, some fraudulent cases involve more money than others. Also, some fraudulent cases are more difficult to detect, causing the fraud verification process to take more time or money. This can give rise to delayed labeling or false labeling. Another problem involves varying amounts of data per batch and varying budgets for fraud verification. Furthermore, new features can become available over time for the predictive model to use. In the current fraud detection scenario, it is assumed that these problems do not exist. To be specific, the following assumptions are made:

- The predictive model does not need to take into account the amount of money that is involved in the fraud cases. The goal of the predictive model is to accurately detect fraud cases, irrespective of the money involved in these cases.
- The fraud verification process takes constant time and money. The requested labels of a batch are available before the next batch of data is available.
- The batch size is constant over time. Also, the number of instances that can be verified for fraud is constant over time.
- The feature set on which the predictions are based, is constant in size over time. In other words, no features are removed or added during the process.

2.2 Data stream framework

In order to compare the active learning strategies to each other, a fixed batch-incremental data stream framework is used that is similar to the one used by Lindstrom et al. [26]. In their approach, the instances within the batch are selected that are close to the decision boundary of an SVM classifier. Each time labeled data is added to the training window, the SVM is rebuilt. Similarly, in the current data stream framework an active learning strategy is used to select instances for labeling from each incoming batch of data. These labeled instances are then added to a sliding window of training data and the classifier (an SVM) is rebuilt using the window of data. Thus, the framework uses no explicit drift detection. The pseudo code of the data stream framework is shown in Algorithm 1. Note that the active learning strategy is given as input variable.

Data: Data stream D, Labeling budget B, Selection Strategy $S()$, Max window length L
Result: Predictions P of classifier for each data batch of D

```
Initialize classifier C;
Initialize train window W;
while Stream D has next batch do
    Receive incoming batch V from stream D;
    Create predictions P = C(V);
    Select instances to be labeled X = S(V, B);
    Receive selected labels (X, Y);
    Add labeled instances to train window W = W ∪ (X, Y);
    while Size(W)> L do
        Drop oldest instance from W;
    end
    Retrain classifier C on W;
end
```

Algorithm 1: Batch-Incremental Data Stream Framework

2.3 Data

For the current project, a dataset is needed that meets the following requirements. First, the dataset should contain many instances recorded across time, in order to simulate a data stream over a long period of time. Second, the dataset should give rise to a binary classification problem, where one class occurs much more often than the other one. Third, the dataset should contain both gradual and sudden concept drifts.

To my current knowledge, a data set that meets all requirements is not publicly available. Therefore, a new data stream generator was implemented that can generate data streams that meet all requirements. This data stream generator was implemented as an extension to the MOA framework [5], and makes use of the well known KDD '99 dataset [1]. In the next sections the dataset and the data stream generator are described.

2.3.1 KDD '99

KDD '99 [1] is a dataset that was used for The Third International Knowledge Discovery and Data Mining Tools Competition, which was held in conjunction with KDD-99 (The Fifth International Conference on Knowledge Discovery and Data Mining). The task of the competition was to detect network attacks in the dataset based on the 41 network connection characteristics. The dataset was generated in a military network environment where a wide variety of network attacks were simulated. The training set contains 4.898.431 observations and 25 classes ('normal', and 24 different types of network attacks) [31]. The test data set contains 311.029 observations, introduces 14 new attack classes and has a different probability distribution than the training set. The attack classes fall

into four main categories: Denial of Service (DoS), User to Root (U2R), Remote to Local (R2L) and Probing.

Some problems exist with the KDD '99 dataset [31]. First, the dataset contains quite some redundancy. There are 1.074.992 distinct observations in the dataset, which is only about 20% of the full size. Second, many instances are very easy to classify and difficult instances are rare. Third, the dataset contains too many observations for most applications, causing many researchers to use a smaller version that contains 10% of the original size. To solve these problems, a cleaned and smaller version (NSL-KDD) has been proposed [31], containing 148.517 observations. In NSL-KDD, redundant observations are removed and difficult instances are overrepresented.

NSL-KDD is commonly used as a benchmark anomaly detection task, but has also recently been employed as dataset for testing data stream algorithms [18] [19]. In the latter case, the train and test set are concatenated. Often the attack classes are merged together into one 'attack' class. This way, a sudden concept drift happens when transferring from the probability distribution of the training set to the distribution of the test set, since the underlying causes of the 'attack' concept changes.

The NSL-KDD dataset does not meet the requirements for the current project. The number of instances is rather low, especially because the 'attack' class should be undersampled to make the data stream class imbalanced. Furthermore, there is only a sudden drift and it happens at the end of the stream, leaving little time for the classifier to regain performance. Also the original KDD '99 dataset does not yet meet the requirements, since it does not contain concept drift nor class imbalance and it contains duplicate instances. Therefore, a data stream generator that meets all requirements is implemented that generates a stream using instances of the original KDD'99 dataset. In the next section it is described how this generator works.

2.3.2 Data stream generator

The classification task for this stream generator is discriminating between 'attack' and 'normal' observations. The data stream generator simulates concept drift by sampling from different types of the 'attack' class across time. The generator is implemented as an extension to MOA [5]. For an overview of its implementation details and some design choices, see Appendix A. Here, a brief description of the final generator is given.

The generator does not use all KDD '99 instances. First, duplicate instances are removed. Second, not all attack types have enough instances to sample from. Main categories 'R2L' and 'U2R' have 999 and 52 instances, respectively. Therefore all attack types from these categories were excluded from the set. Attack types 'land' and 'pod' from category 'DoS' have 19 and 206 instances respectively, and were also excluded from the set. All other attack types were included in the data set. Table 1 depicts the frequencies of all included attack types.

Figure 1 illustrates schematically how the KDD'99 Data Stream Generator

Total	**1.073.716**
Normal	812.814
Attack	260.902
Attack type: *DoS*	*247.042*
... Neptune	242.149
... Smurf	3007
... Back	968
... Teardrop	918
Attack type: *Probe*	*13.860*
... Satan	5019
... Ipsweep	3564
... Portsweep	3723
... Nmap	1554

Table 1: Number of distinct instances per included attack type as taken from the original KDD '99 training set.

creates a drifting data stream. Instances are randomly sampled from the different types of KDD'99 instances. Sampling from the 'normal' class has the highest probability. Sampling from an 'attack' class has a lower probability. Initially in the stream, samples can only be drawn from a certain subset of 'attack' types, for instance 'smurf' and 'back'. During the stream, new attack types are introduced, for instance 'ipsweep' and 'nmap'. The transition between the sampling probability of the initial subset to the second subset is modeled by a sigmoid function. The width of this sigmoid can be tuned such that a small width results in a sudden concept drift and a big width results in a gradual concept drift[3].

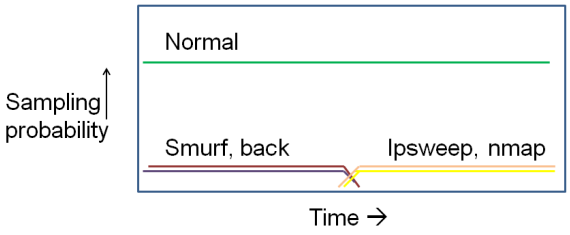

Figure 1: Illustration of how varying the sampling probability of attack subtypes across time can induce a (sudden) concept drift.

[3]Note that it is the probability of observing the initial and final concepts that changes gradually over time. In other words, there are no intermediate concepts.

2.4 Active Learning Strategies

In this project, five active learning strategies are compared. Each one of them deals with one or more of the three problems of concept drift, partial labeling and class imbalance. A strategy deals with concept drift if it explores input space such that all drifts are being detected. It deals with partial labeling by selecting most informative instances, for example those that the classifier is most uncertain about. And it deals with class imbalance by selecting more minority samples such that the labeled data stream is more balanced. Each of the five active learning strategies uses one or a combination of these methods. In the following sections, the strategies are discussed in more detail.

The number of instances that the strategies should select depends on the budget and the batch size. Suppose that b denotes the *budget*. For instance, if $b = 0.1$, then 10% of the instances of the batch are labeled. Suppose also that N denotes the number of instances in the batch. Then the number of instances that should be selected, s, can be computed as $s = b \cdot N$.

2.4.1 Random Sampling

This method randomly selects s instances from the batch. It is a baseline that all other methods should beat. Because this method explores full input space, it can handle concept drifts very well. However, it does not deal with class imbalance. Moreover, it does not choose the most informative instances for the classifier.

2.4.2 Uncertainty Sampling

Uncertainty sampling uses the uncertainty of a classifier in the class label of an instance as an estimate of the impact of the label on classification performance [22]. The instance with the highest uncertainty is selected for labeling. In the current study, the maximum posterior estimate of the classifier is used as uncertainty measure, where low values indicate high uncertainty. All instances of the batch are sorted on uncertainty and the s most uncertain instances are selected for labeling.

Without concept drift, uncertainty sampling is expected to perform better than random sampling, because it selects more informative instances. When concept drift appears, however, it is expected that random sampling performs better, because it explores full input space and can detect drifts more easily, where uncertainty sampling does not.

2.4.3 Randomized Uncertainty Sampling

Uncertainty sampling cannot detect all concept drifts in input space, because it only selects instances close to the decision boundary. Therefore, when it is used to sample from a non-stationary data stream, it is important to continuously explore input space, for example, by randomly selecting instances [37]. In the

current study, randomized uncertainty sampling is implemented by using a parameter r, the *random ratio*, that denotes the proportion of randomly selected instances. First, $(1 - r) \cdot s$ instances are selected from the batch using uncertainty sampling. Then, from all other instances, $r \cdot s$ instances are selected at random.

It is expected that randomized uncertainty sampling performs better than both random sampling and uncertainty sampling. Without concept drift, it can use the informative instances that are selected to achieve a high classifier performance. With drift, it can use the randomly selected instances to detect the drift easily and regain its old performance easily.

2.4.4 Critical Sampling

As explained in the introduction, critical sampling is the active learning strategy that is used within the RLS framework by Arabmakki et al. [3]. This strategy borrows the known minority instances and the instances that are close to the decision boundary from the previous batch. It then selects those instances in the current batch that are close to these borrowed instances, such that the selected instances tend to be either minority or uncertain instances as well. In the current project, critical sampling was implemented as a stand-alone active learning strategy that could be used in the data stream framework as depicted in Algorithm 1. To this end, several small changes were made to the algorithm as compared to its description in [2]:

1. In the original algorithm, labels of instances can be verified during the active learning strategy, whereas in the current framework, this is not possible. Therefore, one change was made. In the original algorithm, for each borrowed instance, the K closest current instances are stored in a preselection, and those current instances that are preselected most often are selected for labeling. If the selection does not contain any minority instance, K is increased and the selection process starts over (rules 12 and 13 of the pseudocode on page 36, in [2]). In the current implementation, a heuristic is used for incrementally increasing K: the preselection should contain twice as many unique instances as is needed for the final selection. If this condition is not met, K is incremented and the selection process starts over (see Algorithm 3, rule 7).

2. In the original algorithm, all support vectors in the previous batch are selected and added to the borrowed instances (page 36 in [2]). Support vectors are the instances that are closest to the decision boundary of an SVM classifier. Some implementations of the SVM classifier have the option to output the support vectors. The implementation of SVM[4] that was used in the current project, however, does not have this option. Therefore, it was chosen to implement the selection of the uncertain instances

[4]The WEKA implementation of John Platt's sequential minimal optimization algorithm for training a support vector classifier was used. See `http://weka.sourceforge.net/doc.dev/weka/classifiers/functions/SMO.html` (visited 2017-07-08).

in the previous batch in the same manner as in uncertainty sampling. A downside of this approach is that the number of borrowed uncertainty instances should be set as a parameter. In the current implementation this is done using the minority ratio parameter (see Algorithm 4). This parameter takes a value between 0.0 and 1.0 and stands for the proportion of minority instances in the borrowed instances. All known minority instances from the previous batch are selected as borrowed instances and uncertain instances are added until the given minority ratio is fulfilled. For example, if 10 minority instances are selected and the minority ratio is 0.333, then 20 uncertain instances are selected. This parameter is tuned such that critical sampling performs best (see Appendix B).

3. Sometimes, the instances that are labeled do not contain many minority instances. To ensure that there will be enough borrowed instances available, the minimum number of borrowed instances is set to 8. If not enough minority and uncertain instances are selected in the default manner, uncertain instances are added until the minimum number of 8 borrowed instances is reached.

4. In the RLS framework, ADASYN oversampling technique is used to make the final selection more balanced (page 41, [2]). In the current framework, this step is left out to ensure honest comparison between the active learning strategies.

Algorithms 2, 3 and 4 show how the current implementation of critical sampling works. As shown in Algorithm 2, critical sampling consists of two steps. First, instances are selected for labeling. After these instances are labeled, the borrowed instances are selected. In the current project, these two steps are subsequent, because all truth labels are known. In practice, however, some time should be in between these two steps, because labeling instances takes time.

