FACULTY
OF MATHEMATICS
AND PHYSICS
Charles University

MASTER THESIS

Martin Procházka

Classification in data streams with abrupt concept drift in a subset of features

Department of Algebra

Supervisor of the ster thesis: doc. Mgr. Viliam Lisý, MSc., Ph.D.

Study programme: Mathematics for Information Technologies

Prague 2024

I am very grateful to my supervisor, doc. Mgr. Viliam Lisý, MSc., Ph.D., for his guidance, the opportunity to consult, and his invaluable feedback on this thesis. I am also profoundly grateful to my family for their unwavering support and encouragement throughout my studies.

Title: Classification in data streams
with abrupt concept drift in a subset
of features

Author: Martin Procházka

Department: Department of Algebra

Supervisor: doc. Mgr. Viliam Lisý, MSc., Ph.D., Computer Science, FEE, Czech Technical University in Prague

Abstract: Malware detection is a crucial aspect of cybersecurity, presenting several challenges, particularly in data stream scenarios that experience strong concept drift and label delay. The concept drift is characterized by the presence of highly influential yet rapidly changing features, such as specific filenames or mutexes, alongside stable features, such as connection types or monetization methods, which remain relatively consistent over time. In this thesis, we formalize this scenario and further exploit the hypothesis that the adaptive removal of severely drifting subsets of features may have a great impact on procedure performance. We indeed demonstrate that current methods exhibit shortcomings connected with these features, especially during short periods following the arrival of a new concept. To validate the hypothesis of performance improvement through adaptive feature elimination, we propose two solutions: one based on Hellinger distance concept drift detection and the other on an incremental Gaussian Mixture Model algorithm. We evaluate both approaches using real-life data and our synthetic dataset, showing significant improvements on the synthetic dataset and promising results on real-life data. Additionally, we provide a comprehensive explanation of the techniques employed in the thesis.

Keywords: malware detection, concept drift, data stream, concept drift detection, Gaussian Mixture Models

Contents

Introduction

Many prediction computation models are based on an assumption of stable data distribution, a premise that often does not hold in real-world scenarios. The presented thesis is inspired by detection of malicious software (malware), which is software intentionally created to disrupt its target, usually a user, computer, or computer network. In this context, the stable data assumption is highly inappropriate due to constant evolution and development of malware by attackers. This leads to a dynamic environment of data streams, associated with the term *concept drift*, referring to the general change of data over time (Lu et al. [2019]). An important aspect of the dynamic data distribution is that we need to evolve our model constantly. Numerous data stream algorithms aim to address this challenge (Wankhade et al. [2020], Gomes et al. [2017], Ceschin et al. [2022]). One such algorithm is the *Dynamic Weighted Majority (DWM)*, introduced by Kolter (Kolter and Maloof [2007]), which is a robust procedure based on an evolving ensemble of experts. This thesis will demonstrate that the DWM algorithm has certain limitations and, in specific scenarios, may suffer from unnecessary drops in accuracy.

The scenario formulated and presented in Chapter 2 is based on malware detection. Samples from this subject have some specific properties, which we discuss in Section 2.1. One of the specifics of malware detection is quite usual presence of features that enable quick and valid decisions about the harmfulness of software. One possible example of such features is the mutex used by a well-known malware family. However, these features tend to change rapidly, sometimes with each new iteration of the malware. This change often results in a deterioration of detection model performance, as the model becomes overly reliant on these highly relevant, strongly decisive features, which, once altered, lead to inaccurate predictions. On the other hand, it is also quite usual that there are aspects of malware that are difficult or even impossible to change as type of connection, used protocol, or way of monetization. This motivates definitions of *severe concept drift* and *relevant stable information* presented in Definition 8. This scenario highlights the existence of feature subsets that provide relatively good decisive information, which machine learning algorithms often overlook due to excessive focus on highly informative, strongly drifting features. This focus, after drift, leads to severe decreases in prediction accuracy. In this thesis, we formalize this scenario.

We will develop two sub-procedures to address the above scenario. These sub-procedures will complement the underlying algorithm, in our case, the DWM algorithm, by providing information about drift in feature vector subsets. The first procedure is based on Hellinger distance concept drift detection (Ditzler and Polikar [2011]), and the second on the incremental Gaussian Mixture Models algorithm (Engel and Heinen [2010]). These techniques are then evaluated and compared with the usual DWM algorithm using both synthetic and real-life data (Arp et al. [2014]).

The organization of the thesis is as follows: Chapter 1 provides the background of concepts used in the thesis, including an overview of concept drift and key aspects of related work and our solutions. It explains the DWM algorithm and its setting, basic Hellinger concept drift detection together with its important principles, and derives the Fast incremental Gaussian Mixture Model algorithm. In Chapter 2, we present the motivation, formalization of the scenario and main objectives together with our definitions and assumptions. Chapter 3 discusses possible shortcomings of DWM algorithm in Subsection 3.2.1 and outlines our general solutions in Section 3.3. Chapter 4 presents our proposed solutions. Section 4.2 describes our sub-procedure algorithm based on Hellinger distance. Section 4.3 introduces the sub-procedure based on the GMM algorithm and discusses possible parameter settings. Chapter 5 presents experiments and evaluations made in this thesis. Chapter 6 offers a brief discussion of potential future work.

The main focus of this thesis lies in formalizing the described scenario and developing methods that extend the general procedure (using the DWM algorithm), which are capable of improving performance in this setting, as tested on both synthetic and real-life data.

1. Background

In this chapter, we describe various techniques that form the foundation of our solutions and formalizations. We present existing procedures, which we supplement with details, pictures, and examples.

Despite our efforts to maintain continuity, the relevant background information spans multiple scientific fields. Therefore, the sections introduced here primarily serve to offer in-depth explanatory information, which will be referenced in other chapters.

The chapter structure is as follows. Section 1.1 defines and widely discusses the term *concept drift* as the underlying principle of non-stationary data distribution. Section 1.2 focuses on the Dynamic Weighted Majority algorithm and its settings. Section 1.3 presents the background for our solution from Section 4.2 based on Hellinger concept drift detection. Section 1.4, Section 1.5 and Section 1.6 describe ground principles of our proposed solution presented in Section 4.3, which is based on Gaussian Mixture Models algorithm.

1.1 Concept drift

Many predictive computational models operate under the assumption of a stable data distribution, which often contrasts with real-world data. In this section, we present the general definitions, notations, and basic properties of learning under the assumption of a non-stationary data distribution, a phenomenon known as concept drift.

The term concept drift is crucial for this thesis, so we will begin by defining it in Subsection 1.1.1. Then we will focus on a wider understanding of the term. First, we will show some illustrations of possible drifts in Subsection 1.1.2. This will better contextualize our future approach to tracking changes in feature vector distribution. In Subsection 1.1.3, we will examine the location of the drift. The most important intuition given in this subsection is that it can be natural that there exists non-trivial information that is stable during the drift and also gives hints as to why we later chose to track the changes at feature subsets level.

1.1.1 General definitions

The general objective of this thesis is to study dynamic environments. All presented algorithms together with problem formalization from Chapter 2 are built in this setting. Therefore, we consider the best to start with a definition of the dataset change during time, which is called concept drift.

In the thesis, we will often refer to a model. This term can generally refer to a wide range of methods and approaches in machine learning. To stay rigorous we present the definition of this term in Definition 1.

Definition 1. *We define model "M" as a function $M(X, \theta)$ together with parameters θ, which for feature vector $X \in \mathbb{R}^d$ gives label $y \in \mathcal{Y}$, where $\mathcal{Y}$ is label space.*

As previously mentioned, one of the primary focuses of this thesis is the study of non-stationary data distribution, commonly referred to as concept drift, which is defined in Definition 2, which follows [Lu et al., 2019, Subsection 2.1]. According to this definition, concept drift is generally understood as a change in the joint probability distribution of the feature vector and the corresponding label over time.

Definition 2. *Given time period* $[0, t_1]$, $0 < t_0 < t_1 \in \mathbb{N}$, *sets of time steps* $S_{0,t_0} = \{e_0, \dots, e_{t_0}\}$, $S_{t_0+1,t_1} = \{e_{t_0+1}, \dots, e_{t_1}\}$, *where* $e_i \subseteq \mathbb{R}^d \times \mathcal{Y}$ *is a sample arrived in time step* i. *Each sample* e_i *consists of feature vector* $X_i \in \mathbb{R}^d$ *and label* $y_i \in \mathcal{Y}$, *for* $\mathcal{Y}$ *be a label space.* S_{0,t_0} *follows a certain distribution* $P_{0,t_0}(X, y)$, *same* S_{t_0+1,t_1} *follows a certain distribution* $P_{t_0+1,t_1}(X, y)$. *Concept drift in time* t_0 *is property* $P_{0,t_0}(X, y) \neq P_{t_0+1,t_1}(X, y)$.

Since Definition 2 is quite complex, we give some more illustration behind it in Subsection 1.1.2. Also, it is usual to divide concept drift into types and assume specific properties. This we do in Section 2.2.

1.1.2 Probabilistic concept drift illustration

The definition of concept drift presented as Definition 2 is quite complex. Therefore, we consider it appropriate to give some intuition behind it. In this section, we present a basic probabilistic types of concept drift which provide us with a better understanding of the problem and give us some intuition behind the term concept drift. Section is based on information from [Lu et al., 2019, Subsections 2.1] and [Bayram et al., 2022, Sections 4].

As seen in Definition 2 the concept drift is generally defined as the change in joint probability of feature vectors X and labels y in time t_0, $P_{t_0}(X, y) \neq P_{t_0+1}(X, y)$. The joint probability can be easily decomposed to

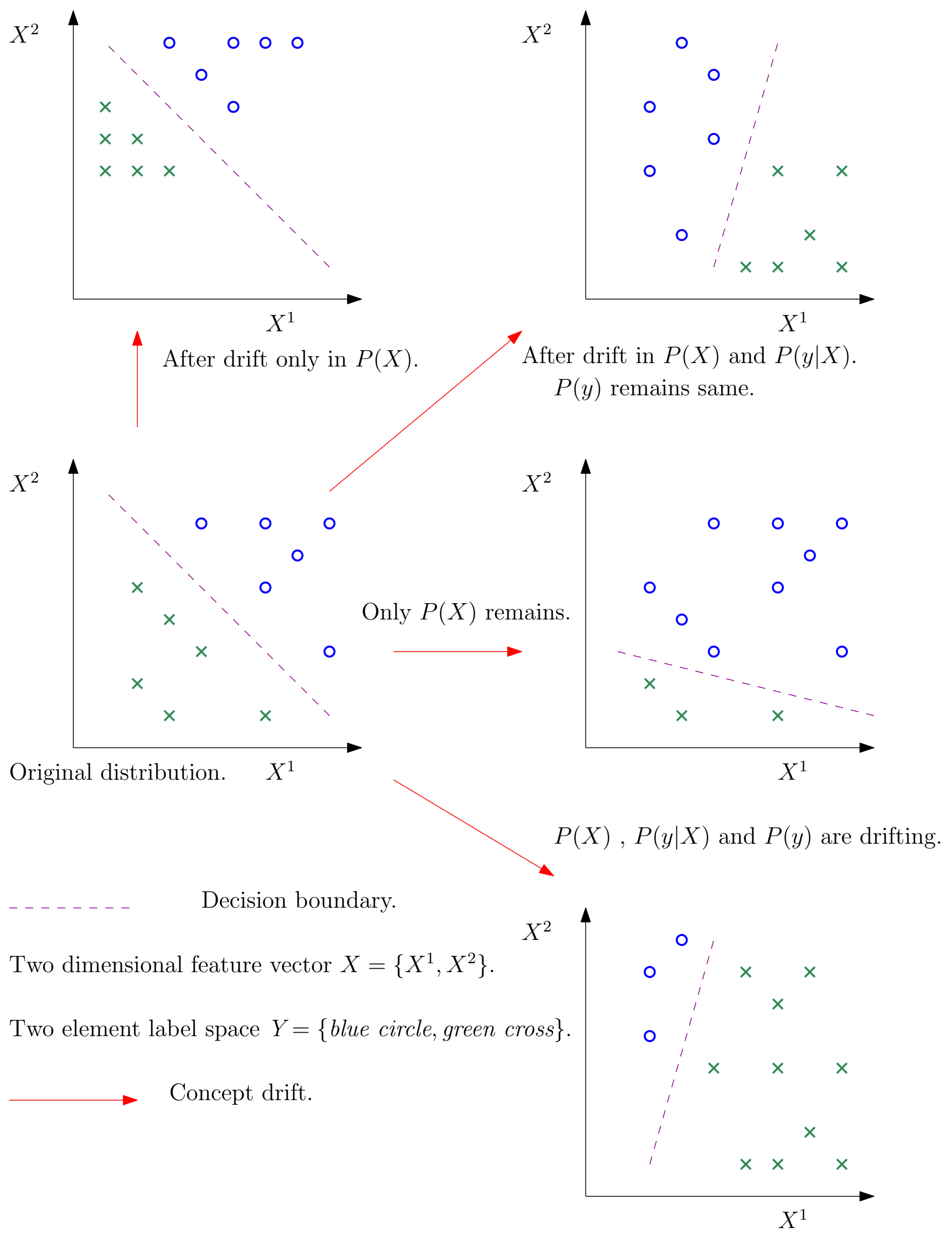
$$P_{t_0}(X, y) = P_{t_0}(X|y)P_{t_0}(y) = P_{t_0}(y|X)P_{t_0}(X). \tag{1.1}$$

First observation we can make is that a change in each of the four parts of decomposition, that means label posterior probability distribution $P_{t_0}(y|X)$, feature vectors distribution $P_{t_0}(X)$, label-conditional density distribution $P_{t_0}(X|y)$ and prior label distribution $P_{t_0}(y)$, is connected with change in at least one another of these distributions. This can be seen using the Bayesian rule from which it follows that if

$$P_{t_0}(X|y) \neq P_{t_0+1}(X|y) \text{ then } \frac{P_{t_0}(y|X)P_{t_0}(X)}{P_{t_0}(y)} \neq \frac{P_{t_0+1}(y|X)P_{t_0+1}(X)}{P_{t_0+1}(y)}.$$

We showed the dependency for $P_{t_0}(X|y)$, approach for the three rest distributions would be similar. Based on the implication we illustrate basic concept drift on changing of $P_{t_0}(y|X), P_{t_0}(X)$ and $P_{t_0}(y)$. In Figure 1.1, which is mostly based on [Lu et al., 2019, Figure 3.], we demonstrate the changes using two feature scenarios and some of the possible probability drifts.

In this subsection, we provided some intuition behind Definition 2, primarily using Figure 1.1, which depicts possible drift scenarios based on the decomposition of joint probability using Bayes' rule. An important aspect of concept drift,



X^2
X^1
After drift only in $P(X)$.
X^2
X^1
After drift in $P(X)$ and $P(y|X)$.
$P(y)$ remains same.
X^2
Original distribution.
X^1
Only $P(X)$ remains.
X^2
X^1
$P(X)$, $P(y|X)$ and $P(y)$ are drifting.
Decision boundary.
Two dimensional feature vector $X = \{X^1, X^2\}$.
Two element label space $Y = \{blue\ circle, green\ cross\}$.
Concept drift.
X^2
X^1

Figure 1.1: Illustration of the possible data distribution drift if specific probability aspects changes.

as illustrated in the figure, is its potential complexity. The figure also presents the intuition behind the main approaches for tracking concept drift, which involve studying changes in the distribution of samples and monitoring the decision boundary as an aspect fundamentally linked to model accuracy. All presented approaches will utilize one or a combination of these two ideas.

1.1.3 Location of drift

In this subsection, we provide some intuition behind our approach. We will present an example that demonstrates how, even after a significant general concept drift, a non-negligible amount of unaffected information can remain. Identifying this stable information is crucial to the algorithms presented in this thesis. Specifically, we achieve this by identifying drifted feature subsets. The rationale behind this approach is elaborated upon in the rest of this subsection.

Our approach is primarily based on [Lu et al., 2019, Subsections 4.3], where the authors analyse the location of the drift by examining whether arriving samples deviate from the current distribution. Throughout this thesis, we focus on feature subsets rather than entire feature vectors. This choice is due to the specifics of our problem presented in Section 2.3, which generally involves the highly variable influence of drift on different feature subsets. In this subsection, we provide intuition for the general approach of studying concept drift by concentrating on its localization.

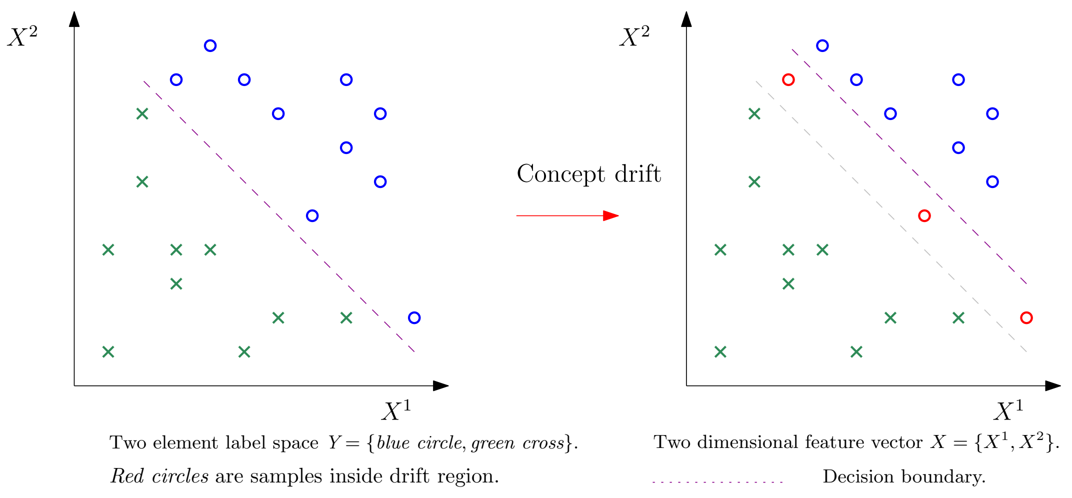

X^2
X^1
Concept drift
X^2
X^1
Two element label space $Y = \{$*blue circle*, *green cross*$\}$.
Red circles are samples inside drift region.
Two dimensional feature vector $X = \{X^1, X^2\}$.
Decision boundary.

Figure 1.2: Illustration of location of drift region on the task of rotating hyperplane.

Similarly as [Lu et al., 2019, Fig 12], we present Figure 1.2. In the figure, we can see that concept drift can naturally create some segments of the dataset that do not drift and some for which the drift is severe. These segments of data, where the drift occurs are called *drift regions*. In Figure 1.2 we can see the problem of rotating hyperplane. A decision boundary for original concept is $X^1 + X^2 = 11$ and for drifted concept is $X^1 + X^2 = 13$, where $X^1, X^2 \in \mathbb{N}$, $X^1, X^2 \leq 10$ are randomly sampled, which means that distribution does not drift. Therefore, we represent the distribution both before and after the drift by the same sampled points. The change of decision boundary reveals the drift region to be set by

$11 \leq X^1 + X^2 < 13$. The rest of the samples, ones out of the drift region, are completely unaffected by the drift.

For a better illustration of how the process can be simplified by considering single features, we give Figure 1.3, where we can see the drift of a two-dimensional vector $X = \{X^1, X^2\}$. Considering the drift of the whole feature vector, due to severe drift in X^2 a model learned on original data can struggle with a right prediction on the drifted ones. On the other hand, when observing single features we see that the severe drift is connected with feature X^2 and feature X^1 drifts only slightly. This could be valuable information when constructing a model. For

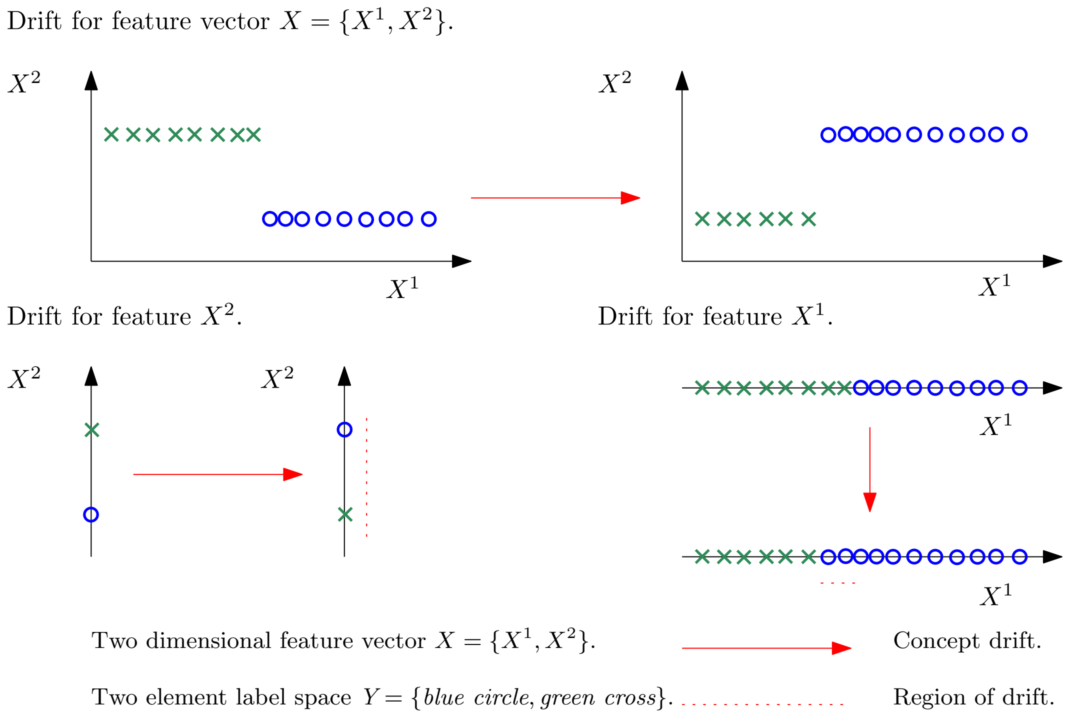

Drift for feature vector $X = \{X^1, X^2\}$.
X^2
X^1
Drift for feature X^2.
Drift for feature X^1.
Two dimensional feature vector $X = \{X^1, X^2\}$.
Concept drift.
Two element label space $Y = \{blue\ circle, green\ cross\}$.
Region of drift.

Figure 1.3: Illustration of the concept drift of two dimensional feature vector $X = \{X^1, X^2\}$. Upper scheme is the drift of entire feature vector X. The lower ones are scheme of drift in single features of feature vector X.

a demonstration of basic intuition behind possible dealing with the concept drift by removing the effect of the most drifting features, we present Lemma 1, where we use the strong assumption of mutually independent features. Proof of this lemma is presented in Attachment A.2.

Lemma 1. *Let $X = \{X^1, \dots, X^d\}$ be a feature vector, where $d \in \mathbb{N}$ and all features in the vector are mutually independent, conditional on the label $y \in \mathcal{Y}$. That means $P(X^i | X^1, \dots, X^{i-1}, y) = P(X^i | y)$. Then it holds that $P(X^1, \dots, X^d, y) = P(y)^{1-d} \prod_{i=1}^{d} P(X^i, y)$.*

In generally known Lemma 1 we showed that using certain assumptions, joint probability can be estimated by single margin probabilities. Thus when considering change in joint probability it can be beneficial to lower the effect of most drifting features in the feature vector. Note also, that assumptions and ideas from the last lemma are highly connected with the approach called Naive Bayes classifier which we present in Subsection 1.2.2.

1.2 Background of related work

In this section, we describe methods, which we will discuss later in Chapter 3 as related to our problem (Section 2.3) and which will be also used as a base component of our overall solution outlined in Section 3.3. The main objective of this section is Dynamic Weighted Majority (DWM) algorithm as the technique we will use for demonstration of the usefulness of our procedure. But we will also derive the Gaussian Naive Bayes classifier and present some of its properties since we will use it as part of the DWM algorithm.

.

1.2.1 Dynamic Weighted Majority

This subsection is based on Kolter and Maloof [2007] and we will describe the Dynamic Weighted Majority algorithm, in the following text just DWM, together with its possible settings.

DWM algorithm is based on ensemble learning developed for data stream scenarios. The properties of this algorithm stand primarily in its robustness and ability to quickly adapt to a completely different concept. Therefore, we choose this algorithm as the suitable algorithm on which we can show the advantages of our solutions presented in Section 4.2 and Section 4.3.

We describe the DMW algorithm similarly as is presented in [Kolter and Maloof, 2007, Section 3]. The DWM algorithm is well-known, robust, and flexible. Therefore, we choose it as our baseline. The process described in Algorithm 1 is based on an ensemble of experts (models), where each has its variable weight. These experts are removed and added according to their prediction performance.

In Algorithm 1 we can see that we start with the single expert of weight 1. Then we proceed by looping over all samples. Every sample is classified by each expert we have at this moment. If the classification by an expert is wrong and at the same time we meet updating period ($i \bmod p = 0$) then we decrease the weight of the expert by decreasing factor β. The global ensemble prediction is made by summing up all the weighted classifications and the largest of them is selected. Finally, if the updating period is met we normalize the weights so that the greatest of them is 1, then we use a threshold for removing outdated experts and if the global prediction is wrong we add a new expert with a weight of 1. In the end, we provide the sample to experts for training.

Note that the experts need not be the same model type. The algorithm is flexible and if the scheme of choosing the suitable expert is provided, models used as experts can shift accordingly. It is also possible to improve the procedure against specific concept drift types (see Subsection 2.2.2). We can keep discarded experts and check their performance on newly arrived concepts and possibly include them back into the ensemble. This is especially connected with reoccurring concept drift.

The properties of the DWM algorithm are connected to the type of used expert. In this thesis, we choose the Gaussian Naive Bayes classifier, which we describe in the next subsection.

Algorithm 1 Dynamic Weighted Majority ([Kolter and Maloof, 2007, Figure 1])

Notation

$\{(e_i, w_i)\}_{i=0}^{m}$ - Set of experts e_i and according weights w_i.

Λ - Global prediction, $\Lambda \in \mathcal{Y}$.

λ - Local prediction, $\lambda \in \mathcal{Y}$.

σ - Sums of local predictions, $\sigma \in \mathbb{R}^{|\mathcal{Y}|}$.

Inputs

$\{(X_i, y_i)\}_{i=0}^{n}$ - Samples, feature vector $X \in \mathbb{R}^d$ and according label $y \in \mathcal{Y}$, where $\mathcal{Y} = \{1 \ldots, c\}$ for $c \in \mathbb{N}$.

β - Factor of decreasing weights, where $0 \leq \beta < 1$.

θ - Removing expert threshold.

p - Period of ensemble and weights update, $p \in \mathbb{N}$.

```
$m \leftarrow 1$
$e_m \leftarrow$ Create new expert
$w_m \leftarrow 1$
**for** $i \leftarrow 1, \ldots, n$ {over all samples} **do**
  $\sigma \leftarrow 0$
  **for** $j \leftarrow 1, \ldots, m$ {over all current experts} **do**
    $\lambda \leftarrow$ Classification of $X_i$ by $e_j$
    **if** $\lambda \neq y_i$ **and** $i \bmod p = 0$ **then**
      $w_j \leftarrow \beta w_j$
    $\sigma_\lambda \leftarrow \sigma_\lambda + w_j$
  $\Lambda \leftarrow \text{argmax}_j \sigma_j$
  **if** $i \bmod p = 0$ {the updating period is met} **then**
    $w \leftarrow$ Normalize weights $w$
    $\{(e, w)\} \leftarrow$ Remove experts using threshold $\theta$
    **if** $\Lambda \neq y_i$ {global prediction is wrong} **then**
      $m \leftarrow m + 1$
      $e_m \leftarrow$ Create new expert
      $w_m \leftarrow 1$
  **for** $j \leftarrow 1, \ldots, m$ **do**
    $e_j \leftarrow e_j$ with provided $(X_i, y_i)$
  **output** $\Lambda$
```

1.2.2 Gaussian Naive Bayes classifier

The DWM algorithm presented in the previous subsection needs the experts for its run. Similarly as in [Kolter and Maloof, 2007, Subsection 3.1] we choose these experts to be Gaussian Naive Bayes classifiers. This is mostly for its ability to fast adaptation, sufficiently convincing accuracy, and natural way of excluding features from the classification process. First, we describe the method according to Murphy et al. [2006], and then we present intuition behind the ability of fast adaptation in Lemma 3 which is based on Ng and Jordan [2001].

Naive Bayes classifiers are based on the Bayes rule and use probability rewriting $P(y|X) = \frac{P(X|y)P(y)}{P(X)}$. A usual approach based on maximizing $P(y|X)$, which could be interpreted as which class label y is the most probable for the given feature vector X, can be rewritten as

$$argmax_y P(y|X) = argmax_y \frac{P(X|y)P(y)}{P(X)} = argmax_y P(X|y)P(y). \tag{1.2}$$

As we can see in Equation (1.2) for the classification we will be modelling only distributions $P(X|y)$ and $P(y)$. Using received data it is easy to model the distribution $P(y)$. On the other hand, modelling $P(X|y)$ can be incomparably more difficult. Therefore there is an assumption similar to one made for Lemma 1. We assume that given label y all features of feature vector X are mutually independent. We can mathematically rewrite this as

$$P(X|y) = P(X^1|y)P(X^2|X^1, y) \cdots P(X^d|X^1, \ldots, X^{d-1}, y) = \prod_{i=1}^{d} P(X^i|y).$$

This means that we model distribution for each feature alone which is much easier.

Based on the way of modelling this distribution there are several possible Naive Bayes classifiers such as Bernoulli, Multinomial, or Gaussian. The difference is in the assumption how would the distribution look like. For our tests we choose similarly to Kolter and Maloof [2007] the Gaussian Naive Bayes classifier, which we will now describe in detail using information from Bishop [2006].

Suppose i-th feature of feature vector X, $1 \leq i \leq d$ and j-th class label, $1 \leq j \leq |\mathcal{Y}|$, then we model $P(X^i|y_j)$ as normal distribution

$$\mathcal{N}(\mu_{i,j}, \sigma^2_{i,j}) = \frac{1}{\sqrt{2\pi\sigma^2_{i,j}}} e^{\frac{-(X^i - \mu_{i,j})^2}{2\sigma^2_{i,j}}},$$

where $\mu_{i,j}$ is the mean and $\sigma^2_{i,j}$ is the variance. Let assume we have $X_1, \ldots, X_N$ training feature vectors corresponding to label y_j. Using maximum likelihood estimation for these N samples drown from normal distribution $\mathcal{N}(\mu_{i,j}, \sigma^2_{i,j})$ we get

$$\begin{aligned}
&argmax_{\mu_{i,j},\sigma_{i,j}} \prod_{s=1}^{N} \frac{1}{\sqrt{2\pi\sigma_{i,j}^2}} e^{\frac{-(X_s^i-\mu_{i,j})^2}{2\sigma_{i,j}^2}} \\
&= argmax_{\mu_{i,j},\sigma_{i,j}} \log \prod_{s=1}^{N} \frac{1}{\sqrt{2\pi\sigma_{i,j}^2}} e^{\frac{-(X_s^i-\mu_{i,j})^2}{2\sigma_{i,j}^2}} \\
&= argmax_{\mu_{i,j},\sigma_{i,j}} \sum_{s=1}^{N} \log \frac{1}{\sqrt{2\pi\sigma_{i,j}^2}} e^{\frac{-(X_s^i-\mu_{i,j})^2}{2\sigma_{i,j}^2}} \\
&= argmin_{\mu_{i,j},\sigma_{i,j}} \sum_{s=1}^{N} -\log \frac{1}{\sqrt{2\pi\sigma_{i,j}^2}} e^{\frac{-(X_s^i-\mu_{i,j})^2}{2\sigma_{i,j}^2}} \\
&= argmin_{\mu_{i,j},\sigma_{i,j}} \sum_{s=1}^{N} -\log \frac{1}{\sqrt{2\pi\sigma_{i,j}^2}} \sum_{s=1}^{N} -\log e^{\frac{-(X_s^i-\mu_{i,j})^2}{2\sigma_{i,j}^2}} \\
&= argmin_{\mu_{i,j},\sigma_{i,j}} \frac{N}{2} \log 2\pi\sigma_{i,j}^2 + \sum_{s=1}^{N} \frac{(X_s^i-\mu_{i,j})^2}{2\sigma_{i,j}^2}.
\end{aligned}$$

In the above formula we used that logarithm is a strictly increasing function which provides us with the property that if $x > y > 0$ then $\log(x) > log(y)$. Then we used a transition from looking for the maximum of a formula to looking for the minimum of the negative formula. The rest comes from straightforward work with logarithms.

Now we study the derivative of the derived formula

$$argmin_{\mu_{i,j},\sigma_{i,j}} \frac{N}{2} \log 2\pi\sigma_{i,j}^2 + \sum_{s=1}^{N} \frac{(X_s^i-\mu_{i,j})^2}{2\sigma_{i,j}^2}$$

with respect to $\mu_{i,j}$ and $\sigma_{i,j}^2$ in order to find the extremes. First, we set the derivative with respect to $\mu_{i,j}$ to zero. We obtain

$$0 = \sum_{s=1}^{N} -2\frac{X_s^i - \mu_{i,j}}{2\sigma_{i,j}^2}$$

which is equivalent to

$$\mu_{i,j} = \frac{1}{N}\sum_{s=1}^{N} X_s^i.$$

By the same procedure for $\sigma_{i,j}^2$ we get

$$0 = \frac{1}{2}\frac{1}{\sigma_{i,j}^2}N - \frac{1}{2}\frac{1}{(\sigma_{i,j}^2)^2}\sum_{s=1}^{N}(X_s^i - \mu_{i,j})^2, \tag{1.3}$$

which is equivalent to

$$\sigma_{i,j}^2 = \frac{1}{N}\sum_{s=1}^{N}(X_s^i - \mu_{i,j})^2. \tag{1.4}$$

From Equations (1.3) and (1.4), we can see that for incremental learning, which is needed in the DWM algorithm (Algorithm 1), it is generally sufficient to store the derived sums which makes the procedure efficient.

The need for incremental learning is no problem for operating with mean, where we will store the number of samples and their sum. More problematic is computing the variance especially together with general stream scenario demand that we should not remember too many samples and ideally we should see each of them only once. This problem is solved in [Chan et al., 1982, Section 2.] and we sum it up in Lemma 2.

Lemma 2. *Let $x_i \in \mathbb{R}$ for all $1 < i < m \in \mathbb{N}$. Denote $T_{k,m} = \sum_{i=k}^{m} x_i$ and $S_{k,m} = \sum_{i=k}^{m}(x_i - \frac{1}{m}T_{k,m})^2$, then it holds that*

$$S_{1,m+n} = S_{1,m} + S_{m+1,m+n} + \frac{m}{n(m+n)}\left(\frac{m+n}{m}T_{1,m} - T_{1,n+m}\right)^2. \tag{1.5}$$

Lemma 2 gives us clear insight into what our incremental learning would look like. All parameters we will have to keep in memory are a number of samples for each class and for each feature both T and S, which we will increment according to Equation (1.5).

The concept drift brings the possibility of a fast change in the probability distributions. Therefore, we need to have a model which is capable of fast learning. A small number of samples needed for reasonably good performance is the key ability why we chose the Naive Bayes classifier. This important power of this procedure is deeply studied in Ng and Jordan [2001]. There are presented results on a number of needed samples for reaching an optimal error of the classifier. These theorems are unfortunately quite technically demanding. Therefore, we will not state them in this thesis but at least we formulate the continuous case of [Ng and Jordan, 2001, Lemma 3] as Lemma 3, which gives us good intuition behind Gaussian Naive Bayes classifier ability of fast learning.

Lemma 3. *Suppose Gaussian Naive Bayes classifier, where $X \subseteq \mathbb{R}^d$ and $\mathcal{Y} = \{y_0, y_1\}$. Then let any $\epsilon, \delta > 0$ be fixed and assume that for some fixed $\rho > 0$ it holds that $\rho \leq P(y = y_0) \leq 1 - \rho$. Let number of training samples $N = O(\frac{1}{\epsilon^2}\log(\frac{d}{\delta}))$, then with probability at least $1 - \delta$ it holds that $|\hat{\mu}_{i|y=b} - \mu_{i|y=b}| \leq \epsilon$, $|\hat{\sigma}^2 - \sigma^2| \leq \epsilon$ and $|\widehat{P}(y = b) - P(y = b)| \leq \epsilon$ for all $i = 1, \ldots, d$ and $b \in \mathcal{Y} = \{y_0, y_1\}$. $\hat{\mu}$ and $\hat{\sigma}$ are mean and variance produced by the classifier estimating real values μ and σ.*

In Lemma 3 we can generally see the speed of parameters approximation, which is in some sense connected with the logarithm of the number of features.

In this subsection, we described the procedure that we will use as part of the DWM algorithm used in this thesis as the main related work and baseline. In the second half of this subsection, we gave some intuition about the ability of fast learning of Gaussian Naive Bayes classifier as one of the reasons for choosing this procedure.

1.3 Concept drift detection

This section serves as the main background for the solution based on concept drift introduced in Section 4.2. First, we will show some basic techniques used for studying the change in the data stream. Then we will present one of the possible approaches for concept drift detection based on measuring distance between distributions using Hellinger distance. This approach is then transformed in Section 4.2.1 into an algorithm solving the setting formalized in Section 2.3.

1.3.1 Window strategy

In this subsection, we describe window-based strategy, which is one of the leading approaches for studying distribution in the stream scenarios. We use this method many times in this thesis but it is especially important for the solution introduced in Section 4.2, which is very much based on this approach.

When analysing changes in distribution within the data stream, the most common and straightforward method is to divide the stream into smaller segments and examine their properties. This method can be considered a specific form of sampling that takes into account the sequential nature of the data. It involves the creation of sets, each consisting of a predefined number of consecutive samples based on their positions in the stream. This technique is referred to as the window strategy. In this subsection, we describe this procedure, primarily following the discussions in [Dasu et al., 2006, Section 2] and [Bayram et al., 2022, Section 2].

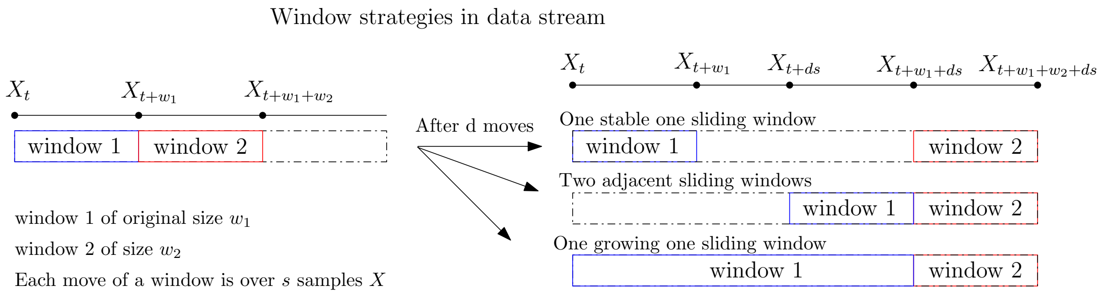

Window strategies in data stream
X_t
X_{t+w_1}
$X_{t+w_1+w_2}$
window 1
window 2
window 1 of original size w_1
window 2 of size w_2
Each move of a window is over s samples X
After d moves
X_t
X_{t+w_1}
X_{t+ds}
X_{t+w_1+ds}
$X_{t+w_1+w_2+ds}$
One stable one sliding window
window 1
window 2
Two adjacent sliding windows
window 1
window 2
One growing one sliding window
window 1
window 2

Figure 1.4: Illustration of the basic options of the window strategy in the data stream.

We define window $W_{i,n}$ as sequence of n samples of the data stream ending in X_i, $W_{i,n} = \{X_{i-n+1}, \ldots, X_i\}$. When we refer to window strategy we mean a process on a data stream consisting of a set of windows and a function which for each time step and possibly other given information defines how the windows change. Generally, a window strategy is used for comparing specific properties and distances of multiple windows. Some of the possible window strategies can be seen in Figure 1.4, and each of them will be briefly discussed in what follows.

The exact approach for using windows can vary based on different types of drift (see Subsection 2.2.2 and Attachment A.1), algorithm complexity, and the properties being studied. For instance, incremental drift can be better detected using two windows: one stable and one moving. Conversely, for sudden drift, it is more effective to use two adjacent sliding windows, as they provide data points showing severe changes in distribution. With incremental drift, adjacent sliding windows may repeatedly detect only small changes, despite an overall large drift.

Typically, sliding windows move by a single sample at a time. However, when handling large datasets or computationally demanding procedures, the window strategy that moves by more samples in each step can be used. Commonly, this involves moving by the size of the window itself, effectively segmenting the stream into windows used as needed in each step. This approach is utilized in Algorithm 2 in Subsection 1.3.2. Another strategy, employed in Algorithm 4, involves a growing and sliding window. Here, as the sliding window moves, the second window incorporates its old values to enhance generalization. However, keeping all samples in a growing window can be memory-intensive. Thus, it's more efficient to incorporate new samples in a way that optimizes memory usage, as seen in Algorithm 4, where growth is managed by adjusting a histogram.

Finally, many other window approaches exist, with both size and step changing during the process. One such approach is described in Attachment A.5, where we present a method for creating windows with adaptive sizes.

1.3.2 Hellinger distance

In this subsection, we describe the Hellinger distance and the algorithm for concept drift detection based on this distance. This subsection and especially Algorithm 2 will serve as a basis on which we will build our Algorithm 4 in Section 4.2.

General drift detection can be understood as estimating the severity of drift, which can naturally be used to highlight the most drifting feature subsets. It can also provide an estimate of the breaking point, the time step of a new concept's arrival. To measure feature drift, we use the well-known Hellinger measure, which is widely used for drift detection. This measure is well-studied and offers a natural way to focus on the drift in individual features or their subsets, as it combines their measured properties.

In this subsection, we present the general idea of histograms as discussed in [Liu et al., 2021, Section II. B]. We describe the Hellinger distance based on [Goldenberg and Webb, 2019, Subsection 4.1.1.1] and then illustrate the use of this measure for concept drift detection using information from [Ditzler and Polikar, 2011, Section III, A and C].

Hellinger distance is, the same as most drift detection techniques, based on distribution change measuring. These approaches usually use histogram density estimations. For building a histogram we divide the feature space into partitions and count the number of elements in each of them. When examining each feature separately, the bins of the histogram are intervals. To be more precise, N-bin histogram of interval $\mathcal{I}$ is N element partition denoted as $\{I_k\}_{k=1}^{N}$, where I_k is interval satisfying $I_k \subset \mathcal{I}$, $\cup_{k=1}^{N} I_k = \mathcal{I}$ and $I_i \cap I_j = \emptyset$ for each $i, j \in \{1, \dots, N\}$, $i \neq j$. This approach uses the important assumption of finite distribution support. In other words, for each feature, there exist finite real numbers B_{min} and B_{max} such that $P(X \leq x) = 0$ for $x \leq B_{min}$ and $P(X \geq x) = 0$ for $x \geq B_{max}$. Note that according to Ditzler and Polikar [2011] when we have no prior knowledge of data and constructing a histogram for w sampled values, it is usual to set the number of bins equal to $\lfloor \sqrt{w} \rfloor$.

Now, we present a definition of Hellinger distance which is almost fully adopted from [Goldenberg and Webb, 2019, Subsection 4.1.1.1].

Definition 3. *Let (Ω, B, ν) be a measure space and P and Q be probability measures that are absolutely continuous with respect to ν. The Hellinger integral (or*

Bhattacharyya coefficient) of P and Q is defined as $HI(P,Q) = \int_\Omega \sqrt{\frac{dP}{d\nu}}\sqrt{\frac{dQ}{d\nu}}d\nu$. The Hellinger distance is then defined as

$$D_H(P,Q) = \sqrt{1 - HI(P,Q)}. \tag{1.6}$$

Formula 1.6 can be rewritten using

$$\sqrt{1 - \int_\Omega \sqrt{\frac{dP}{d\nu}}\sqrt{\frac{dQ}{d\nu}}d\nu} =$$
$$\sqrt{\frac{1}{2}\int_\Omega \left(\sqrt{\frac{dP}{d\nu}}\right)^2 d\nu + \frac{1}{2}\int_\Omega \left(\sqrt{\frac{dQ}{d\nu}}\right)^2 d\nu - \frac{1}{2}\int_\Omega \sqrt{\frac{dP}{d\nu}}\sqrt{\frac{dQ}{d\nu}}d\nu} =$$
$$\sqrt{\frac{1}{2}\int_\Omega \left(\sqrt{\frac{dP}{d\nu}}\right)^2 + \left(\sqrt{\frac{dQ}{d\nu}}\right)^2 - 2\sqrt{\frac{dP}{d\nu}}\sqrt{\frac{dQ}{d\nu}}d\nu} =$$
$$\sqrt{\frac{1}{2}\int_\Omega \left(\sqrt{\frac{dP}{d\nu}} - \sqrt{\frac{dQ}{d\nu}}\right)^2 d\nu}.$$

We used basic rules for computing with integrals and a definition of probability from which it comes that for any probability measure P it holds that $\int_\Omega \sqrt{\frac{dP}{d\nu}}d\nu \equiv 1$.

Since we would be mostly interested in discrete data, the formula important for us is as follows.

Definition 4. *Let P and Q be a discrete distributions defined on the same space. We define Hellinger distance $D_H(P,Q)$ as*

$$D_H(P,Q) = \frac{1}{\sqrt{2}}\sqrt{\sum_{k=1}^{N}\left(\sqrt{P_k} - \sqrt{Q_k}\right)^2}. \tag{1.7}$$

For concept drift detection Hellinger distance is usually used as a metric quantifying the distribution difference of multiple windows, always two at a time. When no prior knowledge is available, the number of partitions N is usually set to the square root of the window size. The formula for the concept drift detection in the d dimensional case is

$$\frac{1}{d}\sum_{i=1}^{d}\sqrt{\sum_{j=1}^{N}\left(\sqrt{\frac{P_{j,i}}{\sum_{k=1}^{N}P_{k,i}}} - \sqrt{\frac{Q_{j,i}}{\sum_{k=1}^{N}Q_{k,i}}}\right)^2}, \tag{1.8}$$

where $P_{j,i}$ is number in bin j of feature i. Note that Formula 1.8 for one-dimensional feature vector is the same as Formula 1.7 except for the normalization. In Formula 1.8 we can see that overall drift is measured as the normalized sum of square roots of the single feature drift score. This fact is one of the reasons why we consider it natural to use the Hellinger distance for analysing single-feature drift.

The practical application of the Hellinger distance is illustrated in [Ditzler and Polikar, 2011, Section III, A and C]. It shows that while this distance measures the difference between two distributions, the difference tested on samples from the same distribution does not necessarily become zero. Instead, the differences between the same distributions remain nearly constant. Therefore, to detect a drift, we monitor deviations from these previously observed differences. This approach leads to the procedure described in [Ditzler and Polikar, 2011, Fig. 4], which we present as Algorithm 2.

1.4 Dirichlet distribution

In this section, we present Dirichlet distribution which is the key part in the process of establishing adaptive cluster prior probability threshold presented in Subsection 4.3.5. We briefly introduce Dirichlet distribution together with its derivation mostly according to [Lin, 2016, Sections 2.2-2.4], where we just add some more details.

Since Dirichlet distribution is a generalized Beta distribution we introduce this distribution first. For this, we briefly remind Gamma distribution which is highly connected with both previous distributions.

Definition 5. *It is said that random variable Y has a Gamma distribution with parameters α and β, denoted as $G(\alpha, \beta)$, if it has a probability density function $f(x)$ of the form*

$$f(x) = \begin{cases} \frac{1}{\Gamma(\alpha)\beta^\alpha} x^{\alpha-1} e^{-\frac{x}{\beta}} & \textit{if } 0 < x < \infty \\ 0 & \textit{otherwise} \end{cases}$$

where $\alpha > 0$, $\beta > 0$ and $\Gamma(\alpha) = \int_0^\infty t^{\alpha-1} e^{-t} dt$ is the Gamma function.

Gamma distribution is often used in cases where we are interested in waiting time until a certain number of events occur. In this scenario, α would stand for the aspect of how quickly those events are likely to happen and β for the average rate of event occurrences. From this intuition, we can see that Gamma distribution is highly connected with Binomial distribution and the problem discussed in Subsection 4.3.4. Closely related to the Binomial distribution is also Beta distribution, which is defined in Definition 6.

Definition 6. *It is said that random variable Y has a Beta distribution with parameter α and β if it has a probability density function $f(y)$ of the form*

$$f(y) = \begin{cases} \frac{\Gamma(\alpha+\beta)}{\Gamma(\alpha)\Gamma(\beta)} y^{\alpha-1} (1-y)^{\beta-1} & \textit{if } 0 < y < 1 \\ 0 & \textit{otherwise} \end{cases},$$

where $\alpha > 0$, $\beta > 0$.

One of the possible views on Beta distribution is that it gives success probability of the Bernoulli experiment when we are given its outcome. Hence, as the Bernoulli distribution takes the probability of positive observation as a parameter and models the number of successes, the Beta distribution works with this probability as input y and can be considered as modelling the probability of success. This point of view would imply that parameters α and β could be set

Algorithm 2 Hellinger distance concept drift detection ([Ditzler and Polikar, 2011, Fig. 4])

```
**Inputs**

  • $\{D_i\}_{i=1}^{n_D}$ - Distributions, each is finite, non-empty subset of feature space $\mathbb{R}^d$.
  • $\gamma$ - standard deviation parameter for drift threshold, $\gamma \in \mathbb{R}$

$\lambda \leftarrow 1$
$D_\lambda \leftarrow D_1$
for $t \leftarrow 2, \dots, n_D$ {over all distributions} do
    Generate a histogram $P$ from $D_t$ and a histogram $Q$ from $D_\lambda$. Each with $b = \lfloor \sqrt{N} \rfloor$ bins, where $N$ is cardinality of $D_t$
    Calculate Hellinger distance $\delta_H(t)$ as distance between $P$ and $Q$ which is eqaul to
        $$\frac{1}{d}\sum_{i=1}^{d}\sqrt{\sum_{j=1}^{N}\left(\sqrt{\frac{P_{j,i}}{\sum_{k=1}^{b} P_{k,i}}} - \sqrt{\frac{Q_{j,i}}{\sum_{k=1}^{N} Q_{k,i}}}\right)^2} \quad \text{(Formula (1.8)}$$
    Compute the difference in Hellinger distance:
        $$\epsilon(t) = \delta_H(t) - \delta_H(t-1)$$
    Update the adaptive threshold:
        $$\hat{\epsilon} = \frac{1}{t-\lambda-1}\sum_{i=\lambda}^{t-1}|\epsilon(i)|$$
        $$\hat{\sigma} = \sqrt{\frac{\sum_{i=\lambda}^{t-1}(|\epsilon(i)| - \hat{\epsilon})^2}{t-\lambda-1}}$$
    $\beta(t) \leftarrow \hat{\epsilon} + \gamma\hat{\sigma}$
    if $|\epsilon(t)_i| > \beta(t)_i$ {Check if drift is present} then
        $\lambda \leftarrow t$
        $D_\lambda \leftarrow D_t$
    else
        $D_\lambda^{(i)} \leftarrow \{D_\lambda^{(i)}, D_t^{(i)}\}$
```

as α is equal to a number of positive observations while β is assigned with the number of negative observations.

Now, we will derive the Dirichlet distribution from the previous two distributions. Let $X_1, \ldots, X_k$ be an independent random variables where each X_i is from the $G(\alpha_i, 1)$ for each $i = 1, \ldots, k$. Now, we consider their joint probability density function, which is of the form

$$f(x_1, \ldots, x_k) = \begin{cases} \prod_{i=1}^{k} \frac{1}{\Gamma(\alpha_i)} x^{\alpha_i - 1} e^{-x_i} & \textit{if } 0 < x_i < \infty \\ 0 & \textit{otherwise} \end{cases}.$$

This formula is provided by independence and by setting $\beta = 1$. Now, we consider $Y_i = \frac{X_i}{X_1 + X_2 + \cdots + X_k}$ for each $i = 1, \ldots, k-1$ and $Z_k = X_1 + X_2 + \cdots + X_k$. We use change of variables with mapping $\{(x_1, \ldots, x_k) \ : \ 0 < x_i < \infty, i = 1, \ldots, k\}$ onto $\{(y_1, \ldots, y_{k-1}, z_k) : y_i > 0, \ i = 1, \ldots, k-1, \ 0 < z_k < \infty, \ y_1 + \cdots + y_{k-1} < 1\}$. The inverse functions are given by $x_1 = y_1 z_k, \ x_2 = y_2 z_k, \ldots, x_{k-1} = y_{k-1} z_k, \ x_k = z_k(1 - y_1 - \cdots - y_{k-1})$. Hence the determinant of the Jacobian matrix is of the form

$$\begin{vmatrix} \frac{\partial x_1}{\partial y_1} & \cdots & \frac{\partial x_1}{\partial y_{k-1}} & \frac{\partial x_1}{\partial y_k} \\ \vdots & \ddots & \vdots & \vdots \\ \frac{\partial x_{k-1}}{\partial y_1} & \cdots & \frac{\partial x_{k-1}}{\partial y_{k-1}} & \frac{\partial x_{k-1}}{\partial y_k} \\ \frac{\partial x_k}{\partial y_1} & \cdots & \frac{\partial x_k}{\partial y_{k-1}} & \frac{\partial x_k}{\partial z_k} \end{vmatrix} = \begin{vmatrix} z_k & 0 & \cdots & 0 & y_1 \\ 0 & z_k & \ddots & \vdots & y_2 \\ \vdots & \ddots & \ddots & 0 & \vdots \\ 0 & \cdots & 0 & z_k & y_{k-1} \\ -z_k & -z_k & \cdots & -z_k & (1 - y_1 - \cdots - y_{k-1}) \end{vmatrix}$$

$$= \begin{vmatrix} z_k & 0 & \cdots & 0 & y_1 \\ 0 & z_k & \ddots & \vdots & y_2 \\ \vdots & \ddots & \ddots & 0 & \vdots \\ 0 & \cdots & 0 & z_k & y_{k-1} \\ 0 & 0 & \cdots & 0 & 1 \end{vmatrix} = z_k^{k-1},$$

where for the penultimate equality we used that the addition of one row of the matrix to another does not change its determinant. The result is then derived by the simple computation of the triangular matrix determinant. Now, we can use theorem from Taboga [2021], formulated as Theorem 4, which presents the formula of joint probability density function of one-to-one transformation.