The selection of instances is further detailed in Algorithm 3. First it is computed how many instances should be selected for labeling (rule 2), and a distance table is computed containing the distance from each current instance to each borrowed instance (rule 4). Then, for each borrowed instance the K closest current instances are preselected. This is repeated until the number of unique preselected instances is equal or larger than twice the sample size (rules 7 to 15). If this condition does not hold, K is incremented with 10. Next, it is counted how often each current instance is preselected (rule 17), after which the current instances with the highest counts are finally selected for labeling (rules 19 and 20).

The selection of borrowed instances is further detailed in Algorithm 4. All known minority instances are selected for labeling (rule 1). The number of needed uncertainty instances is computed based on the minority ratio parameter (rule 3) and in the case that too less minority instances are known, the number of needed uncertainty instances is increased, such that at least 8 borrowed instances will be selected (rule 5). The current instances are sorted on uncertainty (rules 8 and 10) and the most uncertain instances that are not already selected as

known minority instances are added to the borrowed instances (rules 11 and 12).

It is expected that critical sampling works very well without drift, because informative (uncertain) instances are selected and minority instances are selected more often to ensure a more balanced data stream. Therefore, it should perform better than random sampling and uncertainty sampling in case there is no drift. In case there is drift, however, it is expected that critical sampling will not be able to handle drifts very well, because it does not explore input space. Thus, it should have difficulty recovering to its previous performance after drift.

Input: Current Instances I, Labeling Budget b, Borrowed Instances B, Classifier C, Minority Ratio mr
Output: Selected Instances S

```
1 S = SelectInstances(I, b, B) ;                    /* See Algorithm 3 */
  /* Note:  S should now be labeled.                                    */
2 B = setBorrowed(S, I, C, mr) ;                    /* See Algorithm 4 */
3 return S
```

Algorithm 2: Critical Sampling

Input: Current Instances I, Labeling Budget b, Borrowed Instances B
Output: Selected Instances S

Number of instances $N = size(I)$;
Sample size $s = floor(b \cdot N)$;
Dynamic parameter $K = 10$;
Create distance table $D(I, B)$;
Initialize set of indices $J = null$;
Initialize set of unique indices $U = null$;
while $size(U) < 2 \cdot s$ **do**
    clear indices in J;
    foreach bi *in* B **do**
        Select closest K instances in I to bi using $D(I, B)$;
        Put indices of these instances in J;
    end
    $U = unique(J)$;
    K = K + 10;
end
foreach i *in* I **do**
    Count how often i's index is in J;
end
Sort I on counts in descending order;
S = top s instances from I;
return S

Algorithm 3: Select Instances

Input: Selected Instances S, Current Instances I, Classifier C, Minority Ratio mr
Output: Borrowed Instances B

Select minority instances M from S;
Set $m = size(M)$;
Set $u = floor(m \cdot \frac{1-mr}{mr})$;
if $m + u < 8$ **then**
    Set $u = 8 - m$;
end
foreach i *in* I **do**
    Get maximum posterior of C for i;
end
Sort I on maximum posterior value in ascending order;
Set U = first u instances in I that are not already in M;
$B = M \cup U$;
return B

Algorithm 4: Set Borrowed

2.4.5 Randomized Critical Sampling

Randomized critical sampling is a proposed improvement of critical sampling, that not only selects uncertain and minority instances, as critical sampling does, but also explores the input space by selecting some instances at random. Randomized critical sampling is implemented in a similar fashion as randomized uncertainty sampling, using a parameter r, the *random ratio*, that denotes the proportion of randomly selected instances. First, $(1-r) \cdot s$ instances are selected using critical sampling and from all other instances $r \cdot s$ instances are selected at random. In case of a drifting data stream, it is expected that randomized critical sampling outperforms all other methods, because it can deal with all three problems of concept drift, partial labeling and class imbalance. In case of no drift, it should perform similar as critical sampling.

2.5 Evaluation

Basic evaluation measures, such as accuracy, cannot be used for evaluating classifiers on imbalanced data. This type of measures will overestimate the performance of the classifier. Suppose for example that a classifier is trained on a dataset where the majority class represents 95% of the data. If this classifier simply classifies each instance as a majority instance, it would yield an accuracy of 95%, without actually having learned anything.

To deal with this problem, evaluation measures have been proposed that take both recall and precision into account. For example, the Area Under the Curve (AUC) measures the area under the ROC curve. The ROC curve plots the fraction true positives out of all actual positives against the fraction false positives out of all negatives. Ideally, the AUC is equal to 1, meaning that all positive instances have been identified as positives, but no negative instance has been falsely identified as a positive.

Previously, AUC has been used to evaluate classifiers on imbalanced and evolving data streams, by evaluating the classifier on the data stream as a whole. However, this is not a good evaluation measure for evolving data streams, since it cannot detect drops in performance due to drifts very well. Therefore, prequential AUC has been proposed [6]. This method evaluates the classifier on a shifting local time window, such that temporal patterns in the data can be taken into account. This measure can be used to evaluate instance-incremental classifiers. In order to evaluate batch-incremental classifiers, as is the goal in the current project, AUC computed per batch can also be used for evaluation.

The AUC measurement does not take into account the certainty of the classifier in its classification. If a classifier correctly classifies the instances, it gets the perfect score, even if it was uncertain about its classifications. In this manner, the AUC measurement can overestimate the performance of a classifier. The scored AUC measurement has been proposed to correct for this [33]. This measurement does not only take into account the correctness of the classification, but also the maximum posterior probability. This way, the scored AUC is a more strict evaluation measurement than the normal AUC measurement.

The current project uses scored AUC computed per batch as evaluation of the active learning strategies. To this end, the Java implementation of the BasicAUCImbalancedPerformanceEvaluator class by [6] was used, that is available online.

2.6 MOA framework

This project uses Massive Online Analysis (MOA) [5] for implementation. This is an open source data stream framework, implemented in Java. Many data stream mining papers have been published that have made use of MOA. Several existing classifiers and evaluation methods are available. MOA is compatible with Weka. In this project, the Weka implementation of Sequential Minimal Optimization algorithm is used to train an SVM classifier. The active learning strategies have not been implemented yet in MOA, so these were implemented in the current project. The implementation of critical sampling was implemented following Arabmakki's PhD thesis [2]. The scored AUC evaluation measurement is available as an extension to MOA [6].

3 Results

3.1 Experimental settings

Prior to the experiments, some pilots have been run. The pilot results and their discussion can be found in Appendix B. Here, the experimental settings that were found most effective are described.

Based on the pilots, parameter values of the active learning strategies were decided. For randomized uncertainty sampling, the random ratio was set to 0.5. For critical sampling, the minority ratio was set to 0.4. For randomized critical sampling, the minority ratio was set to 0.4 and the random ratio was set to 0.3.

For each combination of conditions, 400 trials were run. Each trial was run using a randomly drawn random seed, causing each data stream to be different. Based on the pilots, it was chosen to set the number of attack types in the data stream to 4. Each data stream consisted of 50.000 instances. If there was drift, the drift's position was at instance 25.000. If the drift was sudden, the width of the drift was 1, whereas if it was gradual, the width was 25.000 instances long. This means that a gradual drift started (a bit before) instance 12.500 and ended (a bit after) instance 37.500. The stream contained a certain proportion of minority instances. This proportion depended on the attack ratio. Two attack ratio's were tested: 0.05 and 0.1.

The data stream was split in batches with 500 instances each. The classifier was trained using a sliding window of 300 instances. The first batch of the stream was used to fill this window. The classifier was trained on the data in the window. For critical sampling and its randomized version, all minority instances present in the window were set as initial borrowed instances. Also, some other random instances were added as borrowed instances. For all subsequent

batches, the borrowed instances were selected from the previous batch of data as described previously.

After the first batch, the active learning strategy selects a certain proportion of each batch for labeling. This proportion depends on the budget. Four budgets were tested: 0.05, 0.1, 0.15 and 0.2.

3.2 Experiments

Figures 22 and 23 in Appendix C give an overview of all results obtained during the experiments. The figures show box plots of the average performance of the active learning strategies across all conditions. As can be seen, the pattern of the performance of the different strategies is consistent across conditions. In almost all conditions, randomized uncertainty sampling performs best, followed by uncertainty sampling and random sampling. In most conditions, uncertainty sampling performs slightly better than random sampling, but this does not hold for all conditions. In all conditions, randomized critical sampling performs slightly worse than the random sampling baseline and critical sampling performs much worse than baseline.

Figure 2 shows the average performance of the strategies across time[5], in the condition where the attack ratio is 0.05 and the budget is 0.1. The shaded areas around the lines give the standard error (SE), that is quite small in general due to the large number of trials. The figure clearly shows the effect of concept drift on the performance of the classifier. If there is no drift, the performance is quite constant over time. A sudden drift gives a sudden drop in performance, where only some strategies allow the classifier to regain its original performance. A gradual drift gives a gradual and longer drop in performance, where again only some strategies allow the classifier to restore to its original performance.

In all conditions, randomized uncertainty sampling performs best. It has the best performance without drift and can also recover best from both sudden as gradual drifts. Uncertainty sampling starts quite good as well, but its performance decreases over time, even when there is no drift. Also, it is clear that it cannot recover from drifts. Randomized critical sampling performs slightly worse than baseline in all conditions. It can recover from drifts quite well, as it reaches its old performance quite fast. Critical sampling performs much worse than baseline in all conditions. Moreover, critical sampling cannot recover from sudden drift as easily as it can recover from gradual drift.

3.3 Validation

Based on the results discussed so far, it can be said that critical sampling does not perform as expected. It was expected that it would outperform both uncertainty sampling and the random sampling baseline, but it turns out to perform much worse than baseline.

As a validation of critical sampling, some extra results were generated, as shown in Figure 3. In the left plot it is shown that critical sampling and its

[5]The patterns shown in this figure are similar across conditions.

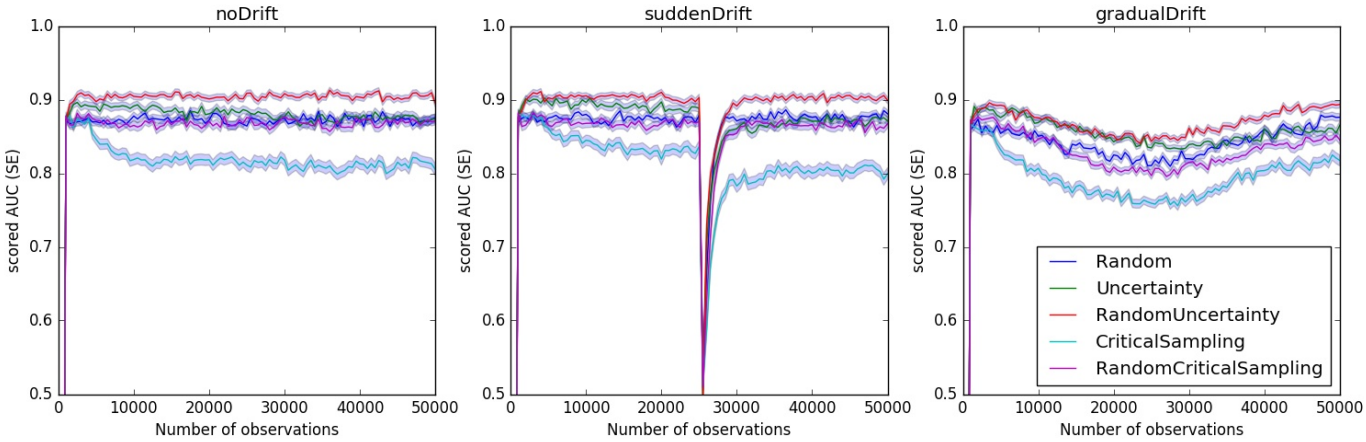

Figure 2: Scored AUC of classifier for the five active learning strategies when there is no drift, a sudden drift or a gradual drift. The randomized uncertainty strategy performs best. The uncertainty strategy performs above or at baseline performance. Randomized critical sampling performs slightly worse than baseline. Critical sampling performs much worse than baseline. *(attack ratio = 0.05, budget = 0.1)*

randomized version select an higher proportion of positive instances than the other methods. Thus, critical sampling indeed oversamples the minority class, as it should do. In the right plot, it is shown that critical sampling and its randomized version also select more uncertain instances than the random sampling baseline. Thus, critical sampling also tends to select instances that are closer to the decision boundary.

Figure 4 shows the performance of the strategies in the condition with no drift, where the performance measure is accuracy instead of scored AUC. As can be seen, all strategies now tend to perform above baseline, but critical sampling still performs worse than the others. Thus, critical sampling has a relatively high accuracy, but a low scored AUC. Apparently, critical sampling selects those instances that bias the classifier into mistakenly classifying a minority instance as a majority class.

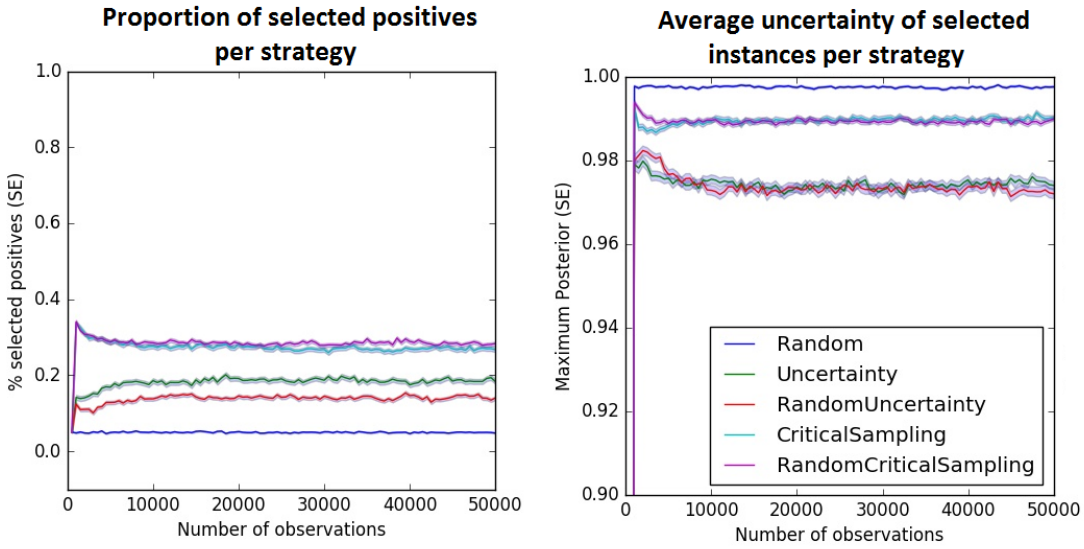

Figure 3: *Left:* Proportion of selected positives per strategy. Critical sampling and its randomized version select the largest proportion of positive instances. Uncertainty sampling and its randomized version select more positive instances than the random baseline. *Right:* Average uncertainty of selected instances per strategy (lower values indicate higher uncertainty). Critical sampling and its randomized version select more uncertain instances than the random baseline. Uncertainty sampling and its randomized version select the most uncertain instances. *(attack ratio = 0.05, budget = 0.1)*

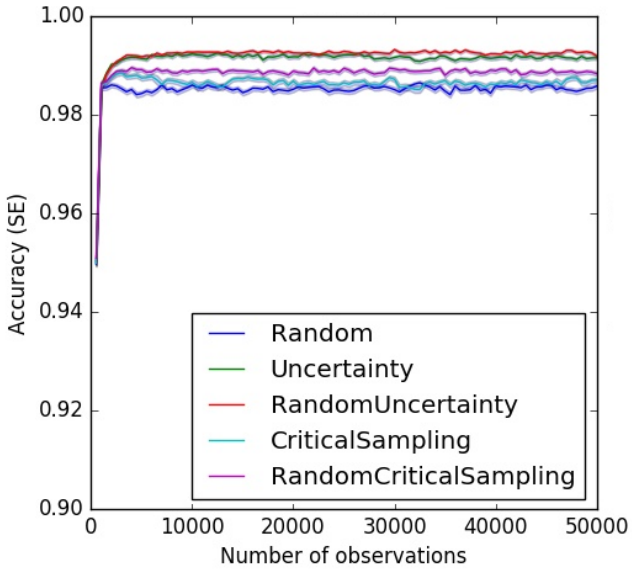

Figure 4: Accuracy of classifier for the five active learning strategies when there is no drift. All strategies tend to perform above baseline, with randomized uncertainty strategy performing best. Results were similar for conditions with drift. *(attack ratio = 0.05, budget = 0.1)*

4 Discussion

This study compared the performance of several active learning strategies on an imbalanced non-stationary data stream. It was expected that adding randomization to the critical sampling strategy would improve its performance and that randomized critical sampling would outperform all other strategies. Whereas it was the case that randomized critical sampling outperformed critical sampling, it was not the case that critical sampling or its randomized version outperformed any other active learning strategy or even the random sampling baseline. These results will now be discussed in further detail.

The other active learning strategies performed as expected. Uncertainty sampling performed better than the random sampling baseline. Moreover, as expected, it could not deal with gradual or sudden drifts, as it never regained its initial performance. In the no drift condition, however, uncertainty sampling also seemed to decrease in performance across time. This could be caused by sampling from areas with high Bayesian error. It is known that, in later learning phases, uncertainty sampling tends to sample from regions where classification performance cannot be improved [22]. Furthermore, randomized uncertainty sampling also performed as predicted. It had the highest performance in case of no drift, as well as in case of sudden or gradual drifts. Randomized uncertainty sampling regained its original performance quickly after a drift had occurred, due to exploration of full input space [37].

It was found that randomized critical sampling outperformed critical sampling. This is not a surprising result, since critical sampling performed worse than the random sampling baseline. By adding randomization to critical sampling, some bad selections made by critical sampling are replaced by better random selections, causing the overall performance to increase. There are two things that should be noted, however. First, as was expected, critical sampling had more difficulty recovering from a sudden drift than recovering from a gradual drift. This was expected, because critical sampling uses knowledge from the previous batch to select its instances. In a gradually drifting data stream, each batch is still quite similar to the subsequent batch, but in a sudden drift this is not the case. Second, as was also expected, randomized critical sampling recovered from drifts, even from sudden drifts. Thus, the random exploration of input space caused the classifier to detect the new position of the minority class and to change its decision boundary accordingly.

4.1 Analyzing critical sampling

The question that remains is, why did critical sampling perform so much worse than expected based on previous findings (e.g. [3] [2])? Multiple explanations are possible. First, it could be that the changes that were made in the current implementation compared to the original algorithm as described in [2] have caused critical sampling to not do what it should be doing. Based on the validations, it seems that this explanation is very unlikely. Critical sampling and its randomized version select the highest proportion of positive instances as com-

pared to the other active learning strategies, as should be the case. Also, they select on average more uncertain instances than the random sampling baseline, as should be the case as well.

A second explanation could be that oversampling the minority instances does not work for the specific data stream that was used in this study. This seems to be indeed the case. First, during the pilot studies it was found that applying SMOTE to the training window before training the classifier did not add any significant improvement in performance (see Appendix B). SMOTE is a popular minority oversampling technique that balances the classes in the data set [7]. Furthermore, in the pilot studies it has been found that a balanced version of the KDD'99 data stream has only a slightly better performance than an imbalanced one (see Appendix B, Figure 15). So apparently, oversampling the minority instances of the KDD'99 data stream is not necessary to deal with the imbalance in the stream. For a further discussion of this finding, see Appendix B.

However, this finding does not fully explain the result that critical sampling performs below the random sampling baseline. Oversampling the minority class does not improve performance, but also should not decrease performance. Furthermore, critical sampling selects uncertain instances, and this should improve the performance of the classifier, at least in the beginning of the stream.

There is a third explanation that could also account for this last point. It could be that critical sampling causes some sort of sampling bias. In the validations, it was found that critical sampling has a slightly higher accuracy, but a much worse scored AUC than random sampling. Thus, apparently the selections that are made by critical sampling cause the decision boundary to be shifted towards the minority instances, such that some minority instances are wrongly classified, but almost all majority instances are correctly classified. In other words, the classifier tends to correctly classify majority instances at the cost of misclassifying minority instances.

One explanation for this bias could be that critical sampling selects those instances that are close to most borrowed instances. In minority areas with multiple borrowed instances, it is more likely that instances will be selected for labeling (and will become new borrowed instances) than in minority areas with less borrowed instances. This pattern has a self enhancing effect. If there are multiple borrowed instances in an area in batch one, there will be more borrowed instances in that area in batch two and even more in batch three. This pattern gives rise to two detrimental effects. First, critical sampling tends to sample from dense minority regions only. And second, critical sampling tends to sample from one minority region only. Both of these effects cause the observed bias. I will first explain the first effect and why this causes the bias, and then I will explain how the second effect makes everything even worse.

The first detrimental effect is that critical sampling tends to sample from dense minority regions only. In dense minority areas, there is an higher probability for borrowed instances to appear, and thus there is an higher probability to select new borrowed instances from those areas. Figure 5 illustrates this pattern. Suppose that the minority instances are distributed across input space in

such a manner that some minority regions are dense and some are more sparse. In frame A, Figure 5, such a situation is illustrated. The left side of input space is much more dense than the right side. Initially, a random subset of the minority instances are present in the training window (frame B). All minority instances in this window are used as borrowed instances for the next batch. This situation is depicted in frame C, where all red circles denote borrowed instances and the blue circles denote current instances in the first batch. Note that more borrowed instances appear in the dense area than in the sparse area. Therefore current instances are close to more borrowed instances in the dense area. In frame D the instances that are selected for labeling are depicted. It can be seen that proportionally more instances are labeled in the dense area than in the sparse area. Frames E to H show that this pattern repeats itself, such that after a few batches all instances that are labeled come from the dense minority area only.

If critical sampling only selects instances for labeling that are either in the dense minority area or close to the decision boundary, this can give rise to the observed bias. This is illustrated in Figure 6. Suppose that we have the situation that is depicted in frame A: Minority instances occur in a dense and a sparse region, and the sparse region is adjacent to the region where all majority instances occur in input space. The optimal decision boundary is depicted as a black dashed line. Suppose this decision boundary is the current decision boundary of the classifier and critical sampling can select new instances to be labeled. The instances that are selected are depicted in frame B. Quite some minority instances from the dense region are labeled, along with many majority instances and some rare minority instances from the uncertainty margin close to the decision boundary. If a classifier is trained on these selected instances, its decision boundary will be closer to the minority instances than the previous (optimal) decision boundary. It underestimates the number of minority instances in the sparse region and overestimates the number of instances in the dense region, causing the decision boundary to be too close to the minority instances. This gives rise to the bias that is observed in the current study: Majority instances are correctly classified at the cost of misclassifying minority instances.

The second detrimental effect of selecting the instances that are close to most borrowed instances, is that critical sampling tends to sample from one minority region only. Suppose that minority instances are positioned in two dense areas in input space. The initial set of borrowed instances contains instances randomly spread across both dense minority areas. Suppose that, by coincidence, the first dense area has four borrowed instances whereas the second area has six borrowed instances. Now, instances from the second area have an higher probability to be selected as new borrowed instance than instances in the first area. So, in the new set of borrowed instances, there are even more instances from the second area than before. After a few batches, critical sampling will only sample instances from the second area and it will ignore the first area completely. This will cause the classifier to be ignorant about the position of many minority instances and misplace the decision boundary completely.

Why is this sampling bias observed in the current study, and why was it not

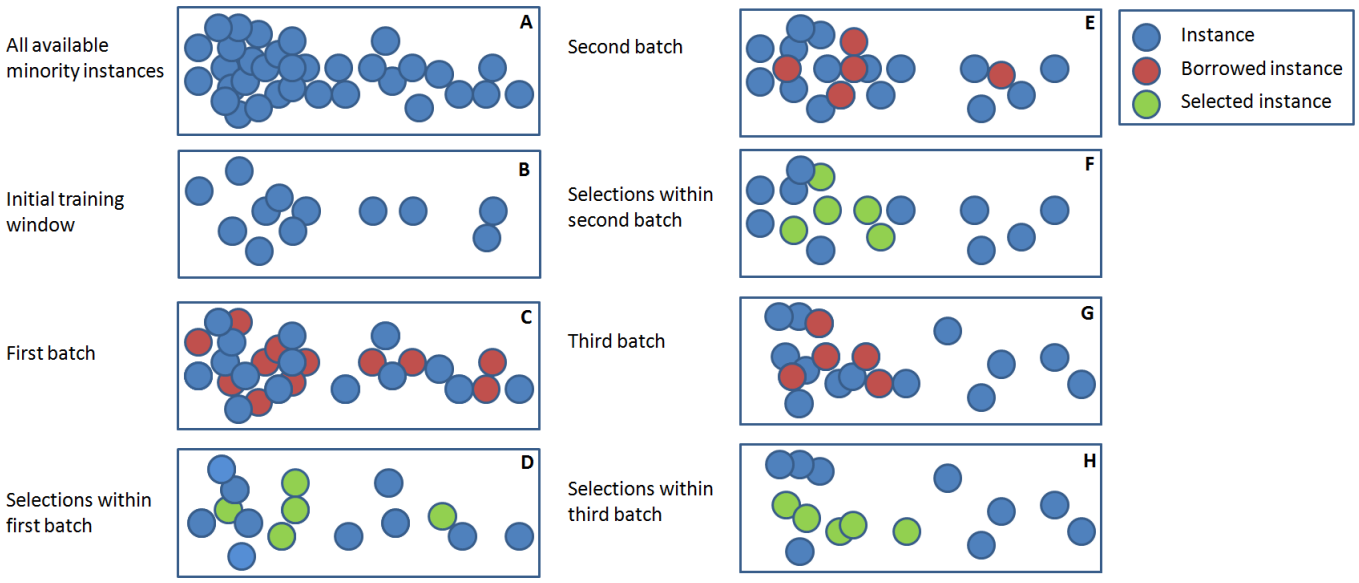

Figure 5: Illustration of why critical sampling tends to sample from dense minority regions. A. All available minority instances. Note that the region on the left is more dense than the area on the right. B. The initial training window contains this set of minority instances. All of them will be used as borrowed instances in the first batch. C. The first batch of data along with the borrowed instances. D. The selected instances within the first batch that were in the top K closest instances to most borrowed instances. E. The second batch of data along with the new borrowed instances. Note that four out of five borrowed instances are in the dense area. F. The selected instances within the second batch. G. The third batch of data with the new borrowed instances. Note that all borrowed instances are in the dense area. H. The selections within the third batch. All instances that are selected are in the dense region on the left.

found in previous studies? This could be caused by the nature of the KDD'99 dataset [1]. Figure 9 in Appendix A shows a scatter plot of the normal instances and the instances from the different attack types that appear in the KDD'99 data stream, after a Principal Component Analysis has been performed. It is clear that there are dense and more sparse regions of the same attack type. Also, the attack types are spread across multiple dense areas in input space. If critical sampling only selects instances from one of these areas, the classifier will be ignorant about the positions of most minority instances.