Theorem 4. *Let Y be a k continuous random vector with support S_Y and joint probability density function $f_Y(y)$. Let $g \ : \ \mathbb{R}^{k-1} \to \mathbb{R}^{k-1}$ be one-to-one and differentiable on the support of Y. Denote $J_{g^{-1}}(x)$ the Jacobian matrix of $g^{-1}(x)$. If the determinant of the Jacobian matrix satisfies that it is non-negative for all x from S_X, where $X = g(Y)$ and $S_X = \{x = g(y) \ : \ x \in S_Y\}$, then the joint probability density function of X is*

$$f_X(x) = \begin{cases} f_Y(g^{-1}(x)) |det(J_{g^{-1}}(x))| & \textit{if } x \in S_X \\ 0 & \textit{if } x \notin S_X \end{cases}.$$

Using Theorem 4, we get that probability density function of $Y_1, \ldots, Y_{k-1}, Z_k$

is

$$
\begin{aligned}
&f(y_1,\dots,y_{k-1},z_k)\\
&=\frac{1}{\prod_{i=1}^{k}\Gamma(\alpha_i)}\left(\prod_{i=1}^{k-1} z_k^{\alpha_i-1}y_i^{\alpha_i-1}\right)e^{\left(-\sum_{i=1}^{k-1} z_k y_i\right)}\\
&\left(z_k\left(1-\sum_{i=1}^{k-1} y_i\right)\right)^{\alpha_k-1} e^{\left(-z_k\left(1-\sum_{i=1}^{k-1} y_i\right)\right)} z_k^{k-1}\\
&=\frac{1}{\prod_{i=1}^{k}\Gamma(\alpha_i)}\left(\prod_{i=1}^{k-1} y_i^{\alpha_i-1}\right)\left(1-\sum_{i=1}^{k-1} y_i\right)^{\alpha_k-1} z_k^{\left(\sum_{i=1}^{k}\alpha_i\right)-1} e^{-z_k}\\
&=\frac{y_1^{\alpha_1-1}\cdots y_{k-1}^{\alpha_{k-1}-1}(1-y_1-\cdots-y_{k-1})^{\alpha_k-1}}{\Gamma(\alpha_1\cdots\Gamma(\alpha_k} e^{-z_k} z_k^{\alpha_1+\cdots+\alpha_k-1}.
\end{aligned}
$$

Now, we can integrate out z_k and obtain

$$
\begin{aligned}
&f(y_1,\dots,y_{k-1})=\int_0^\infty f(y_1,\dots,y_{k-1},z_k)dz_k\\
&=\frac{1}{\prod_{i=1}^{k}\Gamma(\alpha_i)}\left(\prod_{i=1}^{k-1} y_i^{\alpha_i-1}\right)\left(1-\sum_{i=1}^{k-1} y_i\right)^{\alpha_k-1}\int_0^\infty z_k^{\left(\sum_{i=1}^{k}\alpha_i\right)-1} e^{-z_k}dz_k\\
&=\frac{1}{\prod_{i=1}^{k}\Gamma(\alpha_i)}\left(\prod_{i=1}^{k-1} y_i^{\alpha_i-1}\right)\left(1-\sum_{i=1}^{k-1} y_i\right)^{\alpha_k-1}\Gamma\left(\sum_{i=1}^{k}\alpha_i\right),
\end{aligned}
$$

which is exactly the density of the Dirichlet distribution with parameters $\alpha_1,\dots,\alpha_k$ as can be seen in Definition 7. Note that for last equality we used definition of Gamma function, which we presented in Definition 5.

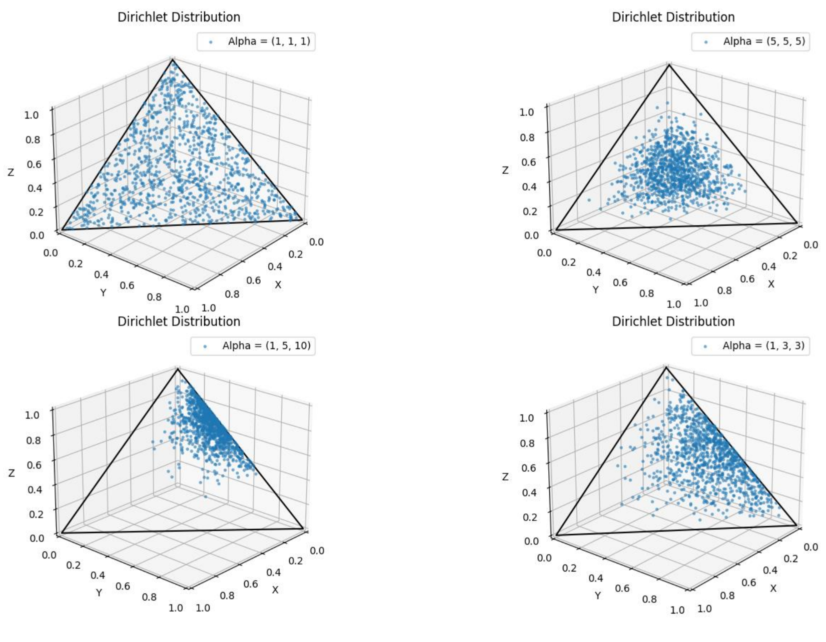


Figure 1.5: Illustration of influence of various choices of alphas on Dirichlet distribution. Each of the graphs represents 1000 random samples from Dirichlet distribution with according values of alpha.

Definition 7. *Let Y^k be a vector with k components, where $Y_i \geq 0$ for $i = 1, 2, \ldots, k$ and $\sum_{i=1}^{k} Y_i = 1$. Also, let $\alpha^k = (\alpha_1, \alpha_2, \ldots, \alpha_k)$, where $\alpha_i > 0$ for each i. Then the Dirichlet probability density function, denoted as $Dir(\alpha_1, \alpha_2, \ldots, \alpha_k)$, is*

$$f(y^k) = \frac{\Gamma(\alpha_0)}{\prod_{i=1}^{k} \Gamma(\alpha_i)} \prod_{i=1}^{k} y_i^{\alpha_i - 1}$$

where $\alpha_0 = \sum_{i=1}^{k} \alpha_i$, $y_1 + \cdots + y_{k-1} < 1$ and $y_k = 1 - y_1 - \cdots - y_{k-1}$.

We can see that an important part of the Dirichlet distribution presented in Definition 7 is parameter α. In Figure 1.5 we give some examples of Dirichlet distribution for various choices of α. This parameter can be generally considered as some kind of weight. The higher α_i the greater the influence of the according component Y^i. On the other hand, if $\alpha_i < 1$ then we can consider it as an aspect of decreasing the effect of Y^i. If all alphas are the same, then we call the distribution flat and it holds that it is symmetric. The case where all alphas have value 1 represents a scenario where all points are distributed uniformly. In Subsection 4.3.5, we propose a prior probability threshold based on Dirichlet distribution. In this approach, alphas would be the key part since they will represent our assumption on cluster probabilities.

1.5 Clustering algorithms

Our solution presented in Section 4.3 is based on the incremental Gaussian Mixture Models (GMM) algorithm. In this section, we describe the derivation of the original non-incremental form of the algorithm, which provides the best insight into the underlying principles of the process.

First, we present a brief introduction of techniques in Subsection 1.5.1. In Subsection 1.5.2, we introduce the K-means algorithm, a simpler variant of the GMM algorithm, which effectively illustrates the core concept of the process. Next, in Subsection 1.5.3, we briefly discuss the Mahalanobis distance, a crucial component of the GMM algorithm. Finally, in Subsection 1.5.4, we present the non-incremental form of the GMM algorithm, which will serve as the foundation for describing its incremental form in Section 1.6.

1.5.1 Algorithms introduction

In this subsection, we provide a brief introduction to the techniques discussed in this section, contextualizing them within the scope of the thesis.

A prominent approach in data stream outlier detection (see Subsection 1.7.1) is, according to Tamboli and Shukla [2016], the clustering approach. The general idea of this approach is to partition the feature space into clusters and assess how well new incoming samples fit into these clusters. There are various methods to perform this partitioning, and we will describe two well-known techniques: the K-means algorithm and the Gaussian Mixture Model (GMM) algorithm. Both methods are based on distance measurement and optimization principles.

The K-means algorithm is relatively straightforward as it assigns each data point to a specific cluster based on distance. In contrast, the GMM algorithm generalizes this concept by allowing each data point to contribute to multiple clusters, resulting in more precise clustering.

Both algorithms were originally designed for a traditional scenario where all training samples are available during the training phase. This is significantly different from the stream scenario described in Subsection 2.3, where we receive and must utilize one sample at each time step. However, explaining these algorithms in their original form, without data stream constraints, can help to understand their fundamental workings. Therefore, we begin our discussion of these approaches in this more traditional setting, primarily based on [Bishop, 2006, Chapter 9].

1.5.2 K-means algorithm

In this subsection, we describe the K-mean algorithm for a non-stream scenario together with its derivation. The presented information will serve for a better understanding of a more general process called the Gaussian Mixture Models algorithm which we introduce in Subsection 1.5.4.

The idea of the K-means algorithm is to shift the centers of the clusters to minimize the distance from the observed points. To be more precise let $X_1, \dots, X_n$ be feature vectors (points) and K be the number of clusters, where each is defined by its center μ_i and let $z_{i,k} \in \{0, 1\}$ be an indicator of assigning X_i to cluster k. We want to minimize the formula

$$\sum_{i=1}^{n} \sum_{k=1}^{K} z_{i,k} \|X_i - \mu_k\|^2.$$

For given centers $\mu_1, \dots, \mu_K$ the best possible $z_{i,k}$ is the one assigning X_i to cluster j for which $\|X_i - \mu_j\|^2$ is minimal. For computing the optimal centers satisfying $\mu_j = argmin_\mu \sum_i z_{i,k} \|X_i - \mu\|^2$, we make derivative of the latter formula with respect to μ and we get

$$\frac{\partial f}{\partial \mu} = \sum_i z_{i,k} 2(X_i - \mu)(-1).$$

Now, we set the derivative equal to zero, then

$$0 = -\sum_i z_{i,k} 2(X_i - \mu) \tag{1.9}$$

$$\mu = \frac{\sum_i z_{i,k} X_i}{\sum_i z_{i,k}}, \tag{1.10}$$

which is the mean of the points in the cluster.

Note that up to this point in this section, we have utilized L^2 norm as the measure of distance. This metric is typically insufficient for the more advanced approach. Therefore, in the following subsection, we will briefly introduce the Mahalanobis distance, which will be frequently employed throughout the thesis.

1.5.3 Mahalanobis distance

In this subsection, we will briefly describe Mahalanobis distance as a key part of measuring the distances for the GMM algorithm used in our solution in Section 4.3.

Most drift detection techniques have a problem in that they, similarly as presented in Subsection 1.3.2), measure the drift between two distributions. This means that in a stream scenario we need to collect enough arriving samples to produce the distributions, which results in a detection delay. Contrary to this, we want to be able to get some information about the drift of just incoming data. For this task, we choose well-known and well-studied Mahalanobis distance. The

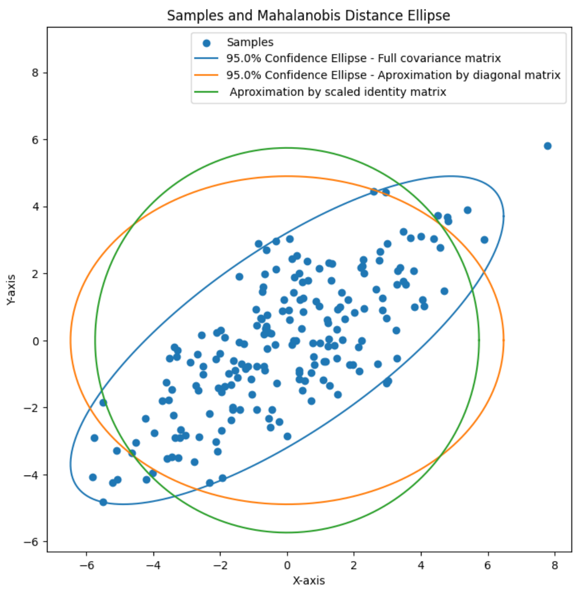


Figure 1.6: Illustration of cluster shapes given by Mahalanobis distance for different covariance matrix approximation.

Mahalanobis distance $D_M(D, X)$, where D is distribution given by mean value μ and covariance matrix Σ and X is feature vector, is defined (as can be seen in [McLachlan, 1999, Section 2]) as

$$D_M(D, X) = \sqrt{(X-\mu)^T\Sigma^{-1}(X-\mu)}.$$

The squared Mahalanobis distance is a crucial component of the GMM algorithm. For a clearer illustration of this distance, we present Figure 1.6. In high-dimensional cases, calculating the full covariance matrix can be computationally demanding. Therefore, approximations are often employed.

In Figure 1.6, we observe that in two dimensions, a cluster formed using the Mahalanobis distance takes the shape of a circle when Σ is a scaled identity matrix. When Σ is approximated by a diagonal matrix, the cluster shape becomes an ellipse with its semi-minor and semi-major axes aligned with the x and y axes, respectively. If the matrix is full, the ellipse can have any orientation.

In our scenario, we will utilize both the full covariance matrix and its diagonal approximation, as these matrices will be computed incrementally. This approach is detailed in Subsections 4.3.3 and 4.3.4.

1.5.4 Gaussian Mixture Model algorithm

In this subsection, we describe the Gaussian Mixture Model (GMM) algorithm for the non-stream scenario. This subsection serves as a base for the incremental version of the procedure described in Section 1.6, which is crucial for our solution presented in Section 4.3.

Based on [Bishop, 2006, Section 9.2], we will present the GMM algorithm, which can be understood as a generalization of the K-means algorithm described in Subsection 1.5.2. GMM algorithm supposes that each cluster in feature space can be modelled by the normal distribution, which for d dimensional vector X has form

$$\mathcal{N}(\mu, \Sigma) = \sqrt{\frac{1}{(2\pi)^d|\Sigma|}} exp(-\frac{1}{2}(X-\mu)^T\Sigma^{-1}(X-\mu)),$$

where μ is the mean and Σ is covariance matrix. Note the relation of the formula with Mahalanobis distance presented in Subsection 1.5.3. The Gaussian mixture of K clusters is defined as their combination of the form

$$\sum_{k=1}^{K} \pi_k \mathcal{N}(\mu_k, \Sigma_k),$$

where π_k represents the weights of the clusters and if they form a distribution then they can be seen as prior probabilities of belonging to each cluster.

Let z be a K dimensional binary random variable standing for assigning of feature vector to each cluster. This means, the variable of a form that for exactly one i, $z^i = 1$ and for the rest it equals to 0. We can rewrite the prior probability of the vector X, $P(X)$, to obtain above formula as follows

$$P(X) = \sum_z P(X, z) = \sum_z P(z)P(X|z) = \sum_{k=1}^{K} \pi_k \mathcal{N}(\mu_k, \Sigma_k).$$

Generally, we can interpret the above probability as the measure of how well the feature vector X fits in the model composed of the clusters and their weights. Maximization of this probability for all available data in the dataset would therefore create optimal clustering. The logarithm of this probability is equal to

$$\sum_{i=1}^{n} \log \sum_{k=1}^{K} \pi_k \mathcal{N}(\mu_k, \Sigma_k),$$

then we can use the maximum likelihood estimation to minimize

$$\mathcal{L}(X) = -\sum_{i=1}^{n} \log \sum_{k=1}^{K} \pi_k \mathcal{N}(\mu_k, \Sigma_k) \tag{1.11}$$

to find the optimal parameters. In order to find the minimum, we set the derivative of $\mathcal{L}(X)$ with respect to μ_k to zero and proceed with the equation in the following way.

$$\frac{\partial \mathcal{L}}{\partial \mu_k} = -\sum_{i=1}^{n} \frac{1}{\sum_{j=1}^{K} \pi_j \mathcal{N}(X_i; \mu_j, \Sigma_j)} \pi_k \mathcal{N}(X_i; \mu_k, \Sigma_k)\Sigma^{-1}(X_i - \mu_k) \tag{1.12}$$

$$0 = -\sum_{i=1}^{n} \frac{\pi_k \mathcal{N}(X_i; \mu_k, \Sigma_k)}{\sum_{j=1}^{K} \pi_j \mathcal{N}(X_i; \mu_j, \Sigma_j)} \Sigma^{-1}(X_i - \mu_k), \tag{1.13}$$

where we used that the derivative of $\mathcal{N}(\mu, \Sigma)$ with respect to μ_k is of the form

$$\begin{aligned}\sqrt{\frac{1}{(2\pi)^d|\Sigma|}}e^{-\frac{1}{2}(X_i-\mu_k)^T\Sigma^{-1}(X_i-\mu_k)}(-(-\frac{1}{2})\Sigma^{-1}(X_i-\mu_k)-(-\frac{1}{2})(X_i-\mu_k)^T\Sigma^{-1})\\ =\mathcal{N}(\mu, \Sigma)\Sigma^{-1}(X_i-\mu_k).\end{aligned}$$

Note that equality holds since Σ^{-1} is a covariance matrix and hence it is symmetric. Denote $r(z_{i,k}) = \frac{\pi_k\mathcal{N}(X_i;\mu_k,\Sigma_k)}{\sum_{j=1}^K \pi_j\mathcal{N}(X_i;\mu_j,\Sigma_j)}$, this term is called responsibility and can be interpreted as probability of X_i belonging to k-th cluster, then using Equation (1.12) we get

$$0 = -\sum_{i=1}^{n} r(z_{i,k})\Sigma^{-1}(X_i-\mu_k) \tag{1.14}$$

$$\mu_k = \frac{\sum_i r(z_{i,k})X_i}{\sum_i r(z_{i,k})}. \tag{1.15}$$

It is important to realize that Equation (1.15) is not an analytical solution since $r(z_{i,k})$ depends on μ_k. Also note the similarity of this equation with Equation (1.9), where we generalized the mean of the points assigned to a specific cluster by a combination of all points where each contributes with weight given by responsibility to this cluster.

When considering the value of π_k, we need to add constraint that $\sum_{i=1}^K \pi_k = 1$ since we want π to create a distribution. Therefore, the Lagrangian is of the form $\mathcal{L}(X) - \lambda(\sum_k \pi_k - 1)$ and its derivative with respect to π_k is equal to

$$\sum_i \frac{\mathcal{N}(X_i;\mu_k,\Sigma_k)}{\sum_{j=1}^K \pi_j\mathcal{N}(X_i;\mu_j,\Sigma_j)} - \lambda.$$

When we set the derivative to zero we get

$$\begin{aligned}0 &= \sum_i \frac{\mathcal{N}(X_i;\mu_k,\Sigma_k)}{\sum_{j=1}^K \pi_j\mathcal{N}(X_i;\mu_j,\Sigma_j)} - \lambda\\ 0 &= \sum_i \frac{\pi_k\mathcal{N}(X_i;\mu_k,\Sigma_k)}{\sum_{j=1}^K \pi_j\mathcal{N}(X_i;\mu_j,\Sigma_j)} - \lambda\pi_k\\ 0 &= \sum_i r(z_{i,k}) - \lambda\pi_k\\ \pi_k &= \frac{1}{\lambda}\sum_i r(z_{i,k}).\end{aligned}$$

Then if we assign every point and we want π to be a distribution we set $\lambda = n$.

By the same approach, which means by setting the derivative of $\mathcal{L}(X) - \lambda(\sum_k \pi_k - 1)$ with respect to covariance matrix Σ to zero, we get the formula for Σ, which is of the form

$$\Sigma_k = \frac{\sum_i r(z_{i,k})(X_i-\mu_k)(X_i-\mu_k)^T}{\sum_i r(z_{i,k})}.$$

The problem of the non-existence of the easy analytical solution for Formula 1.11 results in an approach where we update the function in two separate steps. This approach is generally known as the EM algorithm, which is presented in Attachment A.4.

1.6 Incremental clustering algorithms

In Section 1.5, we derived the clustering algorithm known as Gaussian Mixture Models (GMM) in its usual non-incremental form. Now, we will describe the incremental form of the GMM algorithm, adapted for use in a data stream scenario. This incremental GMM algorithm serves as the foundational algorithm for our solution presented in Section 4.3.

First, we adapt GMM into its incremental form in Subsection 1.6.1, and then we describe a faster version achieved by slight modifications of the incremental equations in Subsection 1.6.2.

1.6.1 Incremental Gaussian mixture model

In this subsection, we describe an incremental version of the Gaussian Mixture Model algorithm. This will be in a slightly modified version, which we present in Subsection 1.6.2, used in Section 4.3 as the ground underlying algorithm of our solution. In the following text, we present the algorithm and also give insight into how it is derived.

The form of the incremental version of the Gaussian mixture model is presented in Engel and Heinen [2010]. For the following sections, we will adopt the notation from Section 1.5. For a proper incremental version, we need to develop the recurrent formulas for the mean μ_k, covariance matrix Σ_k, and the prior π_k. Such equations are given in [Engel and Heinen, 2010, Equation 11.-14.] and for k-th cluster and feature vector X_i, they are of the form

$$sp_k = sp_k + r(z_{i,k}) \tag{1.16}$$

$$\mu_k = \mu_k + \frac{r(z_{i,k})}{sp_k}(X_i - \mu_k) \tag{1.17}$$

$$\Sigma_k = \Sigma_k - (\mu_k - \mu_k^{old})(\mu_k - \mu_k^{old})^T + \frac{r(z_{i,k})}{sp_k}((X_i - \mu_k)(X_i - \mu_k)^T - \Sigma_k) \tag{1.18}$$

$$\pi_k = \frac{sp_k}{\sum_{j=1}^{K} sp_j}, \tag{1.19}$$

where μ_k^{old} refers to value μ_k in previous step. The formula for the covariance matrix is based on the sample covariance matrix and it is used in situations where we want the full covariance matrix approximation (see Subsection 1.5.3). A scenario where we would track only diagonal approximation of the covariance matrix would very much simplify into tracking variances, which is done in Subsection 1.2.2 during the run of Gaussian Naive Bayes classifier.

Rather than detailing the entire parameter estimation process, we provide an overview of the techniques and the overall procedure. This summary draws on information from [Heinen, 2011, Subsection 2.5.2].

The process follows a general incremental version of the EM algorithm (see Attachment A.4). There is estimation step which consists of computing posterior probabilities of component membership $r(z_{i,k})$ $(P(z^k = 1|X_i))$ and maximization step where we use computed posterior probability to produce new estimates of mean μ_k, covariance matrix Σ_k and prior π_k $(P(z^k = 1))$.

Since the approach is based on *Robinson-Monroe* algorithm, we briefly describe this technique according to [Bishop, 2006, Subsection 2.3.5]. The approach supposes that we have function $f(\theta) = 0$, which has a unique root θ^*. We assume that we can not directly observe the value of $f(\theta)$, but instead, we have access to measurements of the random variable $F(\theta)$ for which it holds that $\mathbb{E}(F(\theta)) = f(\theta)$. Now suppose that we observe one value of $F(\theta)$ at a time. The general structure of the incremental algorithm is then of a form

$$\theta_{n+1} = \theta_n - a_n F(\theta_n), \tag{1.20}$$

where coefficients $\{a_i\}$ represent positive step sizes that satisfy conditions

$$\begin{aligned} &\lim_{i\to\infty} a_i = 0 \\ &\sum_{i=1}^{\infty} a_i = \infty \\ &\sum_{i=1}^{\infty} a_i^2 < \infty. \end{aligned}$$

Then Equation (1.20) converges to the root with probability one. To be more detailed, the procedure for mentioned two random variables θ and F also assumed that conditional variance is finite so that $\mathbb{E}((F(\theta) - f(\theta))^2) < \infty$ and that $f(\theta) > 0$ for $\theta > \theta^*$ and $f(\theta) < 0$ for $\theta < \theta^*$.

Now, we get back to our problem, a process of deriving the desired recurrent equations is as follows. Assume $X_1, \ldots, X_n$ to be samples independently drawn from the mixture distribution, which we are trying to describe. Then each of the samples will be used to update the parameters and make them more accurate. The process is based on the Robbins-Monroe stochastic approximation described above. We consider the likelihood of the parameters Θ equal to the joint probability distribution for all given samples which is equal to

$$P(\{X_1, \ldots, X_n\}|\Theta) = \prod_{i=1}^{n} P(X_i|\Theta) = \prod_{i=1}^{n} \sum_{k=1}^{K} P(X_i|z^k = 1)P(z^k).$$

Choosing the similar approach as in previous subsections we would maximize the value of the formula by derivative which we would set to zero and find the maximal likelihood estimation for Θ. This approach and consequent use of the Robinson-Monroe algorithm together with some technical adjustments are possible for deriving an equation for the prior as is shown in [Heinen, 2011, Subsection 2.5.2]. Unfortunately, this approach becomes quite complex for the estimation of the mean and covariance matrix. Therefore, another technique is chosen. According to Keehn [1965] it can be seen that when estimating mean and covariance matrix by their sample estimations, which means

$$\begin{aligned} \hat{\mu} &= \frac{1}{n} \sum_{i=1}^{n} X_i \\ \hat{\Sigma} &= \frac{1}{n-1} \sum_{i=1}^{n} (X_i - \hat{\mu})(X_i - \hat{\mu})^T \end{aligned}$$

and the sample follows a normal distribution, then joint density function
$P(\mu, \Sigma|X_1, \ldots, X_n)$ is reproducible Gauss-Wishart distribution, which is four parameters distribution combining Gaussian distribution over mean and Wishart

distribution over the covariance matrix. Then according to [Heinen, 2011, Subsection 2.5.2] an estimation for single cluster starting with initial guesses μ^0 and Σ^0, provides equations of following form

$$\begin{aligned}
\omega^1 =& \omega^0 + n \quad v^1 = v^0 + n \\
\mu_1 =& \frac{\omega^0 \mu^0 + n\hat{\mu}}{\omega^0 + n} \\
\Sigma^1 =& \frac{(v^0\Sigma^0 + \omega^0\mu^0(\mu^0)^T) + n\hat{\mu}\hat{\mu}^T - \omega^1\mu^1(\mu^1)^T}{v^0 + n},
\end{aligned}$$

where $\hat{\mu}$ is a mean of samples $X_1, \ldots, X_n$, values ω^0 and v^0 stands for confidence in our initial guesses μ^0 and Σ^0, which typically means the number of samples we used for their estimation. At this moment we need to realize that if we are operating in Gaussian Mixture Model scenario, it would not make good sense to count the number of samples, since they are assigned by probability. This probability for i-th sample and k-th cluster is equal to $r(z_{i,k})$ and it is natural to use it together with their sum as this confidence when producing the estimation of the parameters. This finishes a brief illustration of Equations (1.16)-(1.19) derivation.