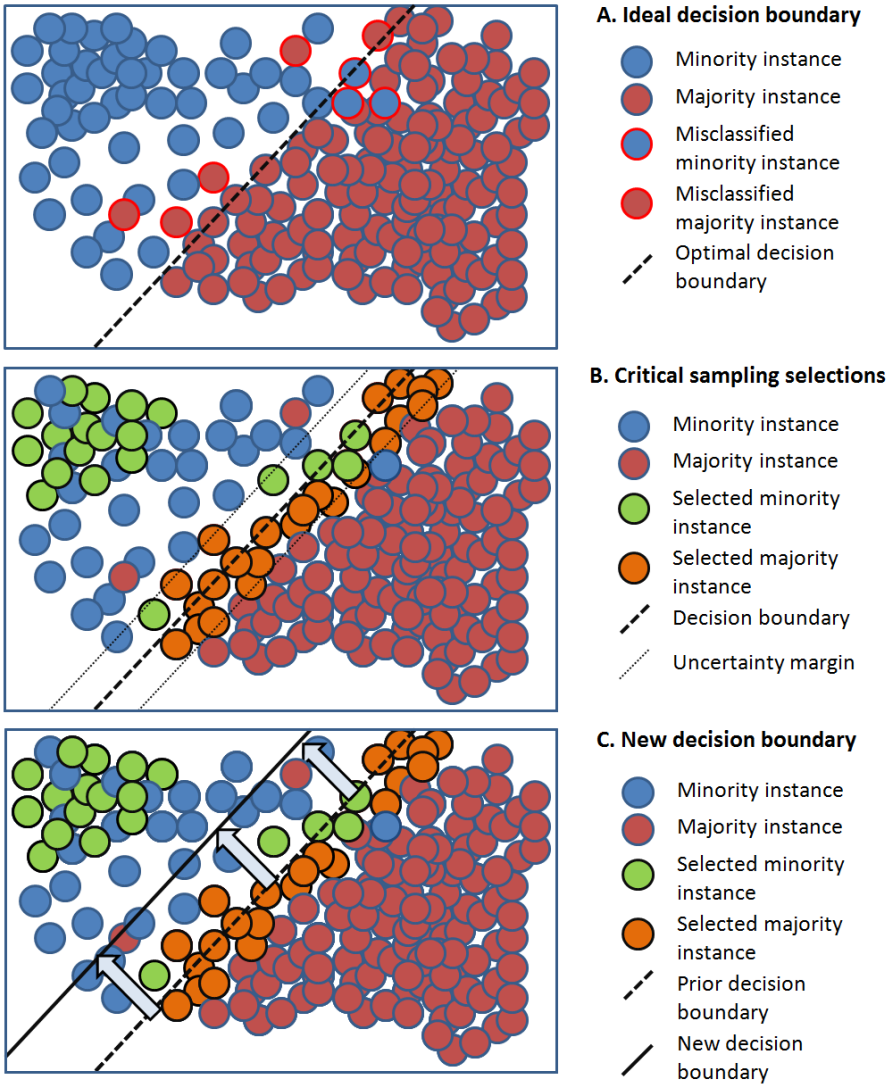

Figure 6: Illustration of why only sampling from dense minority regions can cause a bias. A. A depiction of the situation where the minority region close to the majority instances is a sparse region. B. Suppose that the optimal decision boundary is the current decision boundary of the classifier. The circles with a black edge denote the instances that will be selected for labeling by critical sampling. C. A classifier that is trained on only the selected instances will shift the decision boundary more in the direction of the minority class instances. This causes the accuracy to increase, since more majority instances are classified correctly, but the scored AUC measure to decrease, since quite some minority instances are misclassified.

4.2 Future research

Future research should repeat the current experiment with a different data set. For instance, the Covertype dataset[6] was used for validating the performance of the RLS-framework [2] of which critical sampling is a part, and thus it is expected that the bias does not have such a big impact on classifier performance for that dataset as compared to the KDD'99 dataset. Another step that is important to study is whether critical sampling is worth the extra computational time and resources it takes. For instance, it should be the case that critical sampling outperforms an uncertainty-based approach that is combined with an efficient oversampling technique such as SMOTE.

If this is indeed the case, extensions to critical sampling can be made that make it possible to deal with variations in positioning and denseness of minority data. For example, it could be studied whether a diversity approach as discussed in [13] could be combined with critical sampling. In [13], an active learning strategy is proposed that selects a subset of instances that contains informative instances, but at the same time is as diverse as it can be. The informativeness of an instance is operationalized as the weighted uncertainty of the ensemble classifiers in its label. The disparity between a pair of instances is defined as the product between their feature distance, which is the distance between the two instances in feature space, and their prediction distance, which is the dissimilarity of the predictions of the ensemble classifiers for these instances. From the batch of data, the subset of instances is selected that has approximately the maximum sum of all instances' uncertainty and their disparities, compared to all same sized alternative subsets. This approach could be combined with critical sampling, by adjusting critical sampling such that it gives instances a 'criticality score'. For example, during the algorithm it is counted how often each current instance appears in the top k closest instances of a borrowed instance. This count could be normalized into a score and added to the summation that should be maximized in the diversity-approach.

Another suggestion would be to combine critical sampling with a clustering-based approach, to ensure sampling from multiple separated areas in input space. Similarly as in COPAL [23], the batch of unlabeled instances could be clustered and a probabilistic active learning strategy could be used to select the cluster that will profit most from an additional label. Next, an instance from this cluster could be labeled that is close to most borrowed instances (if they exist in this cluster) and that gives rise to a diverse set of labeled instances within this cluster.

To conclude, the current study clearly confirms the importance of exploring the full input space when dealing with non-stationary data streams. Furthermore, it has contributed to the combined fields of data stream mining and active learning, by discovering what seems to be a sampling bias of the critical sampling strategy. It is suggested that this bias can be overcome by combining critical sampling with clustering- or diversity-based approaches.

[6] See `https://archive.ics.uci.edu/ml/datasets/covertype` (Visited 2017-07-08).

References

[1] *KDD Cup 1999*. Available on: `https://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html`.

[2] E. Arabmakki. *A reduced labeled samples (RLS) framework for classification of imbalanced concept-drifting streaming data*, 2016. Electronic Theses and Dissertations. Paper 2602. Available on: `https://doi.org/10.18297/etd/2602`.

[3] E. Arabmakki, M. Kantardzic, and T. S. Sethi. Rls-a reduced labeled samples approach for streaming imbalanced data with concept drift. In *Proceedings of the 2014 IEEE 15th International Conference on Information Reuse and Integration (IEEE IRI 2014)*, pages 779–786, Aug 2014.

[4] E. Arabmakki, M. Kantardzic, and T. S. Sethi. Ensemble classifier for imbalanced streaming data using partial labeling. In *2016 IEEE 17th International Conference on Information Reuse and Integration (IRI)*, pages 257–260, July 2016.

[5] A. Bifet, G. Holmes, R. Kirkby, and B. Pfahringer. Moa: Massive online analysis. *J. Mach. Learn. Res.*, 11:1601–1604, August 2010.

[6] D. Brzezinski and J. Stefanowski. Prequential auc: properties of the area under the roc curve for data streams with concept drift. *Knowledge and Information Systems*, pages 1–32, 2017.

[7] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer. Smote: Synthetic minority over-sampling technique. *Journal of Artificial Intelligence Research*, 16:321–357, 2002.

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

[9] G. Ditzler and R. Polikar. Incremental learning of concept drift from streaming imbalanced data. *IEEE Transactions on Knowledge and Data Engineering*, 25(10):2283–2301, Oct 2013.

[10] G. Ditzler, M. Roveri, C. Alippi, and R. Polikar. Learning in Nonstationary Environments: A Survey. *IEEE Computational Intelligence Magazine*, 10(4):12–25, November 2015.

[11] P. Domingos and G. Hulten. Mining high-speed data streams. In *Proceedings of the Sixth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '00, pages 71–80, New York, NY, USA, 2000. ACM.

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

[13] Y. Fu, X. Zhu, and A. K. Elmagarmid. Active learning with optimal instance subset selection. *IEEE Transactions on Cybernetics*, 43(2):464–475, April 2013.

[14] I. Gama, J.and Žliobaitė, A. Bifet, M. Pechenizkiy, and A. Bouchachia. A survey on concept drift adaptation. *ACM Comput. Surv.*, 46(4):44:1–44:37, March 2014.

[15] J. Gao, B. Ding, W. Fan, J. Han, and P. S. Yu. Classifying data streams with skewed class distributions and concept drifts. *IEEE Internet Computing*, 12(6):37–49, Nov 2008.

[16] A. Ghazikhani, R. Monsefi, and H. Sadoghi Yazdi. Online cost-sensitive neural network classifiers for non-stationary and imbalanced data streams. *Neural Computing and Applications*, 23(5):1283–1295, 2013.

[17] G. Hulten, L. Spencer, and P. Domingos. Mining time-changing data streams. In *Proceedings of the Seventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '01, pages 97–106, New York, NY, USA, 2001. ACM.

[18] D. Ienco, I. Bifet, A.and Žliobaitė, and B. Pfahringer. Clustering based active learning for evolving data streams. In *Proc. of the 16th Int. Conf. on Discovery Science*, DS, pages 79–93, 2013.

[19] D. Ienco, B. Pfahringer, and I. Žliobaitė. High density-focused uncertainty sampling for active learning over evolving stream data. In *Proceedings of the 3rd International Conference on Big Data, Streams and Heterogeneous Source Mining: Algorithms, Systems, Programming Models and Applications - Volume 36*, BIGMINE'14, pages 133–148. JMLR.org, 2014.

[20] B. Krawczyk. Learning from imbalanced data: open challenges and future directions. *Progress in Artificial Intelligence*, 5(4):221–232, 2016.

[21] B. Krawczyk, L. L. Minku, J. Gama, J. Stefanowski, and M. Woźniak. Ensemble learning for data stream analysis: A survey. *Information Fusion*, 37:132 – 156, 2017.

[22] G. Krempl. *Aktives maschinelles Lernen bei unvollständiger Information.* Habilitation thesis, available on: `http://edoc2.bibliothek.uni-halle.de/hs/content/titleinfo/61104`.

[23] G. Krempl, T. C. Ha, and M. Spiliopoulou. Clustering-based optimised probabilistic active learning (copal). In Nathalie Japkowicz and Stan Matwin, editors, *Proc. of the 18th Int. Conf. on Discovery Science (DS 2015)*, volume 9356 of *Lecture Notes in Computer Science*, pages 101–115. Springer, 2015.

[24] L. I. Kuncheva. *Classifier Ensembles for Changing Environments*, pages 1–15. Springer Berlin Heidelberg, Berlin, Heidelberg, 2004.

[25] R.N. Lichtenwalter and N.V. Chawla. Learning to classify data streams with imbalanced class distributions. *In: New Frontiers in Applied Data Mining. LNCS. Springer, Heidelberg*, 2009.

[26] P. Lindstrom, S. J. Delany, and B. M. Namee. Handling concept drift in a text data stream constrained by high labelling cost, 2010.

[27] M. M. Masud, J. Gao, L. Khan, J. Han, and B. Thuraisingham. Classification and novel class detection in data streams with active mining. In *Proceedings of the 14th Pacific-Asia Conference on Advances in Knowledge Discovery and Data Mining - Volume Part II*, PAKDD'10, pages 311–324, Berlin, Heidelberg, 2010. Springer-Verlag.

[28] N. C. Oza and S. Russell. Online bagging and boosting. In *In Artificial Intelligence and Statistics 2001*, pages 105–112. Morgan Kaufmann, 2001.

[29] C. H. Park and Y. Kang. An active learning method for data streams with concept drift. In *2016 IEEE International Conference on Big Data (Big Data)*, pages 746–752, Dec 2016.

[30] W. N. Street and Y. Kim. A streaming ensemble algorithm (sea) for large-scale classification. In *Proceedings of the Seventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '01, pages 377–382, New York, NY, USA, 2001. ACM.

[31] M. Tavallaee, E. Bagheri, W. Lu, and A. A. Ghorbani. A detailed analysis of the kdd cup 99 data set. In *Proceedings of the Second IEEE International Conference on Computational Intelligence for Security and Defense Applications*, CISDA'09, pages 53–58, Piscataway, NJ, USA, 2009. IEEE Press.

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

[33] S. Wu, P. Flach, and C. Ferri. *An Improved Model Selection Heuristic for AUC*, pages 478–489. Springer Berlin Heidelberg, Berlin, Heidelberg, 2007.

[34] J. Yuan, J. Li, and B. Zhang. Learning concepts from large scale imbalanced data sets using support cluster machines. In *Proceedings of the 14th ACM International Conference on Multimedia*, MM '06, pages 441–450, New York, NY, USA, 2006. ACM.

[35] X. Zhu, P. Zhang, X. Lin, and Y. Shi. Active learning from stream data using optimal weight classifier ensemble. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 40(6):1607–1621, Dec 2010.

[36] I. Zliobaite, A. Bifet, G. Holmes, and B. Pfahringer. Moa concept drift active learning strategies for streaming data. In Tom Diethe, José L. Balcázar, John Shawe-Taylor, and Cristina Tîrnauca, editors, *WAPA*, volume 17 of *JMLR Proceedings*, pages 48–55. JMLR.org, 2011.

[37] I. Zliobaite, A. Bifet, B. Pfahringer, and G. Holmes. Active learning with drifting streaming data. *IEEE Transactions on Neural Networks and Learning Systems*, 25(1):27–39, Jan 2014.

[38] I. Zliobaite, M. Pechenizkiy, and J. Gama. *An overview of concept drift applications*. Springer, 2015.

A KDD'99 Data Stream Generator

In this appendix, more details are given about the KDD'99 Data Stream Generator. First, its implementation is described. Then, some initial plans that did not prove to work are described.

A.1 Current implementation

The generator is implemented as an extension to MOA [5]. It involves three Java classes. The first class, *KDDData*, stores the data that will be sampled from. For each attack subtype, the data is stored separately. The second class, *KDDGenerator*, is responsible for generating a non-drifting data stream where samples are drawn from the data stored in *KDDData*. The inputs for this class are a random seed, the attack ratio and a set of attack subtypes. The random seed is used for all random processes while generating the stream. The attack ratio sets the percentage of 'attack' observations in the stream and thus influences the degree of class imbalance. For instance, if the ratio is 0.1, then 10% of the observations in the stream have the 'attack' class label and all other observations are 'normal'. The attack subtypes that are given as input are the attack subtypes that occur in the stream. For instance, if the given set is {'back','smurf'}, then the stream contains the attack subtypes 'back' and 'smurf'. These subtypes always occur in similar rates.

The third class is the *KDDGeneratorDrift*. It combines two non-drifting data streams generated by the *KDDGenerator* class by using the *ConceptDriftStream* class that was already implemented in MOA. This Java class is designed to combine two other streams. It starts with the first stream and the probability that every new instance comes from the second stream is defined by a sigmoid function. The parameters of this sigmoid function can be tuned. The width of the sigmoid and the position of the middle of the sigmoid can be given as input arguments to the *KDDGeneratorDrift* class. If the width of the sigmoid is small, the stream contains a sudden concept drift, and if it is large, the stream contains a gradual concept drift. Other input arguments are the percentage of 'attack' observations in both streams, the number of 'attack' subtypes in both streams and a random seed. This seed is used for all random processes such as selecting the 'attack' subtypes that are present in the streams and randomly drawing instances from either the first or second stream.

A.2 Original plan: Vary drift strength

One of the original goals of the KDD'99 generator was to vary the strength of the drift. In the following sections, it is explained how this goal was approached and why this did not turn out to work as expected.

A.2.1 Approach

The data stream generator simulates concept drift by varying the underlying causes of the 'attack' concept. In other words, it is the task of the predictive model to discriminate between 'attack' and 'normal' observations, where the specific attack types change over time. The original idea was to make the generator discriminate between attack types and attack subtypes. The attack types are the main categories of attacks and the attack subtypes are specific types that each fall into one of the main categories. For instance, an attack type is 'DoS' and one of its attack subtypes is 'smurf'. When the subtypes underlying the 'attack' concept change over time, this change can be within-type or between-type. On the one hand, within-type refers to a change of one set of attack subtypes to another set of attack subtypes, where all attack subtypes belong to the same attack type. On the other hand, between-type refers to a change of a set of attack subtypes that belong to attack type A to another set of attack subtypes that belong to attack type B. It was expected that within-type changes are smaller than between-type changes and therefore the classifier should have more difficulty with between-type changes than within-type changes. This expectation was build upon the expectation that attack subtypes that belong to the same attack type are more similar than attack subtypes that belong to different categories.

A.2.2 Results

Figure 7 shows the average accuracy of 200 trials of a Hoeffdingtree trained on the KDD'99 data stream for the no drift, sudden drift and gradual drift conditions. As can be seen, the expected pattern did not occur. There is no clear difference between the drop in performance for a within-type and a between-type drift. Also the time it takes to restore from the drift does not look very different. Figure 8 splits these results further into the different types of attack. As can be seen, 'DOS' attack subtypes are on average more easy to classify than 'Probe' attack subtypes. This difference in difficulty influences the strength or difficulty to deal with the drift. For example, a drift from 'DOS' (easy) to 'Probe' (difficult) leads to the biggest drop in performance.

As it turns out, one of the assumptions that this approach was build on, is not true, as can be seem from Figure 9. It is not true that within-type attack subtypes are closer together in feature space than between-type attack subtypes. It does not even hold that all instances belonging to one attack subtypes are more similar than the instances of other attack subtypes. Therefore, another approach was given a try, as explained in the next section.

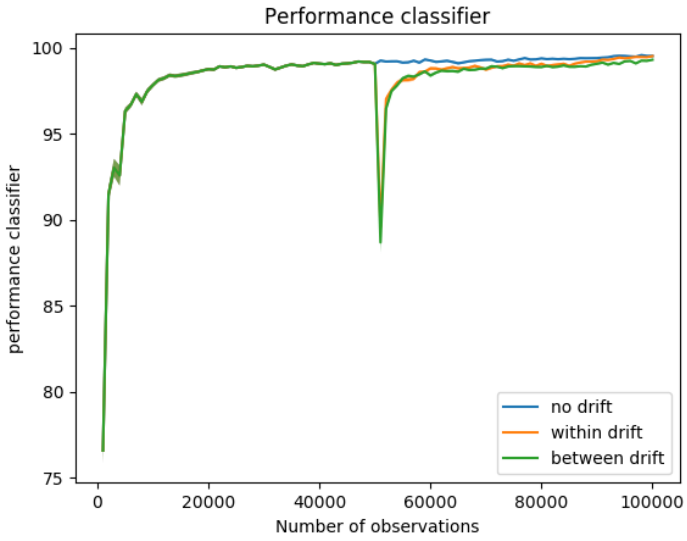

Figure 7: The average accuracy of 200 trials of a Hoeffdingtree trained on the KDD'99 data stream, for no drift, within-type sudden drift and between-type sudden drift. There is no clear difference between the drop in performance of within-type and between-type drift.

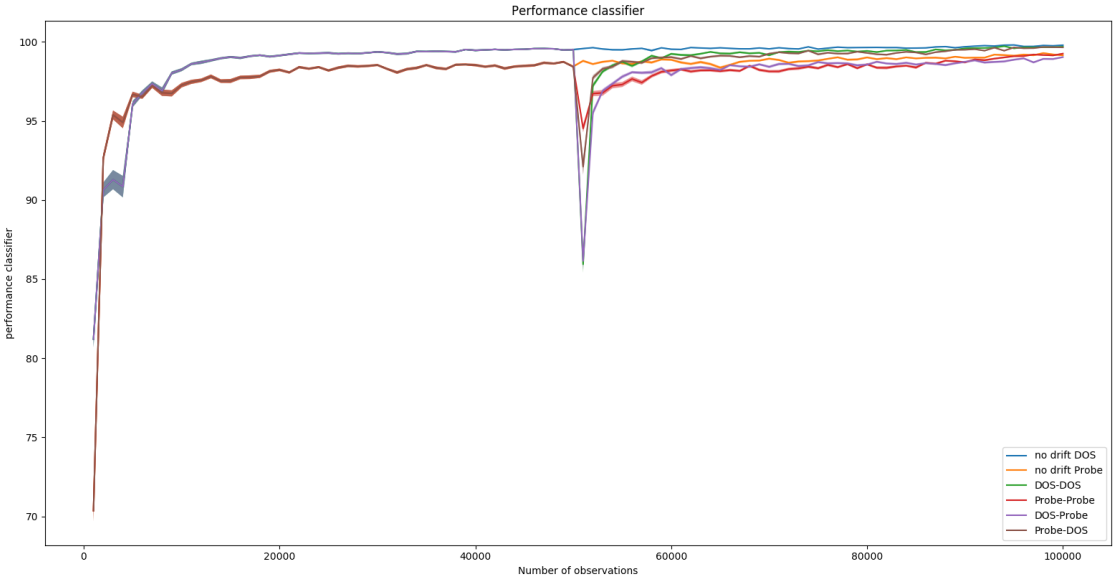

Figure 8: The same data as in Figure 7, but split for the different attack types. Drifting to 'Probe' subtypes shows a larger decrease in performance than drifting to 'Dos' subtypes.

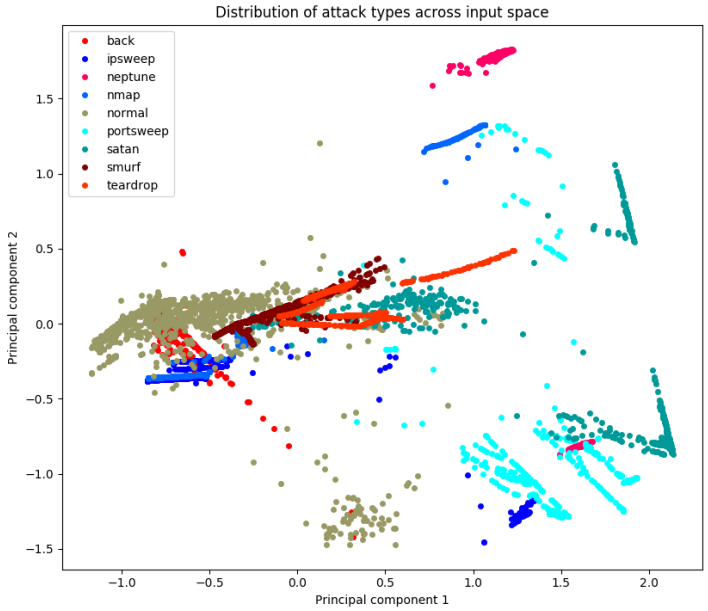

Figure 9: A scatter plot of the KDD'99 instances per type, after performing a Principal Component Analysis.

A.2.3 Alternative approach: Use clusters of data

An alternative approach was designed to overcome this last issue. The KDD'99 data was clustered to artificially create attack subtypes that are more similar within than between-type. K-means clustering was performed for different values of k. Figure 10 shows the clusters for $k = 3$, and Figure 11 shows the clusters for $k = 8$. Based on these cluster results, the clusters from Figure 11 were used as sub clusters and combined into main clusters based on the clusters in Figure 10. To be precise, sub clusters 1, 3, 4 and 7 together belong to main cluster A. Sub clusters 2 and 5 belong to main cluster B and sub clusters 0 and 7 belong to main cluster C.

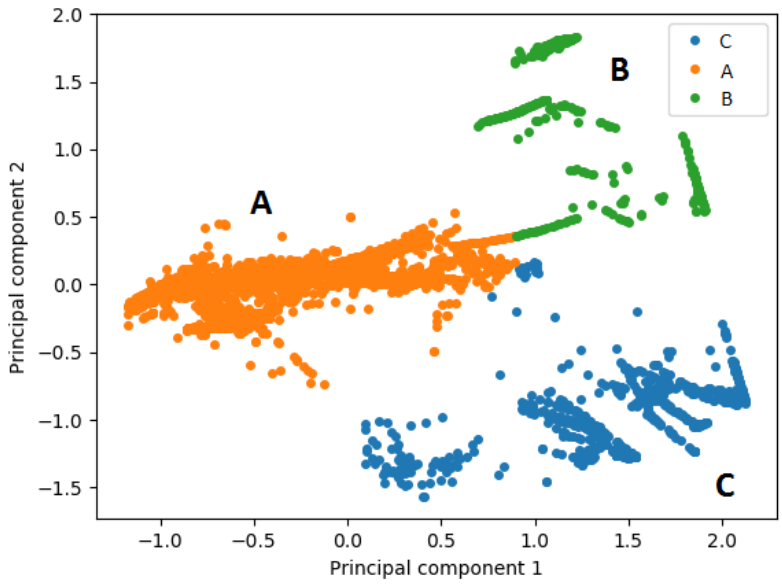

Figure 10: The clusters resulting from k-means clustering with k=3.