The incremental procedure presented in this subsection represents a strong mechanism for computing GMM in stream scenario. A shortcoming stands in the computational costs, especially in the computation of the inverse covariance matrix needed for determining responsibility value and in the covariance matrix determinant computation. In Subsection 1.6.2, we present a possible way how we can avoid to this computation by adapting Equations (1.16)-(1.19) to calculate directly with the matrix inverse and with the determinant.

1.6.2 Fast Incremental GMM

In this subsection, we describe the adaptation of the algorithm presented in Subsection 1.6.1 into a faster version by modifying its incremental equations. This modified algorithm is then used as the foundational part of our solution in Section 4.3.

We adapt Equations (1.16)-(1.19) following the approach of Pinto and Engel [2015]. This adaptation reduces the computational complexity of the desired parameters by enabling their direct calculation during incremental approximation.

We will outline the approach according to [Pinto and Engel, 2015, Section 3]. First, we present the process of incrementally approximating the inverse covariance matrix, Σ^{-1}, which we denote as Λ. In the second part of the subsection, we discuss a possible approach for the incremental computation of the covariance matrix determinant, which is also necessary for the algorithm.

Lets rewrite Equation (1.18) as follows

$$\begin{aligned}
&\Sigma_k - (\mu_k - \mu_k^{old})(\mu_k - \mu_k^{old})^T + \frac{r(z_{i,k})}{sp_k}((X_i - \mu_k)(X_i - \mu_k)^T - \Sigma_k) \\
&= \Sigma_k - (\mu_k - \mu_k^{old})(\mu_k - \mu_k^{old})^T + \frac{r(z_{i,k})}{sp_k}(X_i - \mu_k)(X_i - \mu_k)^T - \frac{r(z_{i,k})}{sp_k}\Sigma_k \\
&= \left(1 - \frac{r(z_{i,k})}{sp_k}\right)\Sigma_k - (\mu_k - \mu_k^{old})(\mu_k - \mu_k^{old})^T + \frac{r(z_{i,k})}{sp_k}(X_i - \mu_k)(X_i - \mu_k)^T,
\end{aligned}$$

where the last formula can be seen as a sequence of two rank one updates of a form

$$\overline{\Sigma}_k = \left(1 - \frac{r(z_{i,k})}{sp_k}\right)\Sigma_k + \frac{r(z_{i,k})}{sp_k}(X_i - \mu_k)(X_i - \mu_k)^T$$
$$\Sigma_k^{new} = \overline{\Sigma}_k - (\mu_k - \mu_k^{old})(\mu_k - \mu_k^{old})^T.$$

This structure of computation provides us with the possibility of using Sherman-Morison formula introduced in Sherman and Morrison [1950], which states

$$(\mathbf{A} + \mathbf{u}\mathbf{v}^T)^{-1} = \mathbf{A}^{-1} - \frac{\mathbf{A}^{-1}\mathbf{u}\mathbf{v}^T\mathbf{A}^{-1}}{1 + \mathbf{v}^T\mathbf{A}^{-1}\mathbf{u}}.$$

In the second case, when we need to subtract formula it becomes

$$(\mathbf{A} + (-\mathbf{u}\mathbf{v}^T))^{-1} = \mathbf{A}^{-1} + \frac{\mathbf{A}^{-1}\mathbf{u}\mathbf{v}^T\mathbf{A}^{-1}}{1 - \mathbf{v}^T\mathbf{A}^{-1}\mathbf{u}}.$$

We can naturally use these formulas for our above equations. In the first step, we set $\mathbf{A} = \left(1 - \frac{r(z_{i,k})}{sp_k}\right)\Sigma_k$ and $\mathbf{u} = \mathbf{v} = \sqrt{\frac{r(z_{i,k})}{sp_k}}(X_i - \mu_k)$, in the second then $\mathbf{A} = \overline{\Sigma}_k$ and $\mathbf{u} = \mathbf{v} = (\mu_k - \mu_k^{old})$. Exact incremental formulas, where for better clarity we denote $\Delta sp_k = \frac{r(z_{i,k})}{sp_k}$ are of the form

$$\overline{\Lambda}_k = \frac{1}{(1 - \Delta sp_k)}\Lambda_k^{old} - \frac{\frac{\Delta sp_k}{(1-\Delta sp_k)^2}\Lambda_k^{old}(X_i - \mu_k)(X_i - \mu_k)^T\Lambda_k^{old}}{1 + \frac{\Delta sp_k}{(1-\Delta sp_k)}(X_i - \mu_k)^T\Lambda_k^{old}(X_i - \mu_k)} \tag{1.21}$$

$$\Lambda_k = \overline{\Lambda}_k + \frac{\overline{\Lambda}_k(\mu_k - \mu_k^{old})(\mu_k - \mu_k^{old})^T\overline{\Lambda}_k}{1 - (\mu_k - \mu_k^{old})^T\overline{\Lambda}_k(\mu_k - \mu_k^{old})} \tag{1.22}$$

where Λ_k^{old} refers to the last approximation of Λ_k.

A similar approach is possible if the value of the determinant is needed. In this case, we use matrix determinant lemma, which can be found in Harville [1998]. It states that for invertible squared matrix $\mathbf{A}$ and vectors $\mathbf{u}, \mathbf{v}$ of corresponding dimensions, it holds that

$$|\mathbf{A} + \mathbf{u}\mathbf{v}^T| = (1 + \mathbf{v}^T\mathbf{A}^{-1}\mathbf{u})|\mathbf{A}|.$$

Now, we can use the same approach as for the computation of the inverse covariance matrix, even apply the same substitutions for $\mathbf{A}, \mathbf{u}$ and $\mathbf{v}$ in both steps. This provides us with equations

$$\overline{|\Sigma|}_k = (1 - \Delta sp_k)^d|\Sigma|_k^{old}\left(1 + \frac{\Delta sp_k}{1 - \Delta sp_k}(X_i - \mu_k)^T\Lambda_k^{old}(X_i - \mu_k)\right) \tag{1.23}$$

$$|\Sigma|_k = \overline{|\Sigma|}_k(1 - (\mu_k - \mu_k^{old})^T\overline{\Lambda}_k(\mu_k - \mu_k^{old})), \tag{1.24}$$

where we suppose d dimensional vectors, which we used in the first equation when applying basic determinant property that for $d \times d$ matrix $\mathbf{A}$ and scalar c it holds that $|c\mathbf{A}| = c^d|\mathbf{A}|$. We consider value $|\Sigma|_k^{old}$ as a last approximation of the according determinant.

It is important to note that the process of adaptation of Equations (1.16)-(1.19) presented in this subsection is exact. It uses no approximation. Therefore, the results should be exactly the same as those we would obtain by using original equations.

In [Sun et al., 2023, Subsection 4.2.2], Equation (1.21)-(1.24) are rewritten into more computationally convenient form, which we present as Equation (1.25) and Equation (1.26). In [Sun et al., 2023, Fig. 7], it can be seen that indeed these equations show smaller runtime than ones presented in Pinto and Engel [2015]. The rewriting procedure results in equations

$$\Lambda_k = \frac{1}{1-\Delta sp_k}\left(\Lambda_k - \frac{\Delta sp_k \Lambda_k (X-\mu_k)(X-\mu_k)^T \Lambda_k}{1+\Delta sp_k d_k}\right) \tag{1.25}$$

$$|\Sigma|_k = (1-\Delta sp_k)^d (1+\Delta sp_k d_k)|\Sigma|_k, \tag{1.26}$$

where d_k refers to square Mahalanobis distance of X from k-th cluster before parameter update. Note also that μ_k in Equation (1.25) and Equation (1.26) refers to not yet updated mean vector. These equations are used in Algorithm 7, which presents a very much similar update procedure as [Sun et al., 2023, Algorithm 1].

1.7 Outlier detection in incremental GMM

In this section, we provide a concise overview of outlier detection techniques within the framework of incremental Gaussian Mixture Models (GMM) algorithms. These approaches are integral to the solution discussed in Section 4.3.

Subsection 1.7.1 introduces the general domain of outlier and novelty detection. Subsequently, Subsection 1.7.2 details the specific techniques from this domain that will be utilized in developing our algorithm in Section 4.3.

1.7.1 Outlier and novelty definition

The terms *outlier* and *novelty* are highly connected to the content of Section 4.3. To ensure clarity in our discussions, we will briefly describe these concepts here. A more extensive discussion is provided in Attachment A.7.

Novelty detection, as defined in [Yang et al., 2021, Subsection 2.2] and [Faria et al., 2016, Sections 1 and 3], refers to the ability to identify unlabelled instances that differ significantly from known concepts and do not fit into any existing categories. These novel instances are typically prepared for further procedures such as incremental learning or analysis. In this thesis, novelty detection generally involves recognizing changes in the current concept that occur in regions of interest, necessitating updates to the model to account for concept drift. Specifically, we aim to identify new patterns resulting from concept drift and distinguish them from noise.

Outliers lack a precise definition. As shown in [Ayadi et al., 2017, Table 1], there are presented twelve distinct definitions of outliers, each from a slightly different perspective. For the purposes of this thesis, we adopt the definitions provided by [Wang et al., 2019, Section II A.] and [Yang et al., 2021, Subsection 2.5]. An outlier is defined as a point that is significantly dissimilar to other observed points or deviates from typical behaviour. It is important to note that the classification of a point as an outlier may change over time as new data samples are collected.

In this subsection, we have provided a general definition of these terms. A consistent mathematical formalization is not typically presented, which we will also follow in this thesis. However, an intuitive understanding of this formalization can be gained from the specific detection techniques discussed in Subsection 1.7.2.

1.7.2 Detection techniques

In this subsection, we present an overview of existing outlier detection techniques associated with incremental Gaussian Mixture Models (GMM) algorithms. These techniques will form a crucial part of the algorithm discussed in Section 4.3.

The advantage of the incremental GMM algorithm lies in its natural capability for outlier detection by assessing whether a sample fits within the modeled clusters. The GMM algorithm, being a probabilistic model, provides the probability of a sample belonging to each cluster. However, determining whether a sample is an outlier is not always straightforward. Hence, we explore several approaches for outlier detection within the GMM framework.

We use the term outlier for clarity and general understanding, though other terms might be appropriate depending on the context. A brief discussion of the term outlier and related domain is provided in Subsection 1.7.1.

In one dimensional case it is straightforward to use well-known 68-95-99.7 *rule*, which for usual value $c = 2$ or $c = 3$ determines by the equation

$$|X - \mu| \leq c\sigma \tag{1.27}$$

probability of X belonging to cluster with mean μ and standard deviation σ to approximately 95.5 respectively 99.7 percent. This approach is discussed for example in [Blázquez-García et al., 2021, Subsection 3.1]. Unfortunately, the direct use of this rule for multidimensional data is not possible since we need to consider the dependencies among dimensions. A possible generalization is described in [Bajorski, 2011, Section 5.7.2] and it is given by Mahalanobis distance (see Subsection 1.5.3), where the criterion is

$$\sqrt{(X-\mu)^T \Sigma^{-1} (X-\mu)} \leq c,$$

where Σ is the covariance matrix. The problem with this generalization is that it obviously does not satisfy the same percentage distribution for c as in single dimensional case. In fact, the probability decreases rapidly as can be seen in Figure 1.7, which follows [Bajorski, 2011, Figure 5.21]. Let $\mathbf{X}$ be distributed according to normal distribution $\mathcal{N}(\mu, \Sigma)$ it holds that

$$P\{d_M(\mathbf{X}, \mu) \leq k\} = \chi^2_d(k^2), \tag{1.28}$$

where d_M represents the Mahalanobis distance and χ^2_d is the cumulative distribution function of the chi-squared distribution function with d degrees of freedom. This fact is formulated in Lemma 5, which follows [Bajorski, 2011, Property 5.9].

Lemma 5. *Let* $\boldsymbol{X}$ *be distributed according to* $\mathcal{N}(\mu, \Sigma)$*, where* Σ *is positive definite covariance matrix. Then the random variable* $(X-\mu)^T\Sigma^{-1}(X-\mu)$ *has the chi-squared distribution with* d *degrees of freedom.*

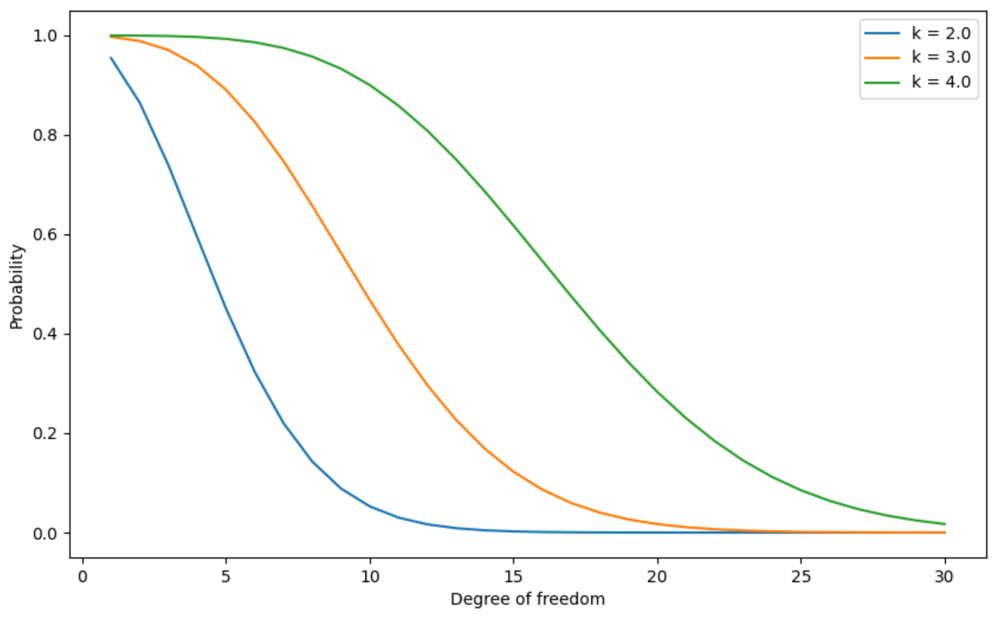


Figure 1.7: Illustration of changing probability using generalized 68-95-99.7 rule in higher dimensions. Presents probabilities presented in Equation (1.28) for $k \in \{2, 3, 4\}$ as function of the degree of freedom (dimension).

Proof. This proof uses the intuitive fact that the affine projection of normal distribution is again a normal distribution, this proposition is stated in [Bajorski, 2011, Property 5.5]. From this property can be derived that $\mathbf{Z} = \Sigma^{-\frac{1}{2}}(\mathbf{X} - \mu)$ is again normally distributed with mean equals to 0 and unity covariance matrix. Note also that we used positive definiteness of the Σ for existence of $\Sigma^{-\frac{1}{2}}$. Now, we can observe that

$$(X - \mu)^T \Sigma^{-1} (X - \mu) = \mathbf{Z}^t \mathbf{Z} = \sum_{i=1}^{d} \mathbf{Z}_i^2,$$

where $\mathbf{Z}_i$ stands for i-th element of $\mathbf{Z}$ and since each of this elements follows $\mathcal{N}(0, 1)$, the definition of a chi-square distribution with d degrees of freedom is exactly the distribution given by $\sum_{i=1}^{d} \mathbf{Z}_i^2$. $\square$

One of the leading approaches for outlier detection is based on this generalization. The process is described for example in [Pinto and Engel, 2017, Section 2.1] or [Sun et al., 2023, Section 2] and it is based on simple criterion of testing if the cluster statistics satisfy

$$(X - \mu_i)^T \Sigma_i^{-1} (X - \mu_i) \leq \chi^2_{d,1-\beta} \quad \forall i,$$

where $\chi^2_{d,1-\beta}$ is the $1 - \beta$ percentile of a chi-square distribution with d degrees of freedom for a dimension d of feature vector X and given parameter β.

We mention also approaches presented in [Engel and Heinen, 2010, Section 2]. This procedure prefers more parametric-based detection, where an arriving sample X is identified as non suiting to the model, based on *novelty parameter* and the following inequality

$$P(X|z_{i,k}) < \frac{novelty\ parameter}{(2\pi)^{d/2}\sqrt{|\Sigma_k|}} \quad \forall k.$$

This approach may give us a straight way how to affect the number of outliers, which can be possibly beneficial in the scenario where we have lots of information about the data.

1.8 Cluster management

In this section, we will discuss cluster management in incremental Gaussian Mixture Model (GMM) algorithm which is an essential part of the Algorithm 5, which represents our solution for problem stated in Section 2.3.

A management of clusters is an important part of the incremental clustering algorithms, especially when we consider adaptive number of them. Generally speaking the following operations on the clusters are the ones we think of for the proper incremental cluster algorithm.

- Update and deletion of the existing clusters
- Creation of a new cluster

GMM, a probabilistic algorithm discussed in Section 1.6, serves as the foundational process for Algorithm 5. In this section, we introduce various existing approaches to cluster management related to GMM, which will be the basis for our work in Section 4.3.

We divide the discussion of cluster management into two parts. First, in Subsection 1.8.1, we provide a brief overview of methods and approaches for managing existing clusters. Then, in Subsection 1.8.2, we address the challenges associated with creating new clusters.

1.8.1 Management of existing clusters overview

A crucial aspect of Algorithm 5 is cluster management, which is essential for achieving the objectives outlined in Section 2.3. In this subsection, we provide a brief overview of the approaches we used as starting points in developing our solution. The information presented is especially important for Subsection 4.3.5.

We discuss various strategies for cluster management, noting that not all selected approaches can be directly applied to our scenario. However, each approach offers valuable insights relevant to our problem.

In [Pinto and Engel, 2015] same as in [Heinen, 2011, Subsection 2.5.3] the approach of creating a new component is connected with two aspects, the number of arrived samples since the cluster was made, denoted by v_k and the importance for the model, which is represented by sp_k. Given parameters v_{min} and sp_{min}, we check a delete criterion of the form

$$v_k > v_{min} \quad and \quad sp_k < sp_{min}. \tag{1.29}$$

If the criterion is met then the cluster k is deleted and parameters in the model are adjusted accordingly. For dealing with problem of too much accumulation in the parameters which can negatively affect the model, it is possible to introduce the parameters α and β which work as follows

$$sp_k^{new} = P(z_{i,k}|X_i) + \alpha sp_k, \tag{1.30}$$

where $0 < \alpha < 1$ and

$$\mathit{if} \ \sum_{k=1}^{K} sp_k \geq \beta sp_{max} \ \ \mathit{then} \ \ sp_k^{new} = \gamma sp_k \ \ \forall k, \tag{1.31}$$

where $0 < \gamma < 1$ and sp_{max} is large value estimating

$$\lim_{t\to\infty} \sum_{j=1}^{n} \sum_{k=1}^{K} sp_k^j = sp_{max}.$$

In [Diaz-Rozo et al., 2018, Section II. C. D.] there is described an approach based on window strategy (see Subsection 1.3.1), where we generally consider some statistics of the latest samples. In their work, they deal with concept drift by a modification of detect and retrain idea. The detection algorithm is based on the number of outliers in the GMM procedure supposing that increase of the outlier detections is connected with data distribution change. This means that when a new concept is arriving and we need to retrain our current outdated model, we do it by common procedure but with the difference that we are using only the latest samples. Using Chernoff bounds we can set a minimum number of instances that fit in the model per defined window and if this number is not met the concept drift is detected and a retraining procedure is launched. More details on this approach can be seen in Attachment A.5.

An approach for handling overly similar clusters generated during the process can be found in [Sun et al., 2023, Section 3]. This procedure involves creating new clusters for samples that do not fit the current model and initiating a removal process periodically. In the first step, clusters with a prior probability below a certain threshold are removed. This step is similar to the criterion in Equation (1.29), but the authors have enhanced it with an adaptive threshold that changes based on the number of components. Specifically, when there are more components, the threshold is lower, allowing for smaller priors. Conversely, when there are fewer components, the threshold becomes more restrictive. By the extensive experiments on the bivariate normal distributions (discussed in Attachment A.8) the adaptive threshold is set to be

$$prior_{threshold} = \frac{\pi}{10} e^{-\frac{\pi}{10}K}.$$

After prior threshold-based removal, we sort the clusters in a descending manner with respect to the prior probability. The next step is a study of the remaining clusters' overlap. For this task, the logical matrix LM of the shape $K \times K$ is used. Its element on position i, j is of the form

$$LM_{i,j} = \begin{cases} 1 & \mathit{if} \ (\mu_j - \mu_i)^T \Sigma_i^{-1} (\mu_j - \mu_i) \leq \chi^2_{d,0.95} \ \mathit{and} \ i \neq j \\ 0 & \mathit{otherwise}, \end{cases} \tag{1.32}$$

where $\chi^2_{d,0.95}$ is chosen accordingly to the 95% confidence level, which is considered quite as a convention. The way LM is defined can be interpreted as if $LM_{i,j} = 1$ then the center of j-th component μ_j is located in the confidence region of component i. By extensive experiments, they concluded that it is beneficial to remove the clusters whose centers are in the confidence interval of another cluster. To be more precise, the process runs as follows. We sum up each column of LM, lets denote it as COL_{LM}, then we sequentially remove the cluster i such that $COL_{LM}(i) \geq 2$ and $sp_i = min(\{sp_j \ ; \ COL_{LM}(j) \geq 2\})$. When there is no column in LM which sums up to more than 1 we proceed with those that sum up to 1 where we again start with ones having the least prior probability and we continue removing until LM is a matrix made only by zeros. The whole described algorithm can be seen in [Sun et al., 2023, Fig. 1].

For the sake of completeness, we also mention that there are approaches for splitting the clusters if their volume is too great and merging if we believe that they represent the same distribution. Merging is described in [Acevedo-Valle et al., 2017, Algorithm 2] and is given by equations of the form

$$\begin{aligned}
f_1 &= \frac{sp_1}{sp_1 + sp_2}, \quad f_2 = \frac{sp_2}{sp_1 + sp_2} \\
sp_{new} &= f_1 + f_2 \\
\mu_{new} &= f_1\mu_1 + f_2\mu_2 \\
\Sigma_{new} &= f_1\Sigma_1 + f_2\Sigma_2 + f_1 f_2(\mu_1 - \mu_2)(\mu_1 - \mu_2)^T.
\end{aligned}$$

Approach for the splitting is presented in Bouchachia and Vanaret [2011] but we do not describe the technique since we will not consider the splitting of clusters in the algorithm.

1.8.2 Creation of a new cluster

In this subsection, we discuss the possibilities of creating a new cluster at the moment when arriving sample is detected as not well-fitting into current distribution. This is important part of our algorithm presented in Section 4.3 and it is especially connected with Subsection 4.3.4. We will evaluate various approaches and their respective advantages and disadvantages.

When encountering a sample that does not conform to the current distribution, we generally have two options. The first is to classify the sample as an outlier but retain it temporarily to assess whether it might represent the arrival of a new concept. The second option presumes that the sample indicates a new concept, subsequently testing this hypothesis over time to confirm or refute it (see Subsection 1.7.1). The literature is divided on which approach is superior. For example, [Heinen, 2011, Subsection 2.5.3] and [Sun et al., 2023, Subsection 2.2] adopt the second approach, creating a new cluster for each non-fitting sample while periodically applying the deletion criterion in Equation (1.29). Conversely, in the same subsection of the above literature other researcher is cited claiming that a single sample lacks sufficient information to justify the formation of a new cluster, an argument reflected in Aletti and Micheletti [2017].

The second solution involves establishing a retention set S_{ret}, where samples identified as non-conforming are stored. When a new sample also does not fit, its similarity to the samples in S_{ret} is assessed. If the similarity exceeds a predefined threshold, a new cluster is formed for these two points. Each data point in S_{ret} has an expiration date, beyond which it is discarded to prevent excessive memory use and the formation of noise clusters from an overabundance of samples in S_{ret}.

For our scenario we chose the second approach, using retention sets, which seems more practical as it accommodates novelty detection while ensuring the classification model is sufficiently adaptive to the new concept. This typically involves assuming a sample is an outlier initially, rather than immediately categorizing it as the start of a new concept. Notably, as the model processes a reasonable number of samples, the difference between these approaches may become negligible. Both methods share the challenge of determining the optimal time to remove a sample from the retention set or to reassess its significance (sp_k). Current literature often treats this as a hyper-parameter, which may not be ideal for statistical interpretability. Further discussion on this topic is presented in Subsection 4.3.4.

A critical aspect of new cluster creation is the parameter setting, particularly concerning the covariance matrix used to determine similarity among elements in the retention set. This issue is examined in Subsection 4.3.4.

2. Problem formalization

The primary objective of this chapter is to describe and formalize the aim and the working environment of this thesis.

Initially, we provide a brief motivation and formalization of the general scenario in Section 2.1. In Section 2.2 we follow up on general definitions from Section 1.1 and in connection with the formalized scenario we define crucial aspects of concept drift. Finally, in Section 2.3, we summarize the working environment along with aims and its underlying assumptions.

2.1 Formalization of scenario

In Subsection 2.1.1, we present the motivation of a scenario based on real-life problem, and then in Subsection 2.1.2 we present general formalization of this environment, which we will be further extended and detailed in Section 2.2.

2.1.1 Scenario motivation

Malicious software, commonly referred to as malware, is intentionally designed to disrupt targets such as users, computers, or networks. A key aspect of malware detection is the presence of features that enable rapid and accurate identification of malicious software. Examples of such features include mutexes used by known malware families, specific filenames, or registry keys. However, these features often change rapidly, sometimes with each new iteration of the malware, leading to a decline in detection model performance. This decline occurs because models overly rely on these highly informative features, which can become misleading when they assume previously unseen values.

Conversely, certain aspects of malware, such as the type of connection, the protocol used, or monetization methods, are difficult or impossible to alter. These more stable features can provide valuable information for detection. Unfortunately, machine learning algorithms may overlook these features, focusing too much on the highly informative but volatile ones. When these volatile features change, the overall detection accuracy can drop significantly.

This decline in performance is further exacerbated by a phenomenon specific to malware detection known as *label delay*. This refers to the typical lag between the need to classify new incoming software and the time it takes to definitively determine whether these samples are malicious or benign. This delay complicates the detection process and is an important factor to consider in improving malware detection strategies.

2.1.2 Scenario formalization

In this subsection, we formalize the process outlined in Subsection 2.1.1. We will still use a bit vague symbols as $<$ and $\ll$, which we will further detail in Section 2.2. A sketch of the scenario can be seen in Figure 2.1.

Suppose that we have label y and feature vector $X = (X_d, X_s)$, which consists of two sets of features X_d and X_s, where the former are indicative but drifting features and the latter are less reliable and more stable features. Also suppose time t_0 and distributions $P_{t<t_0}(y, X)$ and $P_{t>t_0}(y, X)$, which are followed by feature vectors and labels before and after time t_0. Let $P^d_{t<t_0}(X_d, y)$ and $P^s_{t<t_0}(X_s, y)$ be dimensional restrictions of $P_{t<t_0}(y, X)$ according to above sets of features. Similarly we consider $P^d_{t>t_0}(X_d, y)$ and $P^s_{t>t_0}(X_s, y)$.

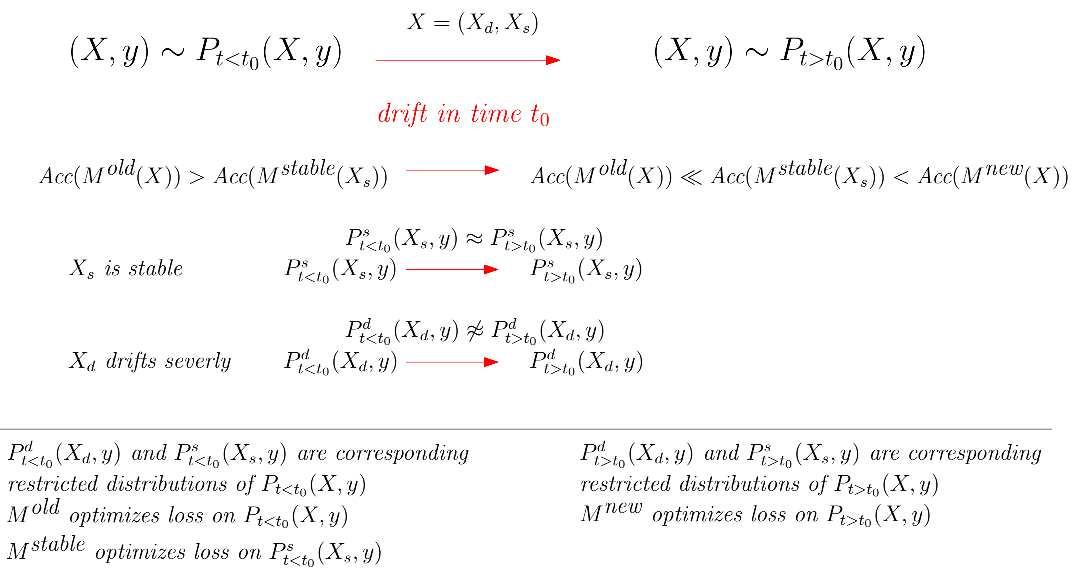

$(X, y) \sim P_{t<t_0}(X, y)$
$X = (X_d, X_s)$
$(X, y) \sim P_{t>t_0}(X, y)$
drift in time t_0
$Acc(M^{old}(X)) > Acc(M^{stable}(X_s))$
$Acc(M^{old}(X)) \ll Acc(M^{stable}(X_s)) < Acc(M^{new}(X))$
X_s *is stable*
$P^s_{t<t_0}(X_s, y) \approx P^s_{t>t_0}(X_s, y)$
$P^s_{t<t_0}(X_s, y)$
$P^s_{t>t_0}(X_s, y)$
X_d *drifts severly*
$P^d_{t<t_0}(X_d, y) \not\approx P^d_{t>t_0}(X_d, y)$
$P^d_{t<t_0}(X_d, y)$
$P^d_{t>t_0}(X_d, y)$
$P^d_{t<t_0}(X_d, y)$ *and* $P^s_{t<t_0}(X_s, y)$ *are corresponding restricted distributions of* $P_{t<t_0}(X, y)$
M^{old} *optimizes loss on* $P_{t<t_0}(X, y)$
M^{stable} *optimizes loss on* $P^s_{t<t_0}(X_s, y)$
$P^d_{t>t_0}(X_d, y)$ *and* $P^s_{t>t_0}(X_s, y)$ *are corresponding restricted distributions of* $P_{t>t_0}(X, y)$
M^{new} *optimizes loss on* $P_{t>t_0}(X, y)$

Figure 2.1: General scheme of the scenario.

Since we are interested in label prediction and connected model performance we add loss function l. For this loss, we assume the following models

$$\begin{aligned} &M^{old} \text{ optimizes } l \text{ with respect to } P_{t<t_0}(X, y)\\ &M^{stable} \text{ minimizes } l \text{ with respect to } P^s_{t<t_0}(X_s, y)\\ &M^{new} \text{ minimizes } l \text{ with respect to } P_{t>t_0}(X, y). \end{aligned}$$

Important property based on scenario motivation is that informally speaking the distribution P^s changes before and after time t_0 only slightly and contrary the distribution P^d at same time suffers from significant drift. This property is in Figure 2.1 denoted as

$$\begin{aligned} P^s_{t<t_0}(X_s, y) &\approx P^s_{t>t_0}(X_s, y)\\ P^d_{t<t_0}(X_d, y) &\not\approx P^d_{t>t_0}(X_d, y). \end{aligned}$$

During formalization, we connect this property with model performance as the most common objective. Assume loss function $l_A : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}_+$, which quantifies the accuracy of a model, the smaller the loss, the higher the accuracy. Models satisfy

$$\begin{aligned} &\mathbb{E}_{(X,y)\sim P_{t<t_0}(X,y)} l_A(M^{old}(X), y) < \mathbb{E}_{(X,y)\sim P^s_{t<t_0}(X_s,y)} l_A(M^{stable}(X), y)\\ &\mathbb{E}_{(X,y)\sim P^s_{t>t_0}(X_s,y)} l_A(M^{stable}(X), y) \ll \mathbb{E}_{(X,y)\sim P_{t>t_0}(X,y)} l_A(M^{old}(X), y)\\ &\mathbb{E}_{(X,y)\sim P_{t>t_0}(X,y)} l_A(M^{new}(X), y) < \mathbb{E}_{(X,y)\sim P^s_{t>t_0}(X_s,y)} l_A(M^{stable}(X), y). \end{aligned}$$

Since we connect the loss function with the accuracy we denote this property in Figure 2.1 as

$$\begin{aligned} Acc_{t<t_0}(M^{old}) &> Acc_{t<t_0}(M^{stable}) \\ Acc_{t>t_0}(M^{stable}) &\gg Acc_{t>t_0}(M^{old}) \\ Acc_{t>t_0}(M^{new}) &> Acc_{t>t_0}(M^{stable}). \end{aligned}$$

Note that a bit vague $<$ and $\ll$ will be replaced in Definition 8 by introduced parameters and consequent terms of *stable relevant information* and *severe concept drift*.

The above process in connection with our scenario implies that before the model manages to learn M^{new}, which is optimal on $P_{t>t_0}(X, y)$ it is beneficial to use already learned M^{stable} instead of M^{old}. This is the fundamental idea followed in this thesis.

To complete formalization based on Subsection 2.1.1 we need to formalize *label delay*. This is done naturally in connection with time series. We say that there is label delay l if it holds that considering sample set $\{(X_i, y_i)\}$ model has no access to label y_i unit it predicts labels of all X_j for $j \leq i + l$.

2.2 Concept drift specifics

In this section, we will present our definition of highly important terms *severe concept drift* and *relevant stable information*, which are motivated by Section 2.1 and are crucial for formalization of the problem. We will also define sudden drift as the underlying type of concept drift considered in our scenario.

2.2.1 Severe drift

Our scenario is specific in that serious drifts occur only in some features, which can corrupt predictions despite the presence of stable information that could yield better results. Given our focus on severe drifts and stable relevant information, it is essential to define these rigorously. To align with the formalization in Section 2.1, we define these concepts using a comparison of the loss. The intuition behind this approach as generally tracking the decision boundary can be found in Subsection 1.1.2.

Definition 8. *Let S_{0,t_1} be a sample set consisting of samples of a form $(X, y) \in \mathbb{R}^d \times \mathcal{Y}$ and containing concept drift in time $t_0 \in \mathbb{N}$, $0 < t_0 < t_1$. Also let $P_{0,t_0}(X, y)$ and $P_{t_0+1,t_1}(X, y)$ be distributions according which are generated sets S_{0,t_0} and S_{t_0+1,t_1}. Consider models M^{old} and M^{new} for which it holds that they optimize loss function l with respect to distributions $P_{0,t_0}(X, y)$ and $P_{t_0+1,t_1}(X, y)$ respectively. Let $l_A : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}_+$ be a loss function quantifying accuracy, then for given constants $\tau^{severe}, \tau^{stable} \in (0, 1), \tau^{severe} \leq \tau^{stable}$:*

- *we call the concept drift in time t_0 severe if it holds that*

$$\tau^{severe} \mathbb{E}_{(X,y)\sim P_{t_0+1,t_1}} l_A(M^{old}(X), y) \geq \mathbb{E}_{(X,y)\sim P_{t_0+1,t_1}} l_A(M^{new}(X), y).$$

- *we say that there exist stable relevant information for concept drift in time t_0 if there exists feature subset $S \subseteq \{1, \dots, d\}$ for which it holds that*

$$\tau^{stable}\mathbb{E}_{(X,y)\sim P_{t_0+1,t_1}} l_A(M^{stable}(R_S(X), y) \leq \mathbb{E}_{(X,y)\sim P_{t_0+1,t_1}} l_A(M^{new}(X), y),$$

  where function $R_S : \mathbb{R}^d \to \mathbb{R}^{|S|}$ preserves all dimensions whose index is in S and cuts of all which are not. Model M^{stable} optimizes loss l with respect to distribution $P^S_{0,t_0}(R_S(X), y)$, which is created from $P_{0,t_0}(X, y)$ by taking into account only dimensions of X which are in S.

In Definition 8 we used general threshold factors τ^{severe} and τ^{stable} as accuracy drop constants. In this thesis, we consider the appropriate value of τ^{severe} such that it decreases accuracy under 80% and stable parameter τ^{stable} such that accuracy does not decrease under 90%, both with respect to optimal accuracy. Setting of these parameters can differ according to specific aims and scenarios. Note also that in Definition 8 we connected the terms with the model, which can be considered as in some sense optimal. We believe that this is necessary for the possibility of proper definition but typically it is not possible to determine needed distributions exactly and we work just with their approximation. Note also that in the definition we intentionally did not use a specific loss function since in the thesis next to the usual percentage of correct predictions we will also consider other scores as F1-score (see Section 5.6).

2.2.2 Sudden drift

Especially for dealing with the concept drift it turned out that it can be beneficial to divide process of the drift into several types. These are highly motivated by real-life datasets. The most common types are sudden, gradual, incremental, and reoccurring drift. We will define sudden drift as underlying for our scenario. The remaining types are described in Attachment A.1.

The above mentioned types are mostly based on speed of change and permanency of arriving concepts. The sudden drift occurs if a new concept quickly and completely replaces the original concept. Its illustration can be seen in Figure 2.2, which follows [Lu et al., 2019, Fig. 4].

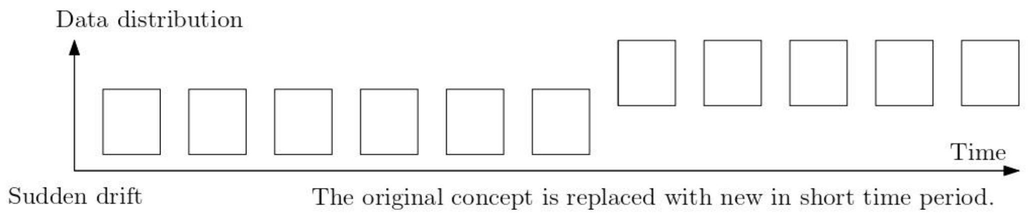

Data distribution
Time
Sudden drift
The original concept is replaced with new in short time period.

Figure 2.2: Illustration of the concept drift types according to way the distribution of samples in the stream changes over time.

For the definition of the concept drift types, we assume a general setup from [Hinder, 2022, Section 2]. We can understand this setup as similar to ours, where we consider the collection of distributions $P_i(X, y)$. These distributions may change in time and are indexed by ordering $t \in [1, \dots, n]$. It holds that $(X_t, y_t) \sim P_i(X, y)$. Based on this we give our definition of the sudden concept drift in Definition 9.

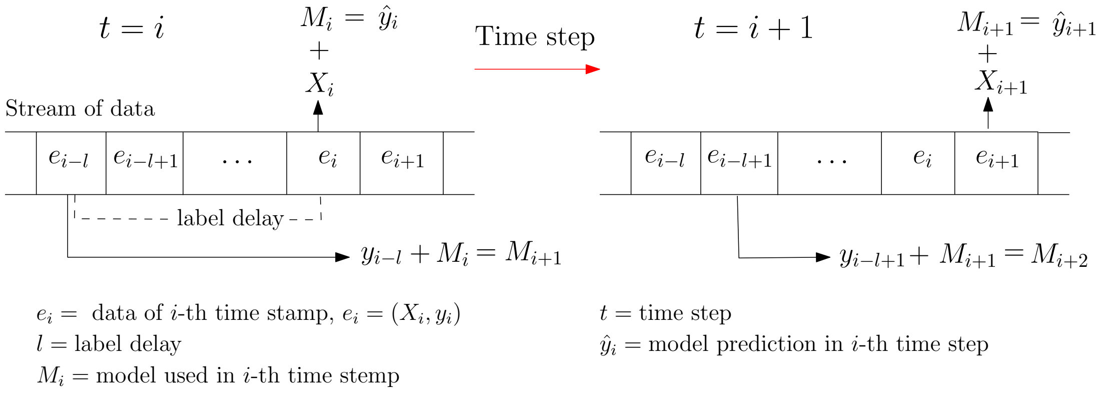
Definition 9. *Considering notation similar as in Definition 2 and problem setup as above (Subsection 2.2.2), for given sample set S_{0,t_n} we call a concept drift sudden if there exist time point $t_0 \in \mathbb{N}$, $0 < t_0 < t_n$ and distinct probability distributions $P_{\leq t_0}(X,y)$ and $P_{>t_0}(X,y)$ such that $P_t(X,y) = P_{\leq t_0}(X,y)$ for all $t \leq t_0$ and $P_t(X,y) = P_{>t_0}(X,y)$ for all $t > t_0$.*

It is important to note that, for testing on synthetic datasets, we primarily consider those with multiple sudden drifts, as dictated by the problem formalization presented in this chapter. Such datasets are more suitable for analysis. We create a synthetic dataset tailored to our scenario in Subsection 5.1.2. However, we must recognize that the definition of concept drift types aims to approximate the main prevailing aspects of the concept, while real-world datasets typically exhibit a complex mixture of properties.

2.3 Formalization of the problem

In this section, we summarize and complete the formalization presented in this chapter. We will present the scenario, our aims, and assumptions, which follow from real-life practice. In addition to previous sections, we also state the overall goal in the context of the scenario.

The problem we are dealing with takes place in a stream scenario, which we define in a way that there is a given dataset D consisting of $n \in \mathbb{N}$ samples e_i, where each sample $e_i = (X_i, y_i)$ consists of feature vector $X \in \mathbb{R}^d$ and label $y \in \mathcal{Y}$. We assume that in time step i we have classified all feature vectors X_t for $t \leq i$ and observe labels with delay l, which means labels y_j for $j \leq i - l$. This is our general definition of the *data stream scenario*, which is generally similar to one presented in [Bayram et al., 2022, Subsection 4.1].



$t = i$
$M_i = \hat{y}_i$ + X_i
Time step
$t = i + 1$
$M_{i+1} = \hat{y}_{i+1}$ + X_{i+1}
Stream of data
e_{i-l}
e_{i-l+1}
...
e_i
e_{i+1}
label delay
$y_{i-l} + M_i = M_{i+1}$
$y_{i-l+1} + M_{i+1} = M_{i+2}$
e_i = data of i-th time stamp, $e_i = (X_i, y_i)$
l = label delay
M_i = model used in i-th time stemp
t = time step
$\hat{y}_i$ = model prediction in i-th time step

Figure 2.3: Illustration of our scenario. Model evolution in data stream.

The overall goal is to find a sequence of models $\{M_t\}_{t=1}^{n}$, where model M_i predicts labels of the feature vector X_i received in time step i. We illustrate this scenario in Figure 2.3. We look for the sequence of models for which it holds that $\mathbb{E}_{(X_i,y_i)\sim D} l_A(M_i(X), y)$ is minimal for the accuracy quantifying loss l_A. Note that the sequence of models typically does not imply significant changes in a single step. Instead, it generally involves slight adjustments to the current model by incorporating newly arrived sample, as illustrated in Figure 2.3.

Our scenario is motivated by malware detection, specifically by sudden and severe changes in highly decisive features but also with non-trivial information that remains stable during the drift (see Section 2.1). All this motivates our assumptions on dataset D which are that

- dataset D contains a sudden severe concept drifts as defined in Definition 8
- for each concept drift occurring in dataset D there exists stable relevant information according to Definition 8.

In Definition 8, we relate the terms severe concept drift and stable relevant information to a drop in prediction accuracy compared to an optimal model. This comparison is moderated by accuracy decrease factors that may vary depending on the specific situation and objectives. A more detailed discussion of these definitions is provided in Subsection 2.2. Generally, in this thesis, we consider a drop in accuracy below 80% indicative of severe drift, and we consider information stable and relevant if its accuracy does not drop below 90%, both relative to the optimal accuracy before the drift.

3. Related work

In this chapter, we discuss prior methods developed for classification with concept drift (Chapter 2), highlighting its shortcomings and illustrating the improvements our work focuses on. This will be demonstrated using the evaluations presented in Section 5.2 and Section 5.3. The related work primarily concentrates on the Dynamic Weighted Majority (DWM) algorithm, a well-known and robust method addressing a problem similar to the one presented in Section 2.3.

In Section 3.1 we give a brief insight into existing related work and highlight some reasons why we chose the DWM algorithm as main underlying procedure. In Section 3.2, we illustrate some of the related work limitations in case of sudden severe drift in a subset of features (Section 2.3). In Section 3.3, we outline our solutions designed to address these identified shortcomings.

3.1 Related work introduction

In this section, we briefly introduce related work relevant to the problem formalized in Section 2.3.

Many algorithms and procedures aim at classification in data stream scenarios, but to our knowledge, no approach focuses on the specific properties described in Chapter 2.

The data stream scenario with concept drift is a complex process without a single best strategy. However, there are two major approaches. The first approach involves creating a model that constantly evolves and updates according to the drift (Wankhade et al. [2020]). The second approach, known as *detect and retrain*, learns the current concept and, upon detecting drift, discards previous information and retrains on the latest samples (Ceschin et al. [2022]).

Generally, the detect and retrain strategy is considerably less computationally demanding if concepts remain relatively stable. Unfortunately, in our scenario, which assumes a highly drifting environment with non-negligible label delay, more adaptive algorithms seem preferable.

There is, in fact, a broad class of algorithms that combine elements of these two approaches: ensemble algorithms (Gomes et al. [2017]). These algorithms utilize a set of models that are added, removed, and modified during the process. Consequently, there exists a wide range of possible strategies to compose these models optimally. Among these algorithms, we chose the DWM algorithm, described in Subsection 1.2.1. This algorithm uses an ensemble of experts, each weighted according to its current accuracy. The ensemble is constantly updated, and outdated experts are discarded when a new concept is detected. We selected this algorithm as our main baseline due to its considerable accuracy, robustness, ability to adapt quickly to new concepts, and straightforward implementation.

3.2 The DWM accuracy drops

In this section, we show some results of DWM algorithm (see Subsection 1.2.1) that illustrate some shortcomings and their possible improvements, especially in relation to working with a dataset containing the severe concept drift, which is important part of our scenario formalized in Chapter 2.

3.2.1 DWM and severe drift

In this subsection, we review results from the experiments presented in Sections 5.3 and 5.2, demonstrating that changes in concepts can lead to unnecessary drops in accuracy. This discussion provides significant motivation for our proposed solution outlined in Section 3.3, which aims to improve accuracy during brief periods of new concept arrival.

The DWM algorithm is robust, and as noted in Kolter and Maloof [2007], can adapt fairly quickly to entirely new concepts. However, the adaptation period associated with the arrival of a new concept often results in a substantial decrease in prediction accuracy, as illustrated by the accuracy drops in Figure 5.2. To further highlight this issue, we evaluate the algorithm on our synthetic dataset in Section 5.3.

Our dataset, detailed in Subsection 5.1.2, is designed according to our scenario described in Section 2.1. It includes features that are highly decisive but subject to drift, as well as features that are less decisive but more stable. These aspects reflect the important characteristics of severe concept drift and relevant stable information (Definition 8).

In Section 5.3, we compare the accuracy of the DWM algorithm in two scenarios: one utilizing all features of the dataset, and the other excluding the highly indicative but severely drifting feature. The results align with the intuition presented in Subsection 1.1.3, indicating that removing the highly decisive but heavily drifting feature can enhance model performance during periods of concept drift. It is important to note that the evaluation depicts a scenario without label delay, which is a significant aspect of our scenario and can naturally amplify periods of accuracy decline.

The evaluation discussed in this subsection demonstrates that during periods of severe drift, focusing on less drifting parts of the feature vector may be advantageous. This insight is crucial to our proposed solutions, the general outline of which is presented in Section 3.3.

3.3 Solution outline

Although the techniques we employ in our solutions will vary, they adhere to a general framework, which we present in this section.

This framework is strongly motivated by the shortcomings highlighted in Section 3.2. We discussed how methods designed to address concept drift in data stream scenarios can experience significant and unnecessary accuracy drops during the arrival of new concepts. These issues are connected with overly relying on severely drifting components of the feature vector.

The stream scenario described in Section 2.3 is well-documented, with numerous approaches for addressing it, as detailed in Section 3.1. To build on existing research and provide a highly usable solution, we have developed an algorithm that supplies critical information about feature drift to the actual classification model. The focus of this improvement is on periods of severe drift, where, as illustrated in Subsection 3.2.1, current methods still exhibit shortcomings.

To maximize flexibility in selecting a classification model, our algorithm operates as an auxiliary procedure, treating the classifier as a black box. The drift information provided by our algorithm can then be used to enhance the classifier's drift response capabilities, which can be implemented in various ways depending on the model. To maintain generality, we have chosen to use this information as a binary indicator during testing, determining whether each feature should be used for classification.

By adopting this solution, we intersect with the field of feature selection, which is briefly described in Attachments A.3.

Results of evaluation (Section 5.3) reviewed in Subsection 3.2.1 provides insight into another aspect of our solutions, which involves examining subsets of features to determine if concept drift is present in any of them. Studying concept drift by tracking non-drifted parts of the data is complicated by feature dependencies, necessitating a search through all feature subsets—a problem that, according to Welch [1982], is NP-hard. Motivated by Subsection 1.1.3, we simplify this process by examining and solving the problem using specific subsets of features and their drift. We assume dependencies among features as follows:

- let $X \in \mathbb{R}^d$ be d-dimensional feature vector. We are provided with a partitioning of these d features into non-overlapping sets. We assume that all features within a single set are highly dependent, while each set of features is independent of the others.

In other words, we assume knowledge of which features are dependent and which are not. The correctness of this assumption does not invalidate the solution algorithms, which is crucial since feature dependencies in real-life datasets are typically complex. The procedures presented use the dependency assumption to create feature subsets where drift is tracked. Thus, when adapting the dataset to this assumption, we consider only highly dependent features that do not make sense to examine alone.

If the dependency assumption is incorrect and we falsely assume strong dependency, a stable feature might be marked as drifted due to its mistaken association with drifting features. Conversely, if we incorrectly assume two highly dependent features are independent, we might fail to detect the present drift. However, it is common to have some knowledge about feature dependencies, especially when focusing on strongly dependent ones. For completeness, we also assume that the partitioning based on feature dependencies remains constant throughout the process.

4. Proposed solutions

In this chapter, we present our proposed solutions for scenarios involving severe concept drift and relevant stable information (Chapter 2). The first solution is based on Hellinger concept drift detection (Section 1.3), and the second is based on the incremental Gaussian Mixture Model (GMM) algorithm (Section 1.6), both following the general outline described in Section 3.3.

Section 4.1 details the adjustments made to the Dynamic Weighted Majority (DWM) algorithm (Subsection 1.2.1) to align with our scenario. Section 4.2 presents the solution based on Hellinger concept drift detection. Finally, Section 4.3 describes the sub-procedure based on the GMM algorithm.

4.1 Modified DWM

In this section, we describe adjustments of DWM algorithm (see Subsection 1.2.1) in order to adapt it fully to our scenario presented in Section 2.3.

Two main aspects we must address are the possible label delay and the fact that, in the DWM algorithm, the true label of the i-th sample is used to decrease weights before these weights are used. This implies that information gained from a label is used to predict the same label. Additionally, we introduce a parameter t, representing the number of most recent labelled samples provided to the newly emerging expert. This adjustment is based on the reasonable assumption that the drift, indicated by the need for a new expert, actually occurred earlier due to the delay caused by the updating period p.

Algorithm 3 is utilized in all experiments involving the DWM algorithm. In Section 5.2 we demonstrate that setting $l = 0$ and $t = 1$ our modified algorithm (Algorithm 3) is equivalent to original DWM algorithm (Algorithm 1).This equivalence is shown by replicating results from Kolter and Maloof [2007]. For our environment, when referring to scenarios without label delay, it is appropriate to consider $l = 1$.

4.2 Solution based on concept drift detection

In this section, we present solution for problem formalized in Section 2.3, which follows general outline described in Section 3.3 and is based on the concept drift detection technique presented in Section 1.3.

Generally, we consider drift which is severe in some part of the feature vector but on the other hand, in others relevant information is preserved. Technique presented in this section aims to detect concept drift in the feature subsets and then for certain time period exclude them from the prediction process in order to give the model time to adapt on the new concept.