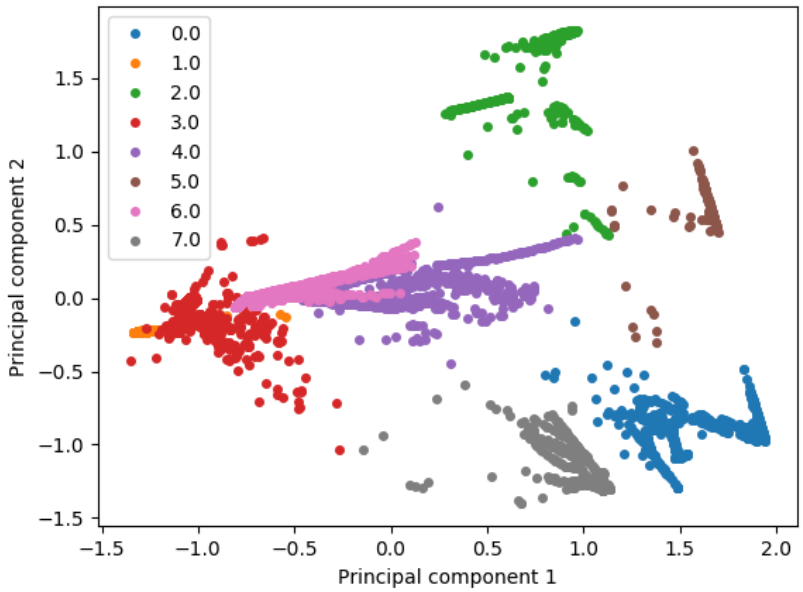

Figure 11: The clusters resulting from k-means clustering with k=8.

A.2.4 Results

These sub- and main clusters were then used to repeat the same experiment as previously. Initially, this showed promising results as can be seen in Figure 12. Between-cluster drift indeed seems more difficult than within-cluster drift. However, when splitting these results on main cluster (Figure 13), there is an alternative explanation for these results. Main cluster A is more difficult for the classifier than main clusters B and C. On average, this causes drift from A to any other cluster to be easy drift, and drift from B or C to any other cluster to be difficult drift. The within-cluster drift in Figure 12 shows the average of drift from A to A, B to B and C to C. The first drift is difficult and the latter two are easy. On average this gives quite an easy drift. The between-cluster drift in the same figure shows the average drift of A to other, B to other and C to other. The first one is easy and the latter two are difficult. On average this gives a more difficult drift. To conclude, the differences between within-cluster and between-cluster drift are not solely caused by the distance between clusters but also (and maybe mainly), by the relative difficulty of the clusters.

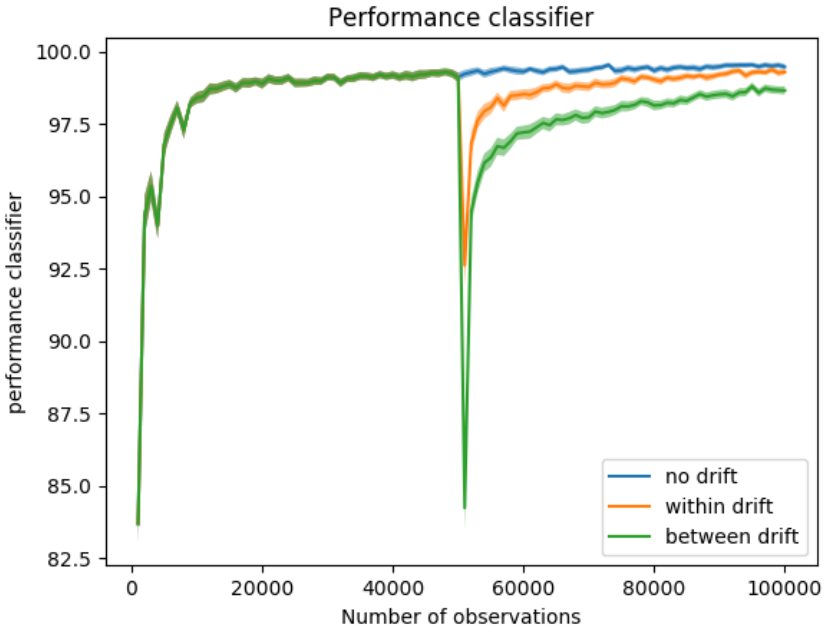

Figure 12: The average performance of 150 trials of a Hoeffdingtree classifier for a stream with no drift, sudden within-cluster drift and sudden between-cluster drift. Between-cluster drift was more difficult to deal with for the classifier than within-cluster drift.

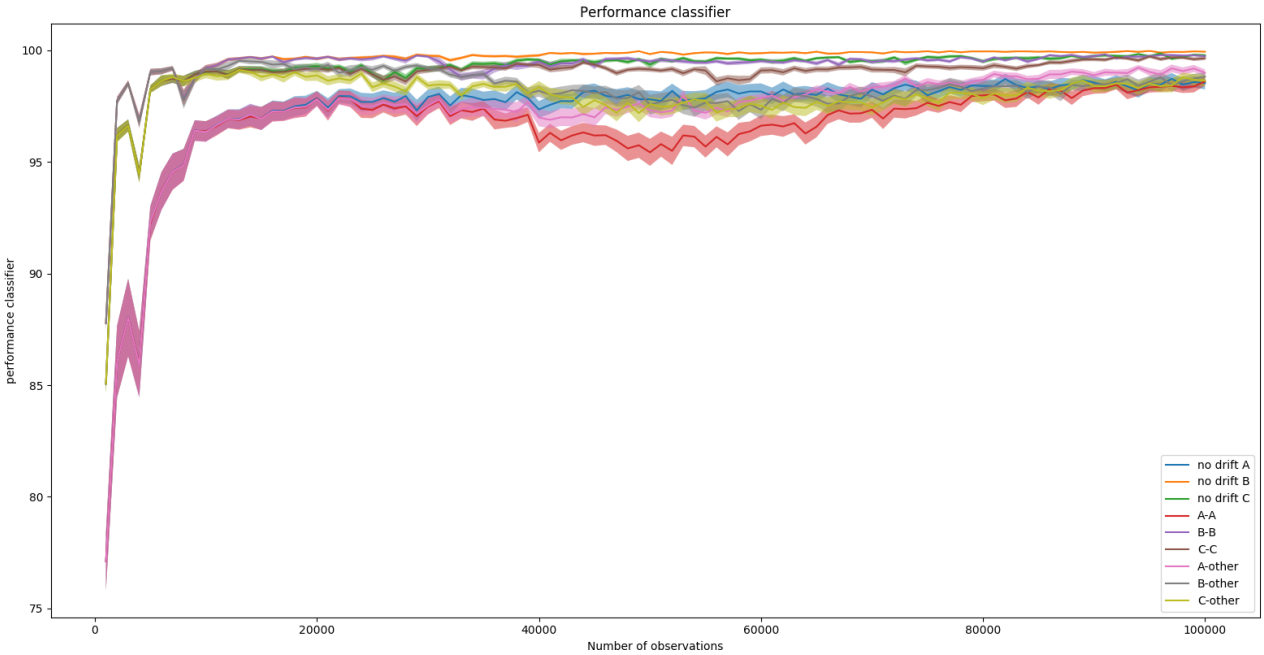

Figure 13: The same results as in Figure 12, but then split between main clusters. Main cluster A is more difficult to classify than the other clusters. Drift from cluster A (difficult) to other clusters (easy) is very easy to deal with, whereas drift from other clusters (easy) to cluster A (difficult) is more difficult to deal with.

A.2.5 Conclusion

To conclude, it is very difficult, if not impossible for the current data set, to disentangle the influence of classification difficulty of attack types or clusters and the influence of the distances between the attack types or clusters. Therefore, it was decided that the within- and between-type drift options were removed from the KDD'99 data stream generator.

B Pilot results

I have run multiple pilots to find the best experimental settings for my experiments. For each combination of pilot conditions, 100 trials were run and the average performance of these trials are plotted in the figures below. The shaded area shows the standard error. Each data stream consisted of 20.000 trials where drift happens, if present, at position 10.000. Sudden drift had a width of 1 whereas gradual drift had a width of 15.000. The attack ratio of the stream was set to 0.1 for all tests. The available budget differed, depending on the test. For some tests the budget did not matter and was set to 1, for other tests where the budget did matter it was set to 0.1.

It was found that the KDD'99 data stream generator generates a stream that is relatively easy to classify, with performance scores often above a scored AUC of 0.9 or even 0.95. This is not very surprising, because one of the issues that exist with the KDD'99 data set is that many instances are very easy to classify [31]. The stream is slightly more difficult when more attack types appear simultaneously in the stream, as is shown in Figure 14 [7]. Therefore, it was chosen to set the number of attack types to 4. When a drift happens, the stream drifts from 4 attack types to the 4 other attack types.

It was tested whether applying SMOTE to the training window before training the classifier would improve classifier performance. It turned out that SMOTE did not have any effect. Apparently, oversampling the minority class does not help in dealing with the class imbalanced data stream. Figure 15 seems to confirm this result. In this figure the average performance for the KDD'99 data stream is shown for different values of the attack ratio parameter. This parameter stands for the proportion of attack instances compared to normal instances in the stream. As can be seen from the figure, a balanced stream gives a slightly higher performance (median around $sAUC = 0.92$) than a stream where attack constitutes only 10 percent of the data (median around $sAUC = 0.90$), but this difference is not very big. Overall, it can be expected that the minority oversampling technique that is used in critical sampling will not improve the results a lot.

This finding is not very surprising, when looking at the scatter plot of the principal component values of the KDD'99 data points in Figure 9. For almost all attack types it holds that most or all instances are linearly separable from

[7] In this figure, each line represents the average performance of 100 no drift, 100 sudden drift and 100 gradual drift trials. This explains the small peak in the middle of the plot.

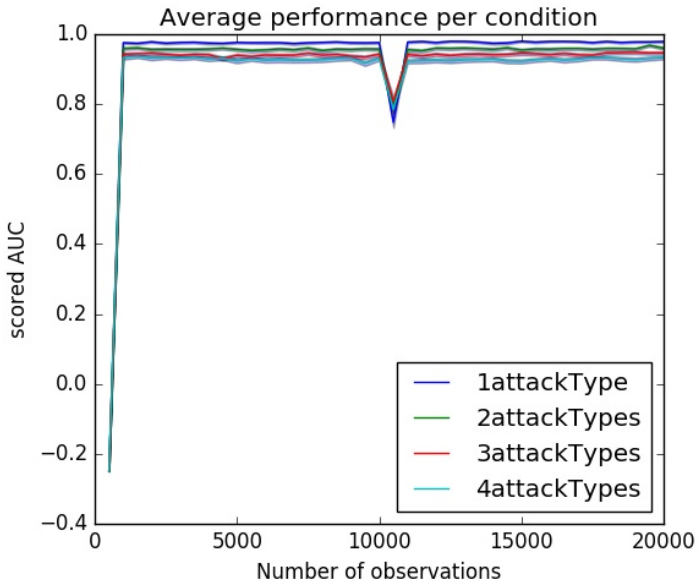

Figure 14: The influence of the number of attack types in the KDD'99 stream on classifier performance.

the normal instances. It is known that the influence of class imbalance on the classification performance of an SVM is very small when the minority class is linearly separable from the majority class [34]. Therefore, this same pattern can be expected (and is indeed observed) in the current experiment.

As can be seen in Figures 16 and 17, the value of the random ratio parameter for the randomized uncertainty sampling strategy does not have a great influence on classifier performance. In Figure 16, a random ratio of 0.5 seems to perform slightly better than other values. In Figure 17, a random ratio of 0.5 causes randomized uncertainty sampling to outperform the other strategies in all drift conditions, whereas for other values this pattern seems less consistent. However, it must be noted that these variations in performance could be caused by noise. Nevertheless, the selected value of the random ratio parameter is 0.5.

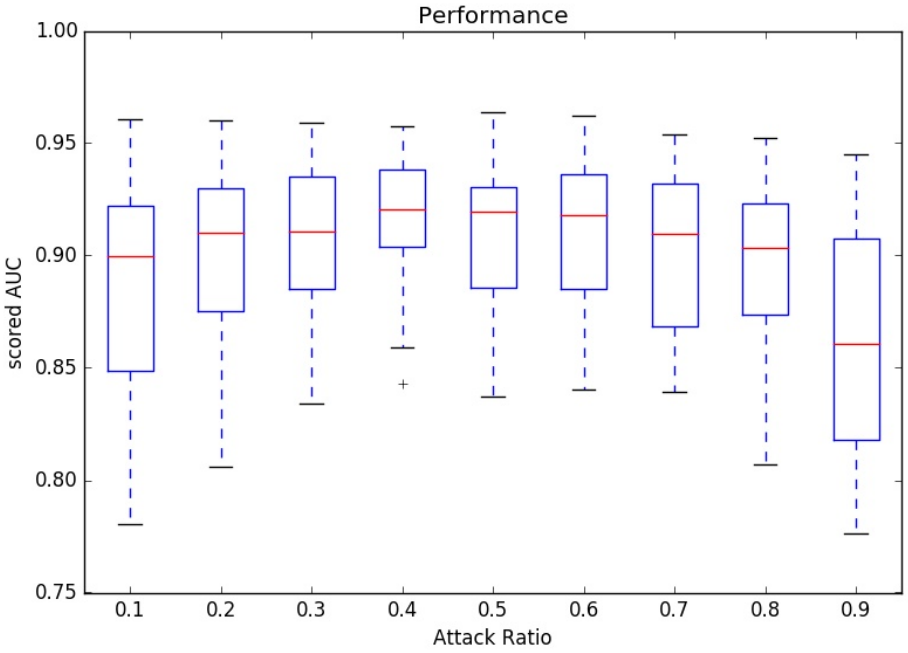

Figure 15: The average performance of the random sampling strategy for different attack ratio's in the KDD'99 data stream.

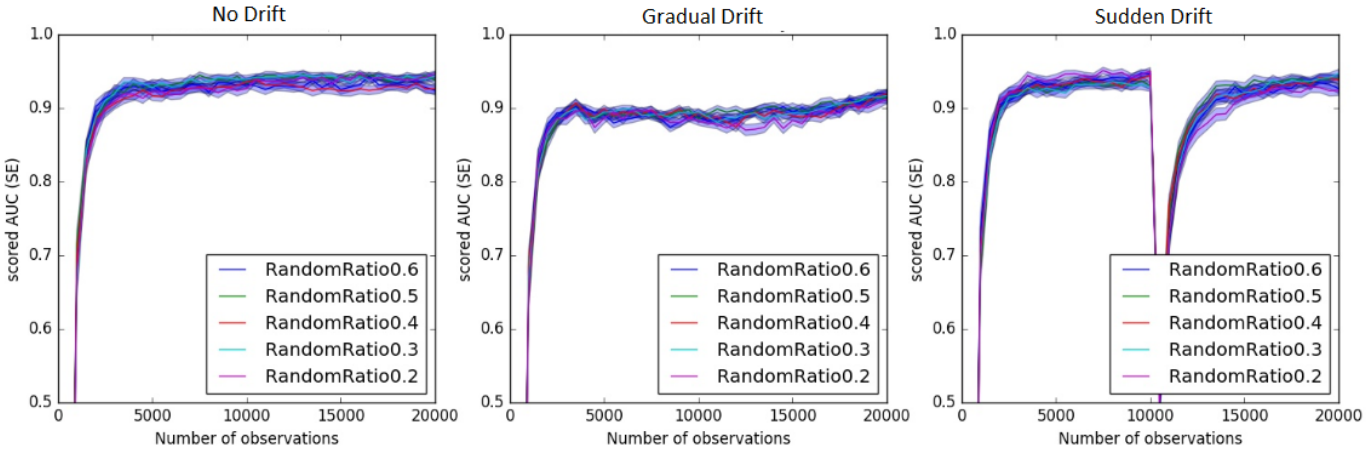

Figure 16: The influence of the random ratio parameter on the performance of randomized uncertainty sampling. As can be seen, there is no clear pattern visible.

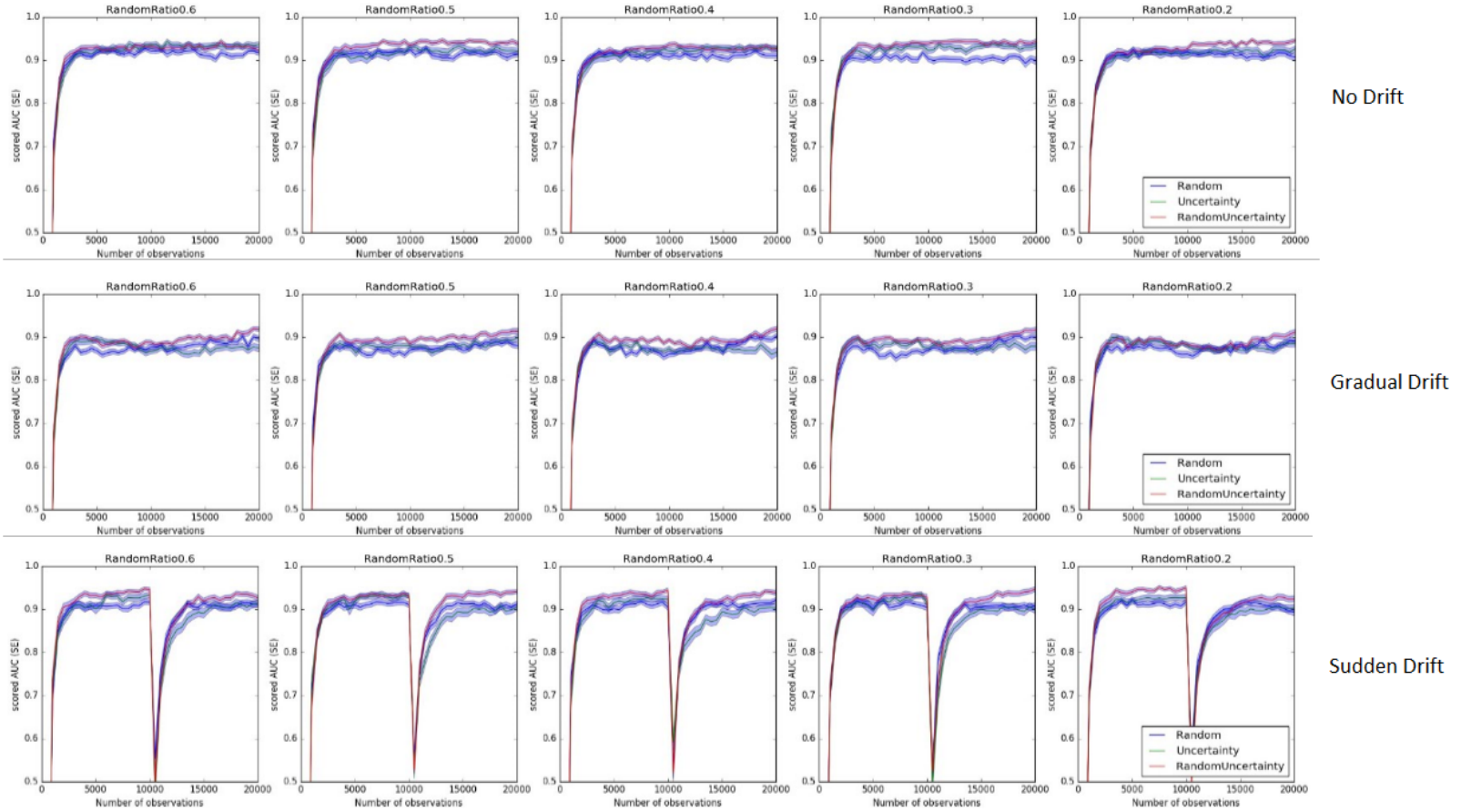

Figure 17: The influence of the random ratio parameter on the performance of randomized uncertainty sampling, as compared to the performance of uncertainty sampling and random sampling. When the random ratio is 0.5, randomized uncertainty sampling consistently outperforms the other two strategies, whereas this pattern is not consistent for other values of the random ratio. It must be noted that this observation could be caused by noise.

As can be seen from Figure 18, the minority ratio parameter of the critical sampling strategy has a great influence on the proportion of selected positive (minority) instances. The higher the minority ratio, the more positive minority instances are selected. This holds even for the condition where positive instances are very rare (attack ratio of 0.05), although the difference between a minority ratio of 0.4 and 0.5 is not very high in that case, because not that many more minority instances exist in the stream. Based on these figures, a minority ratio of 0.4 (green line) would yield a very balanced data stream when the attack ratio is 0.1, and also quite a balanced set when the attack ratio is 0.05. When the attack ratio is 0.15, a minority ratio of 0.3 or 0.2 would yield more balanced results.

As can be seen from Figure 19, the minority ratio parameter does not seem to have a great influence on classifier performance. There seems to be a trend in general that the higher minority ratio values give better performance, but this does not hold for all plots. Also, these results could be due to noise. Therefore, it was chosen to select a minority ratio parameter of 0.4, because this value gives the most balanced data stream based on Figure 18.

Also for the random ratio parameter of randomized critical sampling, it holds that it does not seem to influence the classifier performance much. Based on Figures 20 and 21 a general trend seems to be that an higher random ratio gives a better performance. This is not strange, because the higher the random ratio, the more similar randomized critical sampling will be to random sampling and random sampling gives an higher classifier performance than critical sampling. It was chosen to set the random ratio to 0.3, because this intuitively makes more sense. Randomized critical sampling is a combination of three strategies: exploring input space (random sampling), exploiting informative instances (uncertainty sampling) and oversampling the minority class. When setting the random ratio to 0.3 and the minority ratio to 0.4, each of these three strategies are used in quite equal proportions.

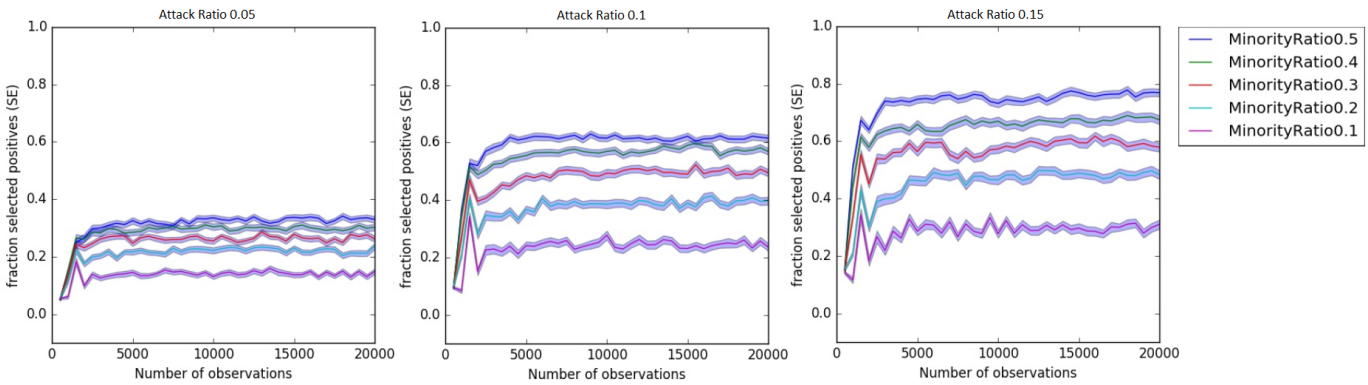

Figure 18: The influence of the minority ratio parameter on proportion selected positive instances. It can be seen that an higher minority ratio causes more minority instances to be selected.

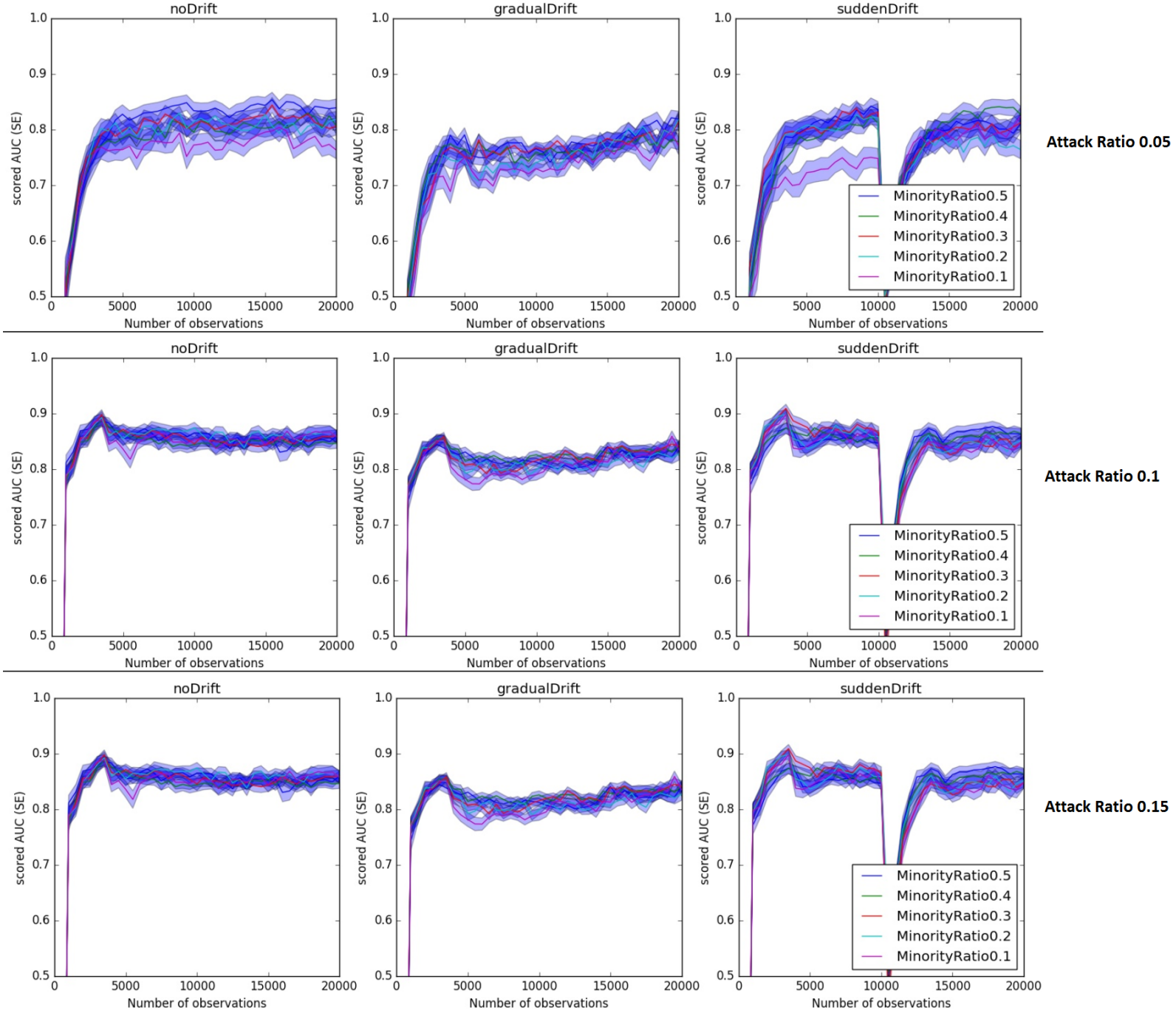

Figure 19: The influence of the minority ratio parameter on critical sampling performance. There seems to be a general trend that an higher minority ratio leads to better performance. Also, it must be noted that these results could be due to noise.