4.2.1 Hellinger distance proposed algorithms

In Section 1.3 we presented one of the possible ways for concept drift detection, which will now serve as the basis for our proposed solution. In this section,

Algorithm 3 Dynamic Weighted Majority with label delay

Notation

$\{(e_i, w_i)\}_{i=0}^{m}$ - Set of experts e_i and according weights w_i.

Λ_i - Global prediction in step i, $\Lambda \in \mathcal{Y}$.

$\lambda_{i,j}$ - Local prediction, made by e_j for X_i $\lambda \in \mathcal{Y}$.

σ - Sums of local predictions, $\sigma \in \mathbb{R}^{|\mathcal{Y}|}$.

Inputs

$\{(X_i, y_i)\}_{i=0}^{n}$ - Samples, feature vector $X \in \mathbb{R}^d$ and according label $y \in \mathcal{Y}$, where $\mathcal{Y} = \{1 \ldots, c\}$ for $c \in \mathbb{N}$.

β - Factor of decreasing weights, where $0 \leq \beta < 1$.

θ - Removing expert threshold.

p - Period of ensemble and weights update, $p \in \mathbb{N}$.

l - Label delay, $l \in \mathbb{N}$.

t - Learning parameter $t \in \mathbb{N}$, which sets how many latest labeled samples are keeping in memory for learning of new experts.

```
$m \leftarrow 1$
$e_m \leftarrow$ Create new expert
$w_m \leftarrow 1$
for $i \leftarrow 1, \ldots, n$ {over all samples} do
  $\sigma \leftarrow 0$
  for $j \leftarrow 1, \ldots, m$ {over all current experts} do
    $\lambda_{i,j} \leftarrow$ Classification of $X_i$ by $e_j$
    if $\lambda_{i-l,j} \neq y_{i-l}$ and $i$ mod $p = 0$ then
      $w_j \leftarrow \beta w_j$
    $\sigma_{\lambda_{i,j}} \leftarrow \sigma_{\lambda_{i,j}} + w_j$
  $\Lambda_i \leftarrow \text{argmax}_j \sigma_j$
  if $i$ mod $p = 0$ {the updating period is met} then
    $w \leftarrow$ Normalize weights $w$
    $\{(e, w)\} \leftarrow$ Remove experts using threshold $\theta$
    if $\Lambda_{i-l} \neq y_{i-l}$ {global prediction is wrong} then
      $m \leftarrow m + 1$
      $e_m \leftarrow$ Create new expert, provide him with $\{(X_k, y_k)\}_{k=i-l-t}^{i-l}$
      $w_m \leftarrow 1$
  for $j \leftarrow 1, \ldots, m$ do
    $e_j \leftarrow e_j$ with provided $(X_{i-l}, y_{i-l})$
  output $\Lambda$
```

we modify Algorithm 2 to be more suitable for our general setting presented in Section 2.3. We present this approach as Algorithm 4 and we will discus some of its properties.

The first difference from the original algorithm (Algorithm 2) lies in splitting the process into independent runs for each predefined feature subset and converting the algorithm into a data stream one. This conversion is achieved primarily by using memory efficiently, replacing the need to remember D_λ as a set of features with simple histograms, where we increment values in corresponding bins. This change reduces the adaptability of window sizes since one of the distributions is represented only by a histogram. Additionally, we adjust the number of samples moved at each step from the original window size to a single sample at a time (see Subsection 1.3.1), which, although more computationally demanding, provides a more detailed detection of the sudden concept drift.

Presented adjustment is significant. It raises the question of whether, on an intuitive level, it affects the process's functionality. Using the approach from Algorithm 2, the crucial step involves computing the difference in Hellinger distances $\epsilon(t)$. Comparing distances of P, differing by only a single sample, seems fundamentally flawed since we seek properties of whole distributions, typically undetectable by a single sample. Thus, we track differences directly between growing and sliding windows. Another change involves the reaction to drift detection. Given the assumption of the sudden concept drift, the breaking point between concepts is likely within the sliding window. Therefore, we discard this window's distribution and use a new window presumed to consist entirely of samples from the new concept.

The algorithm measures differences between distributions provided by two windows and checks for deviations from previously measured ones. This adaptive threshold comprises the mean and standard deviation of previous differences, setting the decision boundary to the mean plus a constant number of standard deviations. This is a single-dimensional case of the problem discussed in Section 1.7, specifically focusing on Equation (1.27), which we modify to:

$$X_\epsilon < \mu(X_\epsilon) + \gamma\sigma(X_\epsilon), \tag{4.1}$$

where γ is a given constant, and μ and σ are the mean and standard deviation of X_ϵ, a random variable representing the distribution differences. Unlike Ditzler and Polikar [2011], we do not detect reductions in the difference concerning observed differences, as our interest lies in detecting increases, which may possibly signal a drop in model prediction accuracy and potential model confusion. Thus, we replace $\pm\gamma\sigma(X_\epsilon)$ with $+\gamma\sigma(X_\epsilon)$. From Section 1.7, we derive information about possible γ values, expecting the differences to follow a Gaussian distribution. Our modification in Equation (4.1) adjusts the percentages by halving the probability of being out of bounds, considering only one tail of the symmetric distribution, approximately 97.75% for $\gamma = 2$ and 99.85% for $\gamma = 3$.

Estimating the mean and standard deviation raises the question of the sample size required for accurate estimation. In our experiments on synthetic data, we address this by setting the sample size to 30, inspired by the well-known *rule of thumb* for estimating Gaussian distributions (Krithikadatta [2014]), though the differences do not meet the assumption of independent sampling. Another concern is that false detections occurring shortly before a real drift can lead to overestimation of parameters, reducing sensitivity to future drifts. To evaluate the algorithm's performance, Section 5.4 provides an evaluation focused on its ability to efficiently detect severe concept drift.

4.3 Solution based on GMM algorithm

In this section, we present our second solution for problem stated in Section 2.3. The first proposed approach, presented in Section 4.2 was based on measuring the distance between two distributions. Contrary to this, the second solution presented in this section will focus on comparing distribution and the sample using incremental Gaussian Mixture Model algorithm.

Content of this section is as follows.

- In Subsection 4.3.2, we present overall scheme of the algorithm.
- Subsection 4.3.3 focuses on determination whether new sample fits into the current cluster description and if it does then how should be incremented according parameters.
- Subsection 4.3.4 is devoted to process of new cluster creation
- Final Subsection 4.3.5 presents approach of spurious and obsolete clusters deletion.

These processes involve use of many hyper-parameters, whose proper setting will be an important part of each subsection.

4.3.1 Clustering algorithm

In this subsection, we present brief overview of the procedure and its motivation in context of the thesis.

To identify drifted parts of a feature vector in a streaming scenario, we employ clustering algorithms. This method aims to describe the current data distribution through a set of clusters. We chose this approach since it is one of the leading solutions for abnormal samples detection, which is, as briefly discussed in Subsection 1.7.1, very similar class of problems. Note that using this approach we naturally move from concept drift detection on distribution level presented in Section 4.2 to study the change for each arriving sample alone.

For our assumption of drifting data environment (see Section 2.3), it would not seem reasonable to fix the number of clusters. This is since each concept may have a different number of partitions which are needed for its description. This naturally berries the problem of adaptively adding and removing clusters. Consequently, our aim is not only to detect the drifted parts but also to track the changes and be able to determine when the drift regions again reach stability with the respect to classifier performance (see Section 3.3). This approach involves several tasks which have to be solved. Firstly, it is important to get to know the underlying clustering algorithm well. Therefore, in Chapter 1 we derived the Gaussian Mixture Model algorithm (GMM) from the very beginning non-incremental version presented in Section 1.3. Based on this, we then derived both usual and fast version of incremental GMM in Section 1.6.

Algorithm 4 Hellinger distance feature subset stream concept drift detection

```
**Inputs**

  • $\{S_i\}_{i=1}^D$ - set of feature subsets, where all sets are non-empty, without inter-
    ception and their union gives all features of feature space
  • $\gamma$ - standard deviation parameter for drift threshold, $\gamma \in \mathbb{R}$
  • $\{X_t\}_{t=1}$ - stream of feature vectors $X \in \mathbb{R}^d$
  • $w$ - window size $w \in \mathbb{N}$
  • $n_{min}$ - minimal number of collected differences needed for proper estimation of
    $\epsilon$ and $\sigma$, $n_{min} \in \mathbb{N}$

Generate a histogram $Q$ from $\{X_i\}_{i=1}^w$ with $b$ bins (usually $b = \lfloor\sqrt{w}\rfloor$) for each
feature, denote $Q_i$ as histogram for $i$-th feature
Generate sliding window by $W \leftarrow \{X_i\}_{i=w+1}^{2w}$, where $W_i$ is set of window values in
$i$-th feature
$\delta^{prev} \leftarrow \infty^D$   //for each feature subset set previous distance to infinity
for $t \leftarrow 2W+1, \ldots$ {over data stream} do
  for $i \leftarrow 1, \ldots, D$ {over feature subsets} do
    $W_{S_i} \leftarrow \{W_{S_i}, X_t^{(S_i)}\}$   //add feature subset values to according window
    if $|W_{S_i}| = w$ and $Q_{S_i} \neq \{\}$ then
      Generate a histogram $P$ from $W_{S_i}$ with $b$ bins for each feature of $S_i$
      Calculate feature Hellinger distance as follows
```

$$\epsilon = \frac{1}{|S_i|}\sum_{i=1}^{|S_i|}\sqrt{\sum_{j=1}^{b}\left(\sqrt{\frac{P_{j,i}}{\sum_{k=1}^{N}P_{k,i}}} - \sqrt{\frac{Q_{j,i}}{\sum_{k=1}^{N}Q_{k,i}}}\right)^2} \quad (1.8)$$

```
      Update the adaptive threshold
```

$$\hat{\epsilon} = \frac{1}{|\xi_i|}\sum_{\epsilon\in\xi_i}|\epsilon|$$

$$\hat{\sigma} = \sqrt{\frac{\sum_{\epsilon\in\xi_i}(|\epsilon| - \hat{\epsilon})^2}{|\xi_i|}}$$

```
      $\beta(t) \leftarrow \hat{\epsilon} + \gamma\hat{\sigma}$
      if $|\epsilon(t)| > \beta(t)$ and $|\xi_i| \geq n_{min}$ ($\hat{\epsilon}$ and $\hat{\sigma}$ are precise enough) then
        $\xi_i \leftarrow \{\}$   //drift is present in i-th feature subset
        $Q_{S_i} \leftarrow \{\}$   //reset all statistics for i-th feature subset
        $W_{S_i} \leftarrow \{\}$
      else
        Remove an oldest sample from $W_{S_i}$ and incorporate it into distribution $Q_{S_i}$

        $\xi_i \leftarrow \{\xi_i, \epsilon\}$
    if $|W_{S_i}| = w$ and $Q_{S_i} = \{\}$ then
      Generate a histogram $Q_{S_i}$ from $W_{S_i}$ with $b$ bins for each feature of $S_i$
      $W_{S_i} \leftarrow \{\}$
```

4.3.2 Algorithm scheme

As stated in introduction to this section, we will build procedure based on fast incremental GMM algorithm (see Section 1.6). In this subsection, we present overall structure of the process with focus on high-level cluster management.

Our algorithm is conceptually the most similar to one presented in [Sun et al., 2023, Fig 1] (see Subsection 1.8.1). Now, we present basic general parts of the procedure which is summarized as Algorithm 5. For K current clusters, retention set S_{ret} (Subsection 1.8.2) and newly arrived feature vector X, the general steps of the algorithm for feature vector restriction X^S given by predefined feature subset S will be as follows:

- Determine whether X^S is the outlier based on generalized 68-95-99.7-rule presented in Section 1.7. If X^S is not detected as outlier increment the model parameters accordingly. (Subsection 4.3.3)
- If X^S is detected as outlier then, check the squared Mahalanobis distance from each sample in S_{ret} if similarity is detected create a new cluster. (Subsection 4.3.4)
- If criterion is met, delete spurious clusters (Subsection 4.3.5) and adjust *sp* parameters.

This scheme is precisely outlined in Algorithm 5, which presents the general framework of the solution. It is important to note that the critical operations occur immediately after the arrival of a new sample. At this stage, we assess whether the sample fits well and whether it is appropriate to utilize specific parts of the feature vector. A feature vector subsect is considered unsuitable if it does not fit well or if it only fits into non-incorporated clusters (Subsection 4.3.3). Then we do not use this part of sample for prediction. The rest of whole process generally keeps accurate description of the current distribution.

4.3.3 Sample fit and update of mixture

In this subsection, we describe part of our solution based on GMM algorithm (Algorithm 5). We deal with determination of how well does newly arrived sample fit into existing mixture of clusters, whether is the sample incorporated by the classifying model and how to update the current parameters. Overall approach is described in Algorithm 6.

One of the main advantages of the GMM is the natural way of determination whether new sample fits into the mixture or should be considered as an outlier. This is due to the probabilistic nature of the algorithm. Therefore, we will use well known generalized 68-95-99.7-rule presented in Section 1.7. This can be seen in Algorithm 6, which represent a general scheme of testing how well is the sample described by the model and where parameters update takes place in our procedure. In the algorithm, we also determine whether feature subset should be used for classification or not. This is done by marking the feature restriction by *outlier* and *non-incorporated*, where the first refer for sample being completely new for us while the second states that it is the novelty, but we suppose that using it would probably worsens the classifier prediction. This part of the procedure is described in what follows.

Algorithm 5 Solution based on Gaussian Mixture Model algorithm

Inputs

- $\{S_i\}_{i=1}^{D}$ - set of feature subsets, where all sets are non-empty, without interception and their union gives all features of feature space
- $\{X_t\}_{t=1}$ - stream of feature vectors $X \in \mathbb{R}^d$
- t_b - number of samples used for mixture of clusters initialization, $t_b \in \mathbb{N}$ (Algorithm 8)
- β - parameter used for determining percentile of setting threshold for belonging to cluster (Algorithm 6)
- SP^L, SP^U - lower and upper bound on sum of sp for forgetting procedure, $SP^L < SP^U \in \mathbb{R}^+$ (Algorithm 9)
- n_{last}, c^{id} - incorporation parameters, c^{id} is needed number of similar prediction in n_{last} last samples from the cluster, $n_{last}, c^{id} \in \mathbb{N}$ (Algorithm 7)
- π_{min}^{K} - adaptive threshold for cluster prior probability, discussed in Subsection 4.3.5 (Algorithm 9)
- $\Lambda_k^{initial}$ - Diagonal inverse covariance matrix used at the beginning when there is no cluster (Algorithm 8)

```
for $t \leftarrow 1, \dots$ {over data stream} do
  for $i \leftarrow 1, \dots, D$ {over feature subsets} do
    Check whether $X_t^{S_i}$ fits into mixture of clusters and accordingly update parameters. In this procedure we also determine whether $X_t^{S_i}$ is outlier or fits into non-incorporated clusters only. Process follows Algorithm 6.
    if $X_t^{S_i}$ is marked as outlier then
      Run cluster creation procedure described in Algorithm 8.
    if $\sum_{k=1}^{K} sp_k^{S_i} \geq SP^U$ and $t > t_b$ then
      Check for spurious clusters according to Algorithm 9.
```

Algorithm 6 Interaction between X_i^S and corresponding mixture of clusters

```
Inputs
    • $X_i^S$ - feature vector, (subset of original feature vector) $X_i^S \in \mathbb{R}^{|S|}$
    • $\beta$ - parameter used for determining percentile of setting threshold for belonging to cluster
    • $\{C_k\}_{k=1}^K$ - set of $K \in \mathbb{N}$ clusters each consisting of $(sp_k, \pi_k, \mu_k, inc_k, \Lambda_k, var_k)$
for $k \leftarrow 1, \dots, K$ {over all components} do
    Compute how well does $X_i^S$ fit into $k$-th cluster by evaluating $d_k = (X_i^S - \mu_k)^T \Sigma_k^{-1} (X_i^S - \mu_k)$.
for $k \leftarrow 1, \dots, K$ for which it holds $d_k < \chi^2_{|S|,1-\beta}$ do
    Update $k$-th cluster by $X_i^S$ according to Algorithm 7.
if $\forall k \in \{1, \dots, K\}$ it holds that $d_k \geq \chi^2_{|S|,1-\beta}$ or $inc_k = False$ then
    Mark $X_i^S$ as non-incorporated.    //$X_i^S$ fits only to non-incorporated clusters
if $d_k \geq \chi^2_{|S|,1-\beta}$ $\forall k \in \{1, \dots, K\}$ then
    Mark $X_i^S$ as outlier.    //$X_i^S$ does not fit into any existing cluster
```

Cluster parameters update

In this paragraph, we present procedure of cluster parameter update as a part of our solution whose scheme is discussed in Subsection 4.3.2. The exact approach for cluster parameter update is presented in Algorithm 7.

Algorithm of cluster parameter update is the most similar to [Pinto and Engel, 2015, Algorithm 2] and [Sun et al., 2023, Algorithm 1], whose derivation is presented in Subsection 1.6.2. We add two specific sets of parameters, which are used for tracking variances, whose importance can be seen in Subsection 4.3.4 and for determining incorporation of the cluster by classifier. Incremental tracking variances have already been discussed in Subsection 1.2.2 (Lemma 2) Incorporation testing, which in other words means whether information from cluster does significantly worsen the prediction or not, is discussed in rest of the subsection.

Cluster incorporation - general

According to the general outline (Section 3.3), an important part of the overall approach is the classifier, which predicts labels of the incoming feature vectors. Now, we formalize the problem and discuss possible ways of determining whether cluster has already been incorporated by the classifier, which is in our scenario considered as the black box.

We say that cluster is *non-incorporated* by a classifier if it holds that if we include information from this cluster to prediction process then accuracy drops by a factor of predefined parameter. We present this term in more rigorous form in Definition 10.

Definition 10. *We follow notation of Definition 8. Consider feature subset $D \subseteq \{1, \dots, d\}$ and set of feature vector values $G \subset \mathbb{R}^{|D|}$ corresponding to features in D. Then we call set G (cluster) non-incorporated in time t_0 if there exists feature subset $S' \subseteq \{1, \dots, d\}$, $D \cap S' = \emptyset$ such that for given constant $\tau^{inc} \in (0, 1)$ it*

Algorithm 7 Update single cluster parameters

```
**Inputs**

  • $X_i^S$ - feature vector, (subset of original feature vector) $X_i^S \in \mathbb{R}^{|S|}$
  • $\{C_k\}_{k=1}^K$ - set of $K \in \mathbb{N}$ clusters each consisting of $(sp_k, \pi_k, \mu_k, inc_k, \Lambda_k, var_k)$
  • $k$ - index of updated cluster $k \in \mathbb{N}, 1 \leq k \leq K$
  • $n_{last}, c^{id}$ - incorporation parameters, $c^{id}$ is needed number of similar prediction
    in $n_{last}$ last samples from the cluster, $n_{last}, c^{id} \in \mathbb{N}$

//Increment k-th cluster by value $X_i^S$
$d_k \leftarrow (X_i^S - \mu_k)^T \Lambda_k (X_i^S - \mu_k)$    //Squared Mahalanobis distance
Calculate $Pr(X_i^S|z_k)$ as $\mathcal{N}(X_i^S; \mu_k, \Sigma_k) = \sqrt{\frac{1}{(2\pi)^{|S|}|\Sigma_k|}} e^{-\frac{1}{2}d_k}$
$r(z_k) \leftarrow \frac{\pi_k \mathcal{N}(X_i^S;\mu_k,\Sigma_k)}{\sum_{j=1}^K \pi_j \mathcal{N}(X_i^S;\mu_j,\Sigma_j)}$
$sp_k \leftarrow sp_k + r(z_k)$
Update parameters variances $var_k$ according to Subsection 1.2.2
$\Lambda_k \leftarrow \frac{1}{1-\Delta sp_k}\left(\Lambda_k - \frac{\Delta sp_k \Lambda_k (X-\mu_k)(X-\mu_k)^T \Lambda_k}{1+\Delta sp_k d_k}\right)$    //Update inverse covariance matrix
$|\Sigma|_k \leftarrow (1-\Delta sp_k)^d (1+\Delta sp_k d_k)|\Sigma|_k$    //Update determinant of covariance matrix
$\mu_k \leftarrow \mu_k + \frac{r(z_k)}{sp_k}(X_i^S - \mu_k)$
**if** $inc_k = False$ **then**
    //$X_i^S$ fits into non-incorporated cluster
    Test incorporation of $k$-th cluster by connected classifier according to Subsection
    4.3.3 which means:
    Let $cop_k$ be a set representing previous comparisons
    Compare prediction made by classifier on $X_i^{inc}$ and $X_i^{inc \cup S}$, where $inc$ is union
    of all feature subsets $S_j$ for which it holds that $X_i^{S_j}$ is not marked as outlier or
    non-incorporated.
    If $|cop_k| \geq c^{id}$ then drop the oldest value out of $cop_k$.
    Add 1 to $n_{last}$ if the two above mentioned predictions agree and add 0 otherwise.
    Count number of ones in $cop_k$ if it is at least $c^{id}$ then set $inc_k = True$.
**for** $k \leftarrow 1, \ldots, K$ {over all components} **do**
    $\pi_k \leftarrow \frac{sp_k}{\sum_{i=1}^K sp_i}$    //Update priors
```

holds that

$$\tau^{inc}\mathbb{E}_{(X,y)\sim P_{t_0+1,t_1}\cap G} l_A(M^G(R_{D\cup S'}(X)), y) \geq \mathbb{E}_{(X,y)\sim P_{t_0+1,t_1}\cap G} l_A(M^{inc}(R_{S'}(X), y)$$

, where M^G and M^{inc} optimizes loss l with respect to $P_{0,t_0}(R_{D\cup S',y}(X)$ and $P_{0,t_0}(R_{S'}(X), y)$ respectively. These distributions are corresponding restrictions of $P_{0,t_0}(X, y)$ created by integrating over dimensions of X which are not in $D \cup S'$ respectively in S'. Expectation is considered over distribution $P_{t_0+1,t_1}(X, y) \cap G$ which is created from $P_{t_0+1,t_1}(X, y)$ by not considering any X for which $R_{D'}(X) \notin G$.

According to Section 2.3 we decided to keep the problem in unsupervised setting since label delay can be extreme. Therefore, we build the approach on property given by Definition 8. Our scenario assumes relevant stable information given by set S, we can consider $S = S'$. Then we can set τ^{inc} in a way, that accuracy threshold for non-incorporated cluster would be connected with accuracy on stable relevant features. This is the key idea of our approach where we compare classifier predictions made using features from S with and without information from non-incorporated clusters. If the similarity exceeds a predefined threshold then we mark the cluster as incorporated.

For the following calculations, we typically set the threshold for similar predictions to 80%, though calculations can be performed for any probability value. This approach is largely motivated by Definition 10, which is based on the formalization in Section 2.2. However, it is important to note that the unsupervised procedure using the threshold for similar prediction probability provides an approximation rather than a strict determination of non-incorporation as per the definition.

Given the relatively small number of samples in this thesis, we avoid using strictly rigorous probabilistic estimations of identical prediction probabilities, as they are typically demanding and could negatively impact model performance. Instead, we provide a brief outline of possible more rigorous solutions but ultimately employ rather a heuristic approach, which we believe is still better than solving the process only by introducing a hyper-parameter.

One potential approach involves adaptive sampling, as detailed in Attachment A.5. Here, the positive observation would be the identical prediction, and the negative observation would be the differing prediction. Our goal is to approximate the probability of similar outputs so accurately that it exceeds a specified bound, with probability δ. However, as shown by Equations (A.1) and (A.2), achieving reasonable ϵ and δ values requires a sample size of at least several hundred. A more straightforward approach is presented in [Wonnacott and Wonnacott, 1990, Chapter 8] and we describe it in the following text.

Cluster incorporation - confidence interval solution

In [Wonnacott and Wonnacott, 1990, Chapter 8], there is presented an approach based on confidence intervals (see Attachment A.6). In what follows, we describe this approach, which we adopt for setting parameters of cluster incorporation (c^{id} and n_{last}) for evaluations discussed in Chapter 5.

The key difference from confidence estimations presented in Attachment A.6 lies in the standard deviation used. For estimating the mean, we used $\frac{\sigma}{\sqrt{n}}$, but now we are interested in the proportion. We approximate the probability of identical predictions as $\hat{P} = \frac{X^{id}}{n}$, assuming the number of identical predictions follows a normal distribution $X^{id} \sim \mathcal{N}(np, \sqrt{np(1-p)})$. From this we derive that $\frac{X^{id}}{n} = \hat{P} = \mathcal{N}(p, \frac{\sqrt{np(1-p)}}{n})$. We use a similar approach to that described in Attachment A.6. We are interested in lower bound, to ensure $\alpha\%$ confidence that our estimation of identical prediction p would not be greater by more that $\delta\%$. We rewrite error bound from lover bound of Equation (A.6) as $\delta = z_\alpha \sqrt{\frac{\hat{p}(1-\hat{p})}{n}}$, where z_α represents the *z-score* for quantile of α. For our scenario, when determining the required sample size n, we derive $n = \frac{z_\alpha^2 \hat{p}(1-\hat{p})}{\delta^2}$.

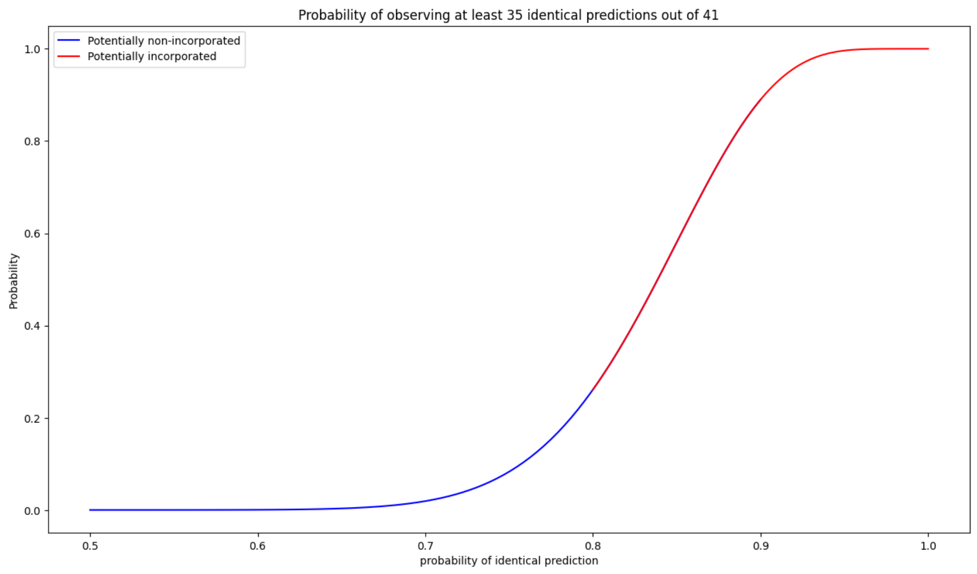


Figure 4.1: Graph of probabilities of observing at least 35 identical predictions out of 41 for distinct probability of matching. (Formula (4.2) for $n_{last} = 41$, $c^{id} = 35$ and changing p)

A challenge in our scenario is that the accuracy on the non-incorporated cluster may change over time as the classifier continues to learn. Consequently, the number of required samples might be greater than the number derived from the previous approaches. In our more practical solution, we address this by tracking $n_{last} = \frac{z_\alpha^2 \hat{p}(1-\hat{p})}{\delta^2}$ latest predictions made using the non-incorporated cluster.