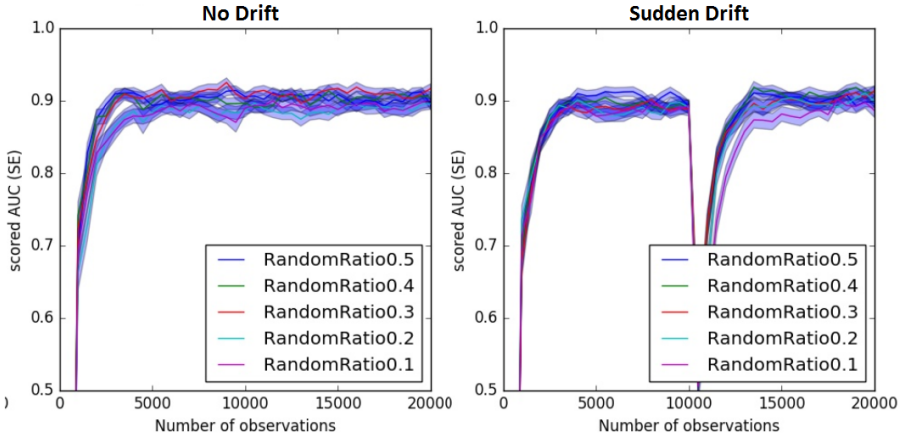

Figure 20: The influence of the random ratio parameter on randomized critical sampling performance. There seems to be a trend that a random ratio of 0.5 gives the highest performance, but this trend is not very strong.

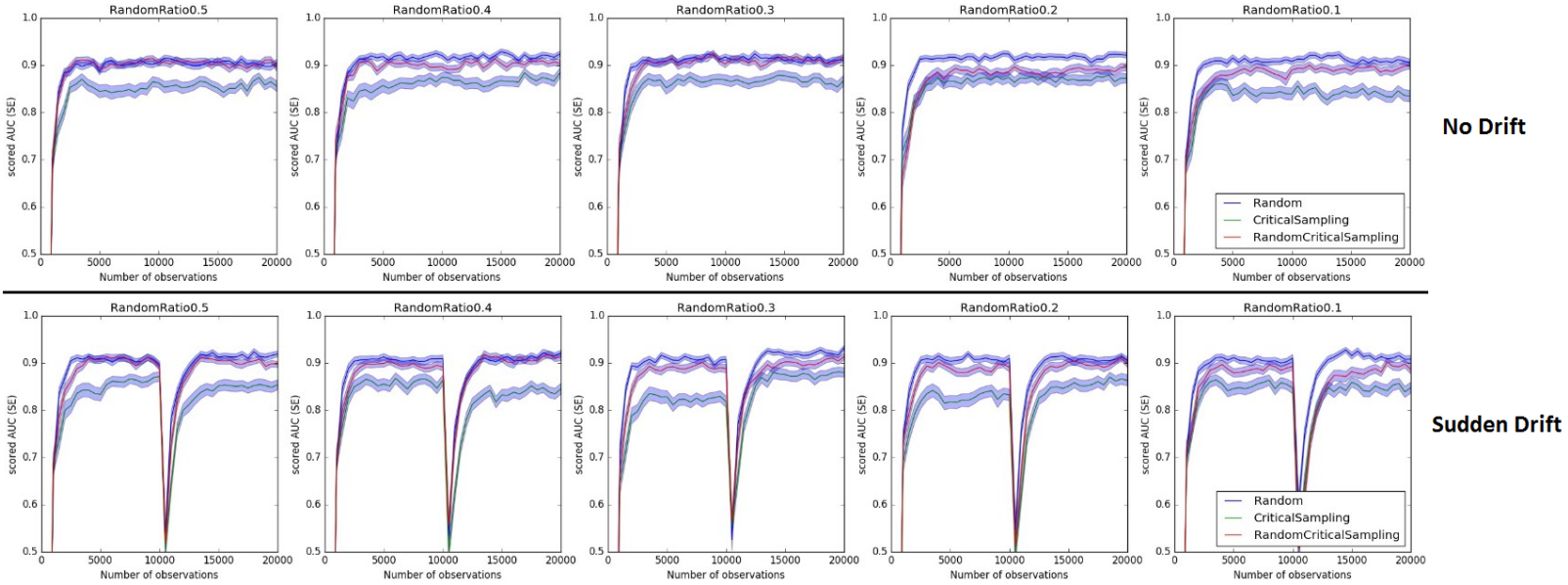

Figure 21: The influence of the random ratio parameter on randomized critical sampling performance. The general trend seems to be that an higher random ratio allows randomized critical sampling to perform more similar to the random sampling baseline.

C Overview of results

Figures 22 and 23 give an overview of all results obtained in the experiment, in the conditions where the attackratio was 0.05 and 0.1 respectively. As can be seen, the pattern of the performance of the different strategies is consistent across conditions. In almost all conditions, randomized uncertainty sampling performs best, followed by uncertainty sampling and random sampling. In most conditions, uncertainty sampling performs slightly better than random sampling, but this does not hold for all conditions. In all conditions, randomized critical sampling performs slightly worse than the random sampling baseline. Also for all conditions, critical sampling performs much worse than baseline.

One observation that stands out from these box plots is that the variance is much larger in the no drift condition than in the drift conditions. This can be explained by the number of attack types that is present in the streams. In the no drift condition, there are four attack types, but in the drift conditions, there are in total eight attack types, drifting from four to the other four types. During the pilot studies it was found that one attack type was more difficult to classify than the other seven[8]. In the no drift streams, half of the trials do not contain this difficult attack type and the other half does, leading to much variance. In the drifting streams, all trials contain this attack type, either in the first or the second sub stream, giving less variance in the average performance across the whole stream.

[8]This result was not included in the appendix.

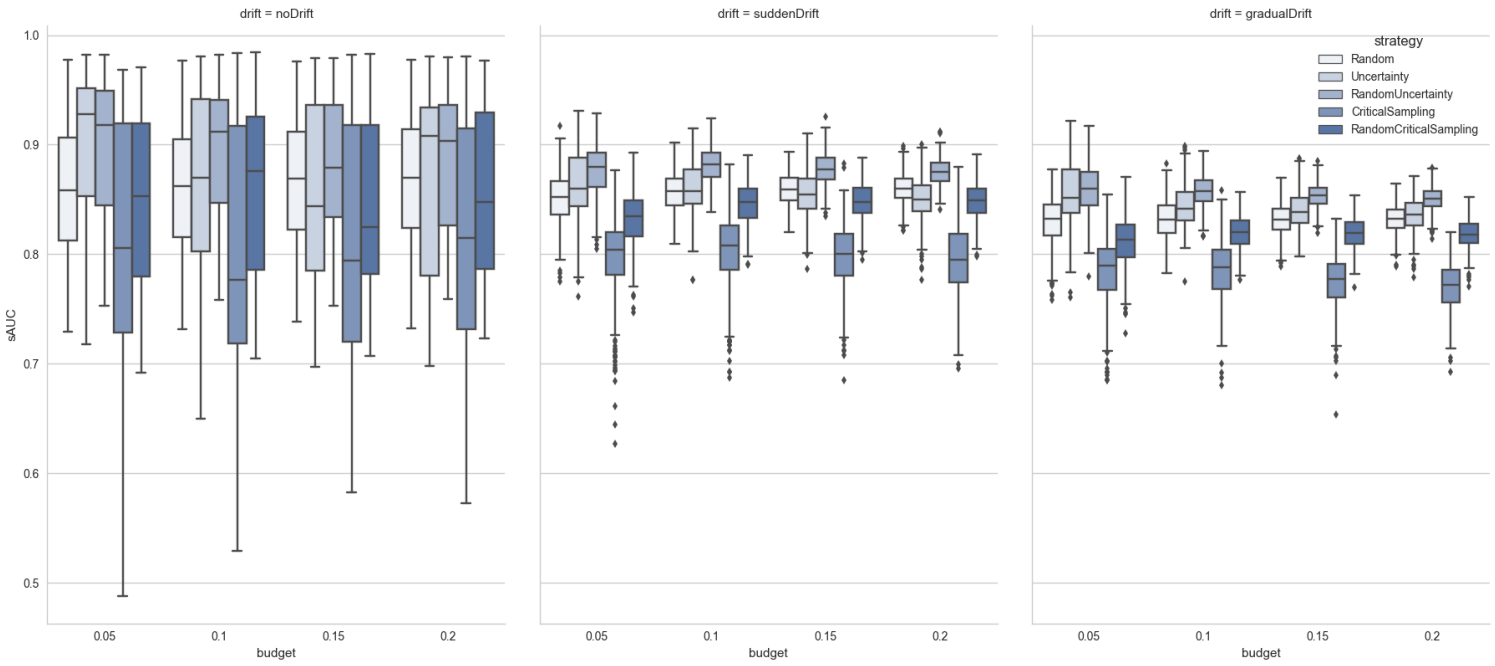

Figure 22: An overview of the average performance across all conditions where the attackratio was 0.05. It can be seen that the strategies perform quite consistently across conditions: randomized uncertainty sampling performs best, followed by uncertainty sampling, random sampling and randomized critical sampling. Critical sampling performs far worse than baseline.

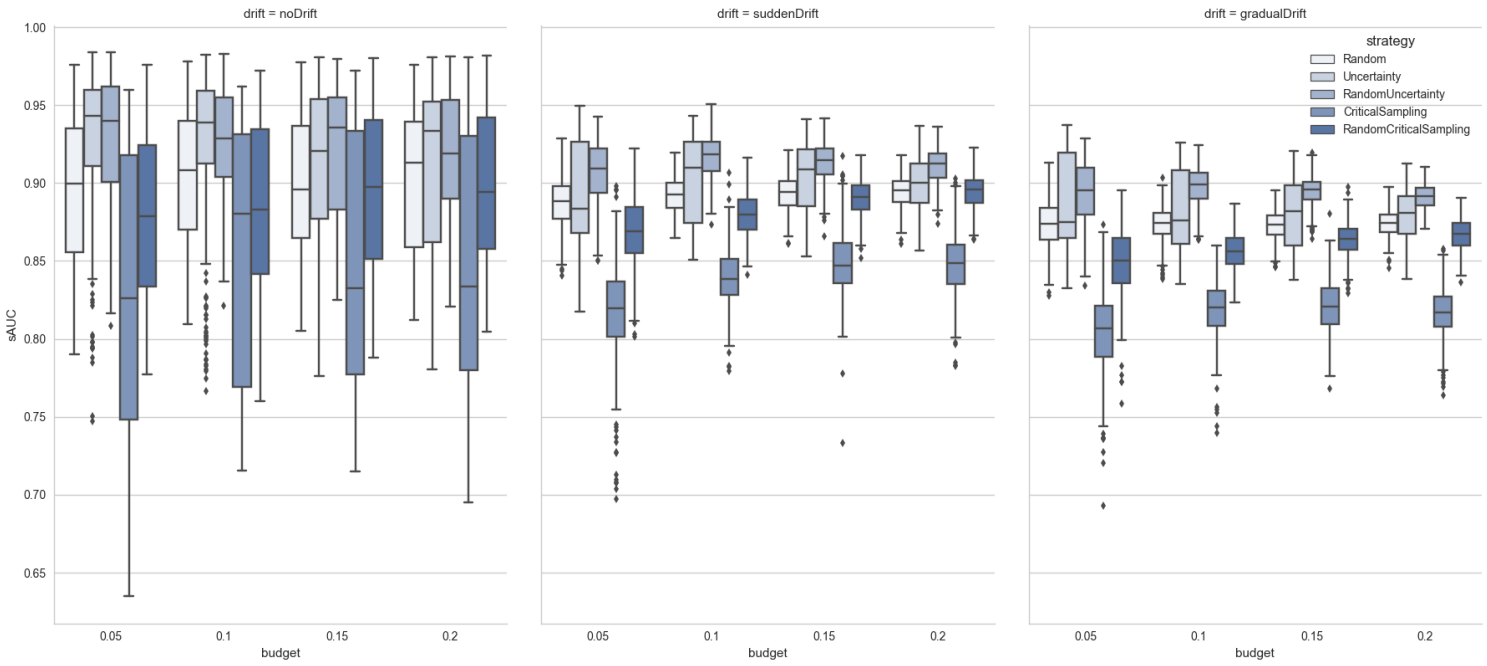

Figure 23: An overview of the average performance across all conditions where the attackratio was 0.1. It can be seen that the strategies perform quite consistently across conditions: randomized uncertainty sampling performs best, followed by uncertainty sampling, random sampling and randomized critical sampling. Critical sampling performs far worse than baseline.