Next, we need to determine the corresponding number of identical predictions to observe in these last n_{last} comparisons to conclude that the cluster has been incorporated. Assuming all latest samples follow the same distribution, the prediction comparison process follows a Bernoulli distribution with probability p. The corresponding cumulative distribution function is

$$Pr(\mathcal{X}^n \leq k) = \sum_{i=0}^{k} \binom{n}{i} p^i (1-p)^{n-i},$$

for $k \in \mathbb{N} \cup \{0\}, k \leq n$. This formula gives the probability of having at most k positive observations. Therefore, probability of observing at least c^{id} identical predictions out of n_{last}, assuming the process follows the identical prediction probability p, is equal to

$$1 - \sum_{i=0}^{c^{id}-1} \binom{n_{last}}{i} p^i (1-p)^{n_{last}-i}. \tag{4.2}$$

The values of this formula as function of p for $\delta = 0.1$, $\alpha = 0.9$ and consequently $n_{last} \approx \frac{1.28^2 0.5^2}{0.1^2} \approx 41$ and $c^{id} = 0.85 * n_{last} \approx 35$ can be seen in Figure 4.1. Note that for calculation of n_{last} we set $\hat{p} = 0.5$, since it maximizes formula $\hat{p}(1-\hat{p})$. We observe that probability for $p = 0.5$ is almost zero, for $p = 0.85$ it is more than 58% and for $p = 0.9$ it is already almost 89%. Note that we set the values of δ, α and c^{id} to be more relaxed than our scenario demands, this is due to small number of samples we test on. Therefore, we include parameters n_{last} and c^{id} as algorithm parameters, adjustable based on the conditions of the algorithm's use.

It is worth mentioning that due to the nature of Gaussian Mixture Models algorithm, a sample may be assigned to multiple clusters simultaneously. If some of these clusters are incorporated and others are not, we assume that the predictions are identical.

Considering the limitations of this approach for checking incorporation into the classifier, we propose future work based on Gaussian processes. This method could naturally connect the problem of incorporation with prediction uncertainty. This approach would involve studying the uncertainty of classifier predictions and determining whether feature subsets improve it.

4.3.4 Cluster creation

In this subsection, we examine the process of a new cluster creation as one of the essential parts of the algorithm scheme presented in Subsection 4.3.2. The motivation of the process is discussed in Subsection 1.8.2. We will first describe the procedure summarized in Algorithm 8, and then we will explore possible values for the unknown covariance matrix and the retention expiration time, both of which are parameters related to our approach.

Cluster creation scheme

In our algorithm, new clusters are created from samples detected as outliers, which are generally considered not to fit well into the current set of clusters (Subsection 1.7.1). These outliers are kept in a retention set, and each time a new sample is added to this set, we test whether a new cluster can be formed from the retention samples. If possible, these samples are removed from the retention set and used to create a new cluster. To determine the distance, we need a covariance matrix. We select this matrix as a weighted sum of the variances of all current clusters, where the weight corresponds to the prior of the cluster. This process is discussed in Subsection 4.3.4.

The problem of algorithm initialization, when no clusters yet exist, is solved by using a hyper-parameter covariance matrix and an initial learning period defined by the parameter t_b. During this period, all incorporation parameters are initialized to *True* (see Subsection 4.3.3). Since we operate in a stream scenario (Section 2.3), we must address the possibility of the retention set continually growing. This is managed by assigning a retention expiration time to each sample, which specifies the number of samples processed after which the sample is discarded from the retention set. This adaptive time threshold is discussed in Subsection 4.3.4. The entire procedure is detailed in Algorithm 8.

Retention set expiration parameter

In what follows, we present a possible solution for determining the parameter that dictates the duration for which a sample detected as an outlier is retained in the retention set. This retention period is crucial for assessing whether the sample represents the first example of a new concept or a rare property of the current distribution, a topic briefly discussed in Subsection 1.8.2. This parameter is important for the process described in Algorithm 8 as it optimizes memory usage and prevents the creation of noisy clusters.

We believe that the retention time should be connected to the model's complexity and the number of outliers. We achieve this by adjusting the retention expiration parameter accordingly. This is similar to the approach in [Diaz-Rozo et al., 2018, Section II C], which adapts the window width based on the assumption that the number of noise samples and concept drifts are related to the number of outlier detections. This idea, discussed in Section 1.7 and detailed in Attachment A.5, helps in detecting drifts and measuring their severity.

For a practical solution, we decided to use a more straightforward and heuristic approach based on simple probability. Given how samples from the retention set create a new cluster, it is natural to connect the retention expiration parameter with the following question:

> Assuming the examined sample belongs to a newly arrived valid cluster, after how many steps should another sample from the same cluster arrive?

We consider the minimal prior probability of the potential cluster, p_c, linked to the cluster deletion threshold discussed in Subsection 4.3.5. This means the minimal cluster prior probability p_{min} defines the threshold for discarding a cluster from the mixture. Since we count all samples, not just those incorporated into the mixture, we must also account for the probability of a sample being an outlier, $p_{outlier}$. Thus, we set the minimal probability of the potential cluster to $p_c = p_{min} * (1 - p_{outlier})$.

Suppose samples are drawn independently from the same distribution. The probability that no element of the potential cluster appears in n steps is $(1-p_c)^n$. To find the number of steps n such that the probability of at least one more example from the cluster arriving is $1-\delta$, we solve $1-(1-p_c)^n = 1-\delta$. From this, we derive $n = \frac{\log(\delta)}{\log(1-p_c)}$.

Since $p_{outlier}$ is unknown, we approximate it. For rigorous approximation can be used adaptive sampling as described in Attachment A.5. In our implementation, we approximate this value by the percentage of outliers in the last hundred samples, assuming this percentage can change with concept drift. Additionally, we note the potential use of the exponential distribution, which models the time until the next specific occurrence based on the historical average waiting time.

This solution also considers the desired model complexity, specifically how easily the concept can be described by the clusters, by setting p_{min}. The intuition

Algorithm 8 Cluster creation procedure

Inputs

- X_i^S - feature vector, (subset of original feature vector) $X_i^S \in \mathbb{R}^{|S|}$
- R^S - retention set
- $\{C_k\}_{k=1}^K$ - set of $K \in \mathbb{N}$ clusters each consisting of $(sp_k, \pi_k, \mu_k, inc_k, \Lambda_k, var_k)$
- $\Lambda_k^{initial}$ - Diagonal inverse covariance matrix used at the beginning when there is no cluster
- t_b - number of samples used for mixture of clusters initialization, $t_b \in \mathbb{N}$

```
if $K \geq 1$ then
    Compute $\Lambda^S_{universal} \leftarrow \left(\sum_{k=1}^K \pi_k D_k^{var}\right)^{-1}$   //Inverse of weighted mean of diagonal
                                                              matrices with variances on the diagonal
else
    $\Lambda^S_{universal} \leftarrow \Lambda_k^{initial}$
$K \leftarrow K + 1$   //Create cluster for new sample
$\mu_K \leftarrow X_i^S$
$sp_K \leftarrow 1$
$\Lambda_K \leftarrow \Lambda^S_{universal}$
$|\Sigma_K| = |\Lambda_K^{-1}|$   //Note that $\Lambda_K$ is diagonal
Initialize parameters for variances $var_K$ according to Subsection 1.2.2
if $i \leq t_b$ then
    $inc_K \leftarrow True$   //At the very beginning we have no incorporated
                               stable relevant information we can compare to
else
    $inc_K \leftarrow False$
for $k \leftarrow 1, \ldots, K$ {over all components} do
    $\pi_k \leftarrow \frac{sp_k}{\sum_{i=1}^K sp_i}$   //Update priors
for $r \leftarrow 1, \ldots, |R^S|$ {over all samples from retention set} do
    if $(X_i^S - R_r^S)^T \Lambda^S_{universal}(X_i^S - R_r^S) < \chi^2_{|S|,1-\beta}$ then
        Update $K$-th cluster by $R_r^S$ according to Algorithm 7
    else
        Check expiration time of $R_r^S$ (Subsection 4.3.4)
if no sample fitted into $K$-th cluster then
    $K \leftarrow K - 1$   //Delete cluster, no match with retention set
    for $k \leftarrow 1, \ldots, K$ {over all components} do
        $\pi_k \leftarrow \frac{sp_k}{\sum_{i=1}^K sp_i}$   //Update priors
    Include $X_i^S$ into retention set $R^S$ and determine the sample retention set expiration
    time as described in Subsection 4.3.4.
```

is that if there are many clusters, rarer properties need more time to reoccur and thus be included in the cluster description.

Parameters initialization and universal covariance matrix

In Subsection 1.8.2, we discussed the general strategy for creating new clusters. Here, we focus on the initial choice of parameters, particularly the covariance matrix, which is essential for comparing samples in the retention set and establishing new clusters. Since we use the same matrix for both comparison and initialization, representing general information about the entire mixture, we refer to it as the *universal covariance matrix*. This matrix plays an important role in the algorithm presented in this section, especially in the process discussed in Subsection 4.3.4.

Initializing cluster parameters is a critical step in new cluster creation. While selecting the mean vector is straightforward, choosing the best value for *sp* is less clear. We follow the good practice of initializing the *sp* parameter to 1, which makes sense considering it represents the fit of the very first sample that creates the cluster.

Now, we will concentrate on covariance matrix. The usual approach when creating cluster for single samples is described in [Pinto and Engel, 2015, Section 2.2]. There they chose to set covariance matrix Σ to diagonal matrix consisting of the overall dataset squared standard deviation (variance) multiplied by the constant δ. Mostly similar approach is chosen in [Sun et al., 2023, Subsection 2.2] where Σ for feature vector $X \in \mathbb{R}^d$ is set ass follows

$$\Sigma = \delta^2 \begin{pmatrix} (X^1)^2 & 0 & \cdots & 0 & 0 \\ 0 & (X^2)^2 & \ddots & \vdots & 0 \\ \vdots & \ddots & \ddots & 0 & \vdots \\ 0 & \cdots & 0 & (X^{d-1})^2 & 0 \\ 0 & 0 & \cdots & 0 & (X^d)^2 \end{pmatrix}.$$

The main advantage of these approaches is the ease of computing the inverse covariance matrix and the determinant due to the diagonal nature of the matrices. Both of these values are necessary for the fast incremental procedure presented in Subsection 1.6.2. The following properties of a diagonal matrix illustrate this advantage. Let:

$$A = \begin{pmatrix} a_1 & 0 & \cdots & 0 & 0 \\ 0 & a_2 & \ddots & \vdots & 0 \\ \vdots & \ddots & \ddots & 0 & \vdots \\ 0 & \cdots & 0 & a_{d-1} & 0 \\ 0 & 0 & \cdots & 0 & a_d \end{pmatrix},$$

then $|A| = \prod_{i=1}^{d} a_i$ and

$$A^{-1} = \begin{pmatrix} a_1^{-1} & 0 & \cdots & 0 & 0 \\ 0 & a_2^{-1} & \ddots & \vdots & 0 \\ \vdots & \ddots & \ddots & 0 & \vdots \\ 0 & \cdots & 0 & a_{d-1}^{-1} & 0 \\ 0 & 0 & \cdots & 0 & a_d^{-1} \end{pmatrix}.$$

The determinant's value can be easily seen from its definition, as all products except the diagonal consist of zero multipliers. The second equation results from the multiplication of these matrices and the property $A^{-1}A = I_d$.

Without the need for fast inverse and determinant computation, we can find an interesting solution in Aletti and Micheletti [2017], where the authors approximate the universal covariance matrix by the normalized weighted sum of the current covariance matrices. Although this approach involves many costly inversions, we can modify it to avoid this by approximating the covariance matrices with their diagonal values, computing variance for each cluster. Unfortunately, we are not aware of a fast way to compute variances from the inverse covariance matrix. Therefore, we track the variances similarly to the Gaussian Naive Bayes method presented in Subsection 1.2.2, using cluster priors as weights. This approach effectively takes the weighted mean of the most important directions, albeit restricted by the diagonal approximation. We also note that there is no known efficient incremental algorithm for tracking weighted variance. Thus, while computing variances, we do not distinguish how well a sample fits into the cluster.

A disadvantage of our approach is that it assumes the existence of some clusters to establish the universal covariance matrix. This can be addressed by either performing a non-incremental clustering algorithm on the initial set of samples to establish the initial mixture of clusters or by initializing the first covariance matrix with given parameters. We adopt the latter method in our algorithm, as initializing the mixture of clusters is not the primary objective of this thesis.

4.3.5 Deletion of cluster

In this subsection, we examine the process of maintaining the relevance and accuracy of the cluster set by removing obsolete or redundant clusters based on prior probability thresholds and significant overlap. This cluster removal procedure is a crucial aspect of Algorithm 5, ensuring the distribution remains relevant and free from noise or outdated clusters. First, we discuss deletion period (used in Algorithm 5). Then we describe the procedure detailed in Algorithm 9. Finally, we introduce a novel method for determining the prior probability threshold used as a criterion for cluster deletion.

Deletion period

Based on specific scenario of drifting environment we connect time of cluster purification and *sp* parameter forgetting with mixture complexity. This give us a bounds on single arriving sample impact and we do not risk, that in longer period of time sum of *sp* parameter will increase beyond all limits or gradually decrease. This approach uses two parameters SP^L and SP^U. SP^L is used in Algorithm 9 for forgetting procedure. It stands for new sum of *sp* parameter we want to reach after purification. It also determines impact of single arriving sample after forgetting. On the contrary, SP^U determines upper bound on the sum of *sp*, which we can understand as bound on minimal possible impact of the newly arriving sample. Based on this intuition we believe that reasonable setting of these parameters should come from assumption on the complexity of the concepts and speed of drift. Idea of this approach is very similar to one presented in Equation (1.31).

Deletion procedure

In this part, we discuss Algorithm 9, which is responsible for updating the mixture by removing noisy and obsolete clusters.

Algorithm 9 is largely based on [Sun et al., 2023, Algorithm 3]. We adopt the concept of a logical matrix LM for determining cluster overlap and the elimination of clusters based on their priors, a common step in similar algorithms (both methods are discussed in Subsection 1.8.1). However, later in this subsection, we propose a novel method for setting the threshold for this operation.

We have also incorporated cluster probability testing using a defined prior probability threshold during the overlap examination. This adaptation is necessary due to the adaptive nature of the threshold and to prevent the deletion of relevant clusters because of non-mutual overlap with less significant clusters. Algorithm 9 follows the approach in [Sun et al., 2023, Algorithm 3] by deleting clusters with excessive overlap. Although merging these clusters, as suggested in Subsection 1.8.1, seems advantageous, it is incompatible with the fast incremental GMM because it would require undesirable approximations or matrix inversion and determinant computation.

Our algorithm also incorporates a forgetting factor, used in Equations (1.30) and (1.31), but modified for our scenario. Assuming a highly drifting environment, we generally set the boundary to the sum of sp parameters, representing the weights of the current clusters in GMM. Each arriving sample adds its weight to each cluster depending on how well it fits into them. If the sum of all sp is great, then the newly arriving sample has only a minimal influence on the overall mixture. Therefore, we factor all sp parameters by value $min(1, \frac{SP^L}{\sum_{i=1}^{K} sp_i})$. If $\sum_{i=1}^{K} sp_i$ was greater than SP^L then we reduce to SP^L, which ensures that the new sample is not seriously overlooked. If the sum is already smaller that the bound, we do not make any changes.

Cluster prior deletion threshold

In the following text, we investigate the deletion threshold for cluster prior probability used in the subprocesses of Algorithm 5. This threshold is crucial for maintaining the relevance of the distribution by removing obsolete and noisy clusters.

In Subsection 1.8.1, we briefly discussed two approaches for setting the threshold for the cluster prior in the Gaussian Mixture Model algorithm. The first approach involves using a basic hyper-parameter, and the second approach, introduced by Sun et al. [2023], employs an adaptive threshold based on extensive search. Here, we propose an alternative solution grounded in the properties of probabilistic distributions. Specifically, we suggest using the Dirichlet distribution, as described in Section 1.4, to set the threshold for the cluster parameter π. The main advantage of our method lies in its ability to accommodate various assumptions about the probability distribution of the expected mixture of clusters.

To construct the threshold for probabilities, we assume that the distribution of cluster probabilities follows a Dirichlet distribution with specific parameters α. The threshold is then defined as the maximum value such that a sample drawn from this Dirichlet distribution with the chosen parameters will, with probability

Algorithm 9 Check for spurious clusters

```
**Inputs**
  • $\{C_k\}_{k=1}^{K}$ - set of $K \in \mathbb{N}$ clusters each consisting of $(sp_k, \pi_k, \mu_k, inc_k, \Lambda_k, var_k)$
  • $\pi_{min}^{K}$ - adaptive threshold for cluster prior probability, discussed in Subsection 4.3.5
  • $SP^L$ - bound on sum of $sp$ for forgetting procedure, $SP^L \in \mathbb{R}^+$
//Remove spurious clusters by testing threshold on their priors and overlay
While there are $\pi_k$ such that $\pi_k < \pi_{min}^{K}$ delete $k$-th cluster set $K \leftarrow K-1$ and update priors
Compute logical matrix $LM$ of shape $K \times K$, where $LM_{i,j}$ is according to Equation (1.32)
while $\exists j \; \sum_{i=1}^{K} LM_{i,j} > 1$ do
    Delete $k$-th cluster such that
        $$\sum_{k=1}^{K} LM_{k,j} > 1 \textit{ and } sp_k = min(\{sp_i \; ; \; \sum_{i=1}^{K} LM_{i,j} > 1\})$$
    Set $K \leftarrow K-1$, delete $k$-th row and column of $LM$
while $\exists j \; \sum_{i=1}^{K} LM_{i,j} = 1$ do
    Delete $k$-th cluster such that
    Update all clusters priors and delete ones with prior probability below $\pi_{min}^{K}$, delete also corresponding row and column of $LM$
        $$\sum_{k=1}^{K} LM_{k,j} = 1 \textit{ and } sp_k = min(\{sp_i \; ; \; \sum_{i=1}^{K} LM_{i,j} = 1\})$$
    $\sum_{k=1}^{K} LM_{k,j} = 1$ and $sp_k = min(\{sp_i \; ; \; \sum_{i=1}^{K} LM_{i,j} = 1\})$
    Set $K \leftarrow K-1$, delete $k$-th row and column of $LM$
    Update all clusters priors and delete ones with prior probability below $\pi_{min}^{K}$, delete also corresponding row and column of $LM$
$\beta^{forget} \leftarrow min(1, \frac{SP^L}{\sum_{i=1}^{K} sp_i})$
for $k \leftarrow 1, \ldots, K$ {over all components} do
    $sp_k \leftarrow \beta^{forget} sp_k$
Update priors according to current values of $sp$
```

δ, have no component smaller than this threshold. This means that with probability δ, no cluster will have a probability less than the observed threshold. By selecting specific α values, we can derive various thresholds based on different assumptions about the cluster probabilities. Figure 1.5 provides an intuition of how different α values affect the Dirichlet distribution. In Attachment A.8 we describe exponential threshold of Sun et al. [2023] in detail.

Advantage of our approach lies in its interpretability and the flexibility to adjust boundaries based on δ and the expected distribution of cluster probabilities. However, the Dirichlet distribution's adaptive computation is computationally intensive, necessitating assumptions about the cluster probability distribution, which are made by selecting parameters α. The thresholds in this thesis are computed by generating 10^6 samples from the Dirichlet distribution with specific α parameters, sorting these samples, and selecting the value corresponding to probability δ.

A significant difference in our proposed prior threshold function emerges when considering a larger number of clusters. The exponential threshold from Sun et al. [2023] was derived from experiments with a limited number of clusters, which might not perform well in scenarios with complex distributions, as its exponential nature results in very small thresholds (e.g., less than 10^{-7} for $K = 50$). In contrast, our distribution-based approach does not decrease as rapidly. To simplify computation in experiments, we approximate our threshold with a basic hyperbolic function $\frac{a}{b+K}$ for given parameters a and b. An example of this approximation is shown in Figure 4.2.

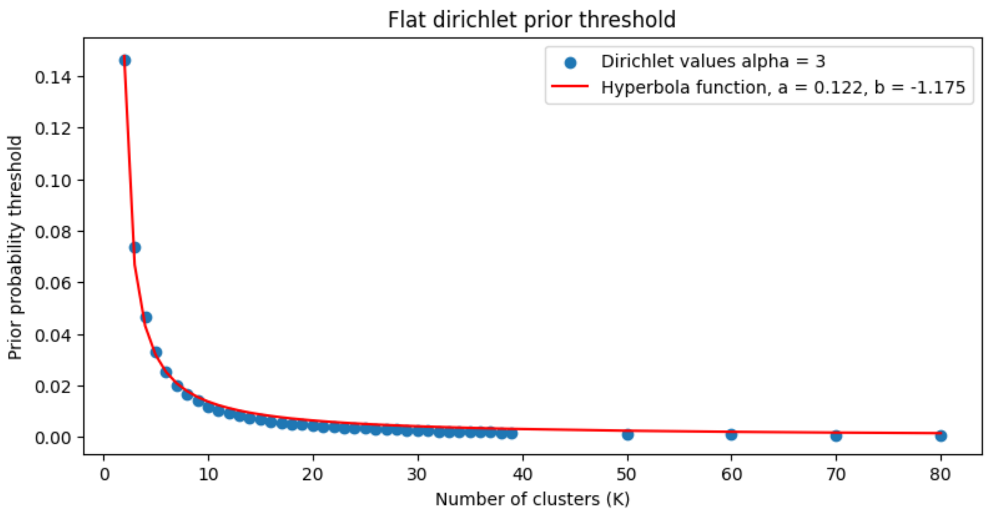


Figure 4.2: Approximation of flat Dirichlet threshold for $\alpha = 3$ by basic hyperbola $\frac{a}{b+x}$.

5. Implementation and experiments

In this chapter we present evaluations of procedures discussed in the thesis.

First in Section 5.1 we describe datasets, which we use in this chapter. Then we present 5 evaluations, listed below.

- In Section 5.2, we replicate some result of Kolter and Maloof [2007] using our modified DWM algorithm (Algorithm 3).
- In Section 5.3, we examine performance of DWM algorithm (Section 1.2.1) in our scenario (Chapter 2) and we will highlight its limitations which served as motivation of our solutions in Chapter 3.
- In Section 5.4 we evaluate Hellinger procedure (Section 4.2) showing that it is able to perform well considering severe concept drift.
- In Section 5.5 we present evaluation of overall solutions in comparison with DWM algorithm using our synthetic dataset (Subsection 5.1.2).
- In Section 5.6 we evaluate our overall solutions in comparison with DWM algorithm using real life data (Subsection 5.1.3).

Each evaluation presented in this chapter is divided into three subsections. The first subsection states the aim of the evaluation and the criteria for its fulfilment. The second subsection describes the execution of the evaluation, detailing the exact parameter values and the reasoning behind their selection. The final subsection presents the results of the evaluation, discussing whether they meet the hypothesis criteria and the implications of these findings.

In Attachment A.14, we briefly discuss structure and used software of our supplementary code utilizing evaluations of this chapter.

5.1 Datasets

In this section, we outline the datasets utilized for evaluations in this chapter.

In Subsection 5.1.1, we present the "SEA" dataset as described in Kolter and Maloof [2007]. This dataset is employed to demonstrate certain properties of Dynamic Window Mechanism (DWM) and to replicate results from Kolter and Maloof [2007] in Section 5.3.

Next, in Subsection 5.1.2, we introduce a synthetic dataset we developed specifically to closely capture the properties of the scenario presented in Chapter 2. This dataset serves as the foundation for evaluations in Section 5.3, Section 5.4, and Section 5.5.

Finally, in Subsection 5.1.3, we describe the real-world DREBIN dataset (Arp et al. [2014]), which is used for comparison purposes in Subsection 5.6.

12 concepts dataset description:

$X = (X^1, X^2, X^3, X^4, X^5, X^6)$ $\mathcal{Y} = \{1, 2\}$ X^6 is irrelevant, randomly sampled from interval $[0.0, 10.0]$.

For label 1 : $X^2 + X^3 + X^4 + X^5 \leq b$ $X^1 = c$	For label 2 : $X^2 + X^3 + X^4 + X^5 > b$ $X^1 = d$

Concept :	1	2	3	4	5	6	7	8	9	10	11	12
b :	20.0	18.0	17.0	18.0	17.6	20.6	20.2	19.6	17.4	18.0	18.4	19.0
c :	2	1	3	6	11	5	4	9	4	10	2	5
d :	1	3	4	7	2	4	7	8	3	6	0	4

Figure 5.1: Description of the 12 sudden concept drifts contained in the dataset. Each of these concepts are represented by 2000 samples.

5.1.1 "SEA" dataset

"SEA concepts" dataset described in Kolter and Maloof [2007] consists of feature vectors $X = \{X^1, X^2, X^3\}$, where for all $i \in \{1, 2, 3\}$ it holds that $X^i \in \mathbb{R}$ and $0.0 \leq X^i \leq 10.0$. The decision boundary is set to $X^1 + X^2 \leq b$ for given drifting b. X^3 is an irrelevant feature (independent of label) uniformly randomly sampled from interval $[0.0, 10.0]$.

5.1.2 12 Concepts dataset

We design this dataset to capture the key aspect of our scenario presented in Chapter 2, especially with focus on terms severe concept drift and relevant stable information defined in Definition 8.

Our dataset consists of 12 sudden concepts 2000 samples each. The feature vector X consists of 6 features $X^1, \ldots, X^6$ for which it holds that for all $i \in \{1, 2, 3, 4, 5, 6\}$ it satisfies that $X^i \in \mathbb{R}$, $0.0 \leq X^i \leq 10.0$. X^6 is similarly as in "SEA concepts" irrelevant and generated randomly. The rest of features are connected with exact concept drift specifics. $X^2, \ldots, X^5$ follow rule $X^2 + X^3 + X^4 + X^5 \leq b$ for drifting b and the most important feature X^1, representing severe drifting feature, follows $X^1 = c$ for drifting c. Note that features $X^2 + X^3 + X^4 + X^5$ also drift in distribution since our dataset is label balanced. According Definition 8, $X^2, \ldots, X^5$ provide stable information and X^1 ensures severe concept drift.

The features are generated randomly but they follow primarily known boundary they have to satisfy. To be more specific in implementation for uniformly randomly generated label we generate feature vector uniformly at random until they meet the according concepts specifics. The dataset description can be seen in Figure 5.1.

5.1.3 DREBIN dataset

This dataset is taken from Ceschin et al. [2022] and is based on well known DREBIN dataset Arp et al. [2014], which consists of real life samples used for malware detection. What is the most important for us is that this dataset was collected during period of time which, as shown in [Ceschin et al., 2022, Fig. 3 (a)], caused that there are concept drifts in the data. Note that it is not clear if these drifts are severe and therefore if they meet assumptions from Section 2.3.

The dataset was primarily collected in 2011 and 2012. While we acknowledge that this data may be outdated and not indicative of current malware performance, it remains suitable for comparing different algorithmic approaches, which is the primary objective of this thesis.

Dataset is ordered by submission date and consists of 13 columns, 3 are not part of the feature vector these are label, submission date and sha256. Feature vector consists of columns api call, permission, url, provider, feature, intent, activity, call, service render and real permission. All these are textual columns. At all there are 123453 benign and 5560 malignant samples, whose distribution in time can be seen in [Ceschin et al., 2022, Fig. 2 (a)].

5.2 Replication of DWM results

In this section we will demonstrate brief evaluation on "SEA" dataset (Subsection 5.1.1) where we replicate some results of Kolter and Maloof [2007]. This replicated results will also demonstrate important property of DWM algorithm, which will be further examined in Section 5.3.

5.2.1 Evaluation hypothesis

In this subsection, we state the main hypothesis of this evaluation, which is focused on validation of our modifications of DWM algorithm (Algorithm 3).

- Our modified DWM algorithm (Algorithm 3) generalizes original DWM algorithm (Algorithm 1).

We will examine whether is our modified DWM algorithm, given specific parameters (label delay $l = 0$, learning period $t = 1$), equivalent to original DWM algorithm.

We replicate results presented in [Kolter and Maloof, 2007, Subsection 4.2], specifically [Kolter and Maloof, 2007, Fig. 7]. We will state that hypothesis is met if our results would follow depicted confidence intervals.

5.2.2 Evaluation setting

We are replicating results presented in [Kolter and Maloof, 2007, Fig. 7]. This experiment uses "SEA" dataset (Subsection 5.1.1) and Gaussian Naive Bayes classifier (Subsection 1.2.2) as expert.

Their setting, which we adopt, is as follows. They created 50000 samples, divided into four concepts 12500 samples each. The concepts are sudden (see Subsection 2.2.2) and are given by b for which it holds that the first concept satisfy $b = 8$, the second $b = 9$, the third $b = 7$ and the fourth $b = 9.5$. The experiment parameters consists of the decreasing weight factor $\beta = 0.5$, updating period $p = 50$ and the removing threshold $\theta = 0.01$.

They track number of experts and percentage accuracy. For the accuracy, they generate 2500 random samples of the current concept, which should be estimated at each time step. Results and confidence intervals are reached by averaging accuracies of 10 runs of the algorithm. Note that considering accuracy

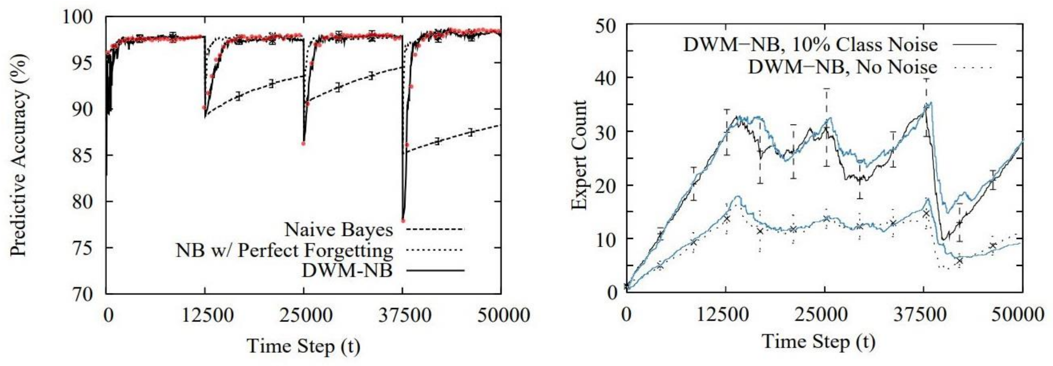


Figure 5.2: Comparison of our results and original results. The blue and light blue are ours corresponding to black and light back original lines. The red dots are ours and corresponds to black line.

experiment, due to insufficient computational power we evaluate accuracy only every 500 time stamp.

To mimic the original experiment we further set label delay $l = 0$ and learning period $t = 1$.

5.2.3 Evaluation results and discussion

Results of the evaluation are presented in Figure 5.2, where we see [Kolter and Maloof, 2007, Fig. 7] together with our results. We can observe that our results meet original confidence intervals. The left part of the Figure 5.2 is accuracy comprising where our results are the red dots trying to meet the black line "DWM-NB", due to insufficient computing power we tested every 500 time stamp. The right graph represents the number of experts in the run of the algorithm. The blue and the light blue line represents our result following the black and the light black line.

We note that in the Figure 5.2 we can see great accuracy drops in the periods of concept drift. This is key phenomenon in context of this thesis.

5.3 Evaluation of DWM algorithm

In this section, we present the results of the DWM algorithm (see Subsection 1.2.1), highlighting its shortcomings and potential improvements, particularly in the context of datasets experiencing severe concept drift (Definition 8).

The information discussed here, particularly the results of DWM on our synthetic dataset (Subsection 5.1.2), is crucial in Section 3.3, where we presented a general outline of our solutions.

5.3.1 Evaluation hypothesis

In this subsection we state the main hypothesis following our focus on DWM algorithm accuracy drops connected with scenario experiencing severe concept drift.

Since we are interested in performance of DWM algorithm in our scenario, which includes severe concept drifts and relevant stable information (see Chapter 2) we state the main evaluation hypothesis as follows.

- DWM algorithm (Subsection 1.2.1) suffers on dataset including severe concept drift and relevant stable information (Definition 8) from unnecessary drop of prediction accuracy.

We will examine the hypothesis by demonstrating that the DWM algorithm on the 12 Concepts dataset (Subsection 5.1.2) overly relies on highly decisive yet severely drifting feature (X^1), which degrades prediction accuracy during periods of concept drift. We consider the hypothesis to be confirmed if there are clear periods where the mean accuracy, considering the severely drifting feature, is significantly lower than the accuracy when this feature is excluded.

5.3.2 Evaluation setting

In this evaluation, we run our modified DWM algorithm (Algorithm 3) on 12 Concepts dataset (Subsection 5.1.2) and measure accuracy in last 100 time steps. We compute mean accuracies of 10 runs for scenarios with and without severely drifting feature X^1.

When setting the parameters, we very much adopt values from Subsection 5.2.2. We sett decreasing weight factor $\beta = 0.5$, updating period $p = 50$, the removing threshold $\theta = 0.01$, label delay $l = 0$ and learning period $t = 1$.

Note that in Section 5.2, we showed that modified DWM algorithm (Algorithm 3) with these parameters shows the same results as original DWM algorithm (Algorithm 1).

5.3.3 Evaluation results and discussion

The results of the evaluation are presented in Figure 5.3. There we compare the accuracy of the DWM algorithm in two scenarios. The red line represents our dataset described in Subsection 5.1.2, while the green line represents the same dataset with the severely drifting feature X^1 removed. The results show the mean accuracies on the last 100 samples over 10 runs of the algorithm.

We consider drops in the short periods of drift change as significant and therefore we state the hypothesis as fulfilled. This evaluation motivates the general solution presented in Chapter 3.

5.4 Evaluation of Hellinger subcomponent

In this section, we give some insight into Hellinger concept drift detection presented in Section 4.2. We will examine the procedure by evaluating whether it is able to efficiently detect severe concept drift on the 12 Concepts dataset. Additionally, we observe how many samples from the new concept process actually needs for the detection.

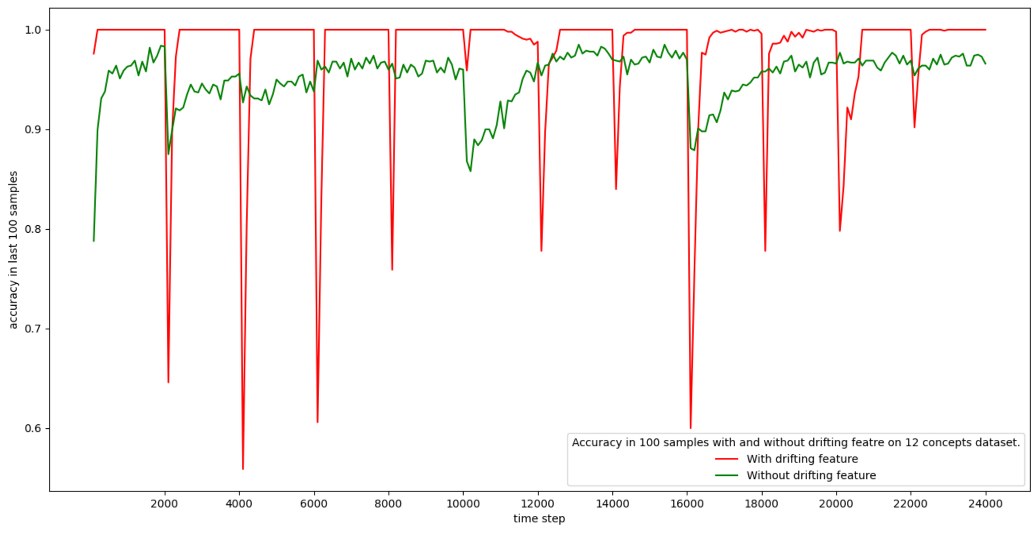


Figure 5.3: Accuracies in last 100 samples on dataset described in Figure 5.1. Comparison of accuracies on dataset including highly relevant but drifting feature X^1 (green) and excluding this feature (red).

5.4.1 Evaluation hypothesis

Evaluation in this section aims to show ability of Algorithm 4 on data containing severe concept drift (Definition 8).

The main evaluation hypothesis is that

- Proposed procedure based on Hellinger concept drift detection (Algorithm 4) is able to efficiently detect severe concept drift.

We will examine this hypothesis by checking right and wrong concept drift detections on the 12 Concepts dataset (Subsection 5.1.2). For this, we need to define what we consider to be the *right detection* and the *wrong detection*.

> Right detection occurs when a new concept in the highly drifting feature subset (X^1) is detected within one window size of its actual arrival.
>
> False detection is any detection that is not considered the right.

To state that the algorithm can efficiently detect the severe drift we demand that on a dataset containing the severe concept drift, there exists a setting for which the 95% confidence interval of right detections contains the real number of the drifts and at the same time the 95% confidence interval of false detections contains zero. We note that considering the 12 Concepts dataset, according to our definition we can not detect concept drift correctly on any feature subset, which does not contain feature X^1. There is also drift in other features but we do not consider it to be severe.

5.4.2 Evaluation setting

The evaluation presented in this section focuses on assessing the performance of the Hellinger procedure (Algorithm 4) using the 12 Concepts dataset (Figure 5.1), which inherently includes severe concept drift due to its construction. During evaluation, we will track several metrics: correct detections, false detections, and the number of samples required for correct detection. The latter metric represents the number of samples between the arrival of concept drift and the accurate detection of the drift, constrained by the window size. We will compute the mean across 10 algorithm runs and calculate 95

Each of these metrics will be recorded for each individual feature subset. Although the last metric (number of samples for right detection) is not essential for evaluating the hypothesis, we include it because it is relevant to the overall process discussed in Section 3.3.

We will run two experiments, one with 3 feature subsets according to dependencies and one where all features create single overall feature subset. Parameters are set as follows, window size $w = 150$, number of bins is set accordingly to square root of w (see Subsection 1.3.2), parameter of standard deviation γ is set to 4 (see Section 4.2.1). To not unnecessarily complicate a histogram creation, we suppose that we know the maximal and minimal values which features can acquire. Hence we set $B_{max} = 11$ and $B_{min} = 0$ for all features (see Subsection 1.3.2). Number of minimal collected epsilons before concept drift detection is set to 30.

5.4.3 Evaluation results and discussion

In this subsection we present results of the evaluation of Algorithm 4 on 12 Concepts dataset (Figure 5.1) according to the setting presented in previous subsection.

Table 5.1: Algorithm 4 on 12 Concepts dataset, mean of 10 with 95% confidence interval.

	Right detections	False detections	Needed samples
$(X^1, X^2, X^3, X^4, X^5, X^6)$	$\mathbf{11 \pm 0}$	$\mathbf{0.1 \pm 0.2}$	38.6 ± 4.4
(X^1)	$\mathbf{10.75 \pm 0.35}$	3.2 ± 0.9	5.2 ± 2.9
(X^2)	-	0.4 ± 0.4	-
(X^3, X^4, X^5, X^6)	-	0.1 ± 0.2	-

The results are presented in Table 5.1. There we can see that optimal number of correct detections (11) is consisted in both confidence intervals, for single overall feature subset it was detected correctly every time. Number of false detections differ. Confidence interval for overall feature subset contains 0 and therefore fulfils our evaluation hypothesis, if we consider single feature subset then number of false detections is greater. On the other hand, we can see that as divided feature is more sensitive to false detections it is able to detect concept drift using less samples. For better intuition of the algorithm process, we present graphs of epsilon evolution during the single run in Attachment A.10.

5.5 Evaluation of overall process on 12 Concepts dataset

In this section, we present evaluation of overall procedure illustrated in Section 3.3 on our synthetic dataset described in Subsection 5.1.2. We will show comparison of DWM algorithm with and without our proposed sub-procedures aiming indication of the feature subsets drift (Section 4.2 and Section 4.3).

Dataset is evaluated on three algorithms.

- DWM algorithm without drift indicator - Algorithm 3
- DWM with Hellinger drift indicator - Algorithm 3 with Algorithm 4
- DWM with GMM based drift indicator - Algorithm 3 with Algorithm 5

5.5.1 Evaluation hypothesis

Evaluation of this section aims to show ability of the sub-procedures (Algorithm 4 and Algorithm 5) according to our scenario (Section 2.3) to improve the results of the general robust procedure that we chose to be DWM algorithm (Algorithm 3).

Main hypotheses of this evaluation are

- Proposed sub-procedure based on Hellinger concept drift detection (Algorithm 4) is able to improve accuracy of DWM algorithm (Algorithm 3) in scenario containing severe drift, relative stable information and label delay (Section 2.3).
- Proposed GMM based sub-procedure (Algorithm 5) is able to improve accuracy of DWM algorithm (Algorithm 3) in scenario containing severe drift, relative stable information and label delay (Section 2.3).

We will state that certain sub-procedure improves the algorithm if it holds that 95% confidence interval of the mean accuracy is without overlay greater than when the sub-procedure is not used. Since the examined dataset has generally the same number of labels we choose to measure and compare the percentage accuracy of the algorithms.

5.5.2 Evaluation setting

In evaluation presented in this section, we compare results of three algorithms on 12 Concepts dataset (Subsection 5.1.2). General approach of evaluation is that we will compute 95% confidence intervals for mean percentage accuracy on 10 runs for each of the procedures. This will be done for two scenarios (Section 2.3), with very small label delay $l = 50$ and regular delay $l = 1000$. We note that the data drifting assumption is satisfied by the 12 Concepts dataset from its construction.

Since we want to show that our sub-procedures have ability to improve the result of DWM algorithm we set parameters of this algorithm identically for each process. We sett them very similarly as were set in Section 5.3. This means updating period $p = 50$, removal threshold $\Theta = 0.01$, decreasing factor $\beta = 0.5$ and we add learning parameter $t = 2$.

Algorithm 4 considers 3 feature subsets according dependencies. Tt is set similarly as in Subsection 5.4.2, which means parameter of standard deviation $\gamma = 4$, window size $w = 150$, $B_{max} = 11$ and $B_{min} = 0$ for all features. Number of minimal collected epsilons before concept drift detection is set to 30. After drift is detected the classifier do not use the drifted feature subset for 75 time steps for label delay $l = 50$, respectively for 400 time steps for $l = 1000$.

Algorithm 5 considers 3 feature subsets based on dependencies and is set according discussion in Section 4.3. Initialization period $t_b = 300$, outlier percentile $\beta = 0.05$, upper and lower bound for sum of *sp* $SP^U = 600$, $SP^L = 300$, incorporation parameters $n_{last} = 41$, $c^{id} = 35$ (Subsection 4.3.3), unit initial inverse covariance matrices and prior threshold given by 3 flat Dirichlet distribution (Subsection 4.3.5). This setting is similar for both label delays.

5.5.3 Evaluation results and discussion

In this, subsection we present accuracy of DWM algorithm with and without presented sub-procedures on 12 Concepts dataset. We do it for two label delays $l = 50$ and $l = 1000$. In Table 5.2 we can see results of the accuracy comparison.

Table 5.2: Accuracy comparison of DWM algorithm with and without sub-procedures on 12 Concepts dataset for label delay $l = 50$ and $l = 1000$. Mean of 10 with 95% confidence interval.

	Accuracy $l = 50$	Accuracy $l = 1000$
DWM + GMM	$\mathbf{0.964 \pm 0.003}$	$\mathbf{0.809 \pm 0.016}$
DWM + Hellinger	0.954 ± 0.005	$\mathbf{0.675 \pm 0.004}$
DWM	0.947 ± 0.006	0.627 ± 0.003

For delay $l = 1000$ both sub-procedures significantly improved original algorithm accuracy and fulfilled evaluation hypothesis. Using delay $l = 50$ only approach using GMM based sub-procedure met criterion on confidence intervals but we note that for each of 10 examined runs of the algorithms using of Hellinger sub-procedure we reached higher accuracy than regular DWM algorithm. For better insight into algorithms performance we present Figure 5.4 and Figure 5.5, which show evolution of mean accuracy during the evaluation (every time in last 100 samples). We highlight that performance especially of GMM based sub-procedure is very similar to one which we were aiming for in Chapter 3, which is motivated by evaluation in Section 5.3 (especially Figure 5.3). Note that this happens with three exceptions where the drifts primarily change labels of feature vector values which remains same (switches their labels), which is challenging setting for unsupervised techniques.

5.6 Evaluation on DREBIN dataset

We will evaluate 3 procedures, similar ones as in Section 5.5, on real world dataset called DREBIN dataset, which consists of android malware samples and spans

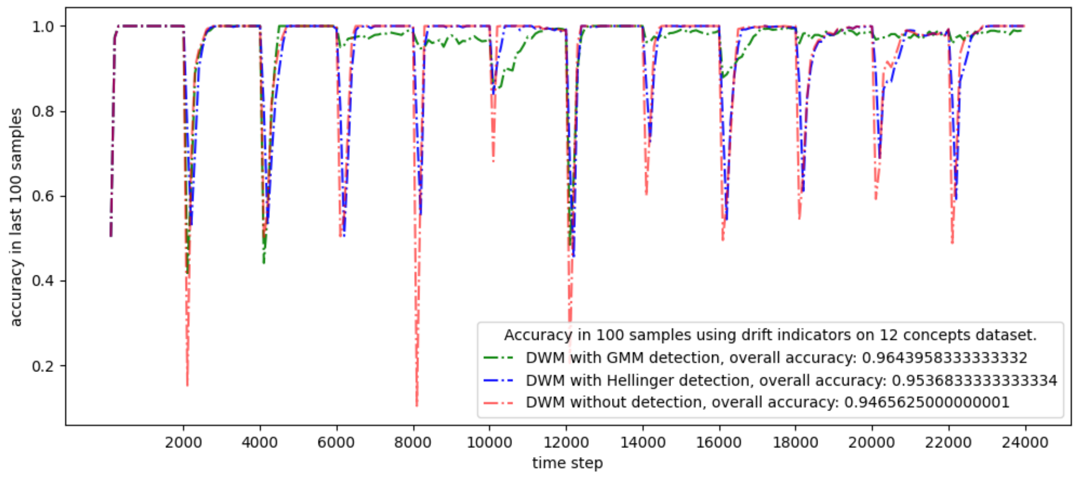


Figure 5.4: Comparison of 3 procedures (Section 5.5) on 12 Concepts dataset with label delay $l = 50$. Mean accuracy from 10 runs in last 100 samples.

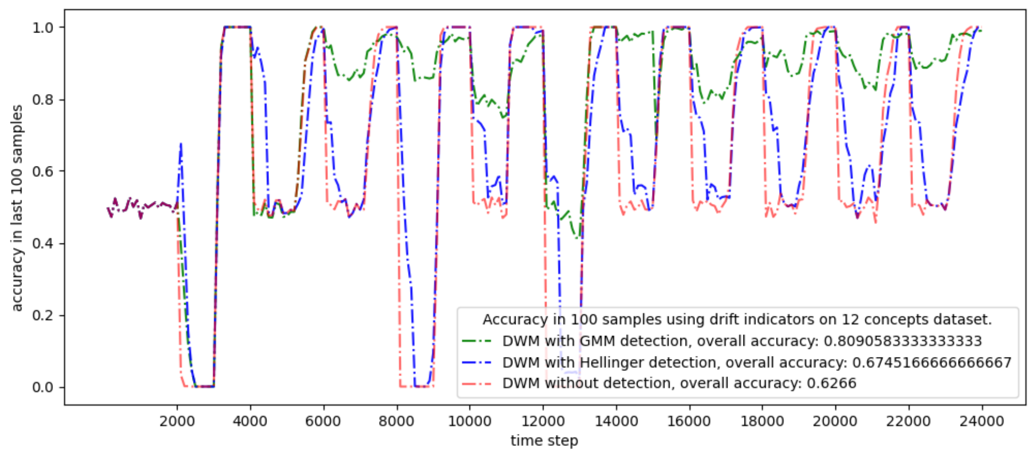


Figure 5.5: Comparison of 3 procedures (Section 5.5) on 12 Concepts dataset with label delay $l = 1000$. Mean accuracy from 10 runs in last 100 samples.

over two years.

Dataset is evaluated on three algorithms.

- DWM algorithm without drift indicator - Algorithm 3
- DWM with Hellinger drift indicator - Algorithm 3 with Algorithm 4
- DWM with GMM based drift indicator - Algorithm 3 with Algorithm 5

The primary objective of the evaluation is to compare the methods, with a particular focus on the improvements brought about by the subprocesses when applied to real-life data.

5.6.1 Evaluation hypothesis

Our hypothesis is quite similar to the one presented in Section 5.5. The primary difference is that, this time, we do not assume strict compliance with our assumptions regarding severe concept drift from Section 2.3, due to the complexity of real-world data.

- Proposed sub-procedure based on Hellinger concept drift detection (Algorithm 4) is able to improve the performance of the DWM algorithm (Algorithm 3) on real word malware detection dataset.
- Proposed GMM based sub-procedure (Algorithm 5) is able to improve the performance of the DWM algorithm (Algorithm 3) on real word malware detection dataset.

We assert that a given sub-procedure enhances the algorithm if the 95% confidence interval for the improvement in the F-1 score is positive and does not include 0. The F-1 score is used due to the significant label imbalance in the dataset, with over 22 benign samples for every malignant one. During the evaluation, we also employ multiple downsampling techniques, including random downsampling and benign-only downsampling.

5.6.2 Evaluation setting

In this subsection we present setting of the evaluation, where we compare DWM algorithm with and without sub-procedures on DREBIN dataset (Subsection 5.1.3).

General structure and results measurement methodology

We will compute confidence interval of mean improvement of F1-score in 10 runs of the procedures using random downsampling and label delay $l = 1000$. We perform 4 different downsampling studying effect of label imbalance on examined procedures.

Improvement is calculated as follows. First, we compute the values of tracked statistics (F1-score, Recall, Precision, and Accuracy). For each run, we subtract the value obtained using the DWM algorithm from the value obtained using the sub-procedure. We then use these differences (10 for each statistic) to construct a confidence intervals. Improvement is present if this interval does not include 0. For completeness, we also list confidence intervals of results for each algorithm.

Dataset preprocessing

The dataset used in this study exhibits significant class imbalance, with more than 22 benign samples for every malignant sample. To address this, we implemented a random downsampling of benign samples. However, in real-world scenarios, the benign or malignant nature of a sample is not known in advance, making this approach less realistic. Consequently, we also conducted experiments with random downsampling without prior knowledge of sample types. Specifically, for each sample, we randomly discarded it with a probability p_d. We evaluated four different downsampling strategies.

For both sub-procedures and types of downsampling, we used the same probabilities p_d, set at $p_d = 0.5$ and $p_d = \frac{5}{6} \approx 83.33\%$. The downsampling with $p_d = 0.5$ aimed to reduce the number of tested samples and create diverse datasets. The downsampling with $p_d = \frac{5}{6} \approx 83.33\%$ aimed to achieve a ratio of less than 4 benign samples to one malignant sample, mitigating the label imbalance.

The primary goal of Ceschin et al. [2022] is to demonstrate that, in real-life tracking of concept drift, it is beneficial to update classifiers and modify feature extraction methods based on measured concept drift. For instance, upon detecting concept drift, the TF-IDF transformer is updated. While this is a crucial problem associated with concept drift, it is not the focus of our work. Therefore, we assume that an adequate representation is provided, or more precisely, that all examples are processed through an initial sample transformation layer. Given that all features are textual, they need to be transformed into numeric ones. Similar to Ceschin et al. [2022], we use the TF-IDF process (see Attachment A.11), with a maximum of 10 features per column.

The created dataset is normalized to have zero mean and unit variance. In practice, the mean is subtracted, and each sample is scaled by the standard deviation.

Label imbalance modification

To address the severe label imbalance in our dataset, in addition to downsampling, we implemented two additional strategies. First, we used the F1-score (see Attachment A.12) as our performance metric instead of percentage accuracy. Second, we modified the way we utilized classifiers. Specifically, for all evaluations, we employed the Gaussian Naive Bayes classifier (Subsection 1.2.2), which incorporates the prior probabilities of the labels.

Severe label imbalance can cause the classifier to underperform on malware samples due to their low prior probability. To mitigate this issue, we smoothed the priors by 20%. This approach sets the base prior probability of each label to 20%, regardless of the actual distribution, with the remaining 60% distributed according to the observed data. This smoothing technique helps to ensure that malware samples are more likely to be correctly classified, despite their lower occurrence in the dataset.

DWM algorithm parameters

Parameters for modified DWM algorithm are set similarly as in Subsection 5.5.2, which means updating period $p = 50$, removal threshold $\Theta = 0.01$, decreasing factor $\beta = 0.5$ and learning parameter $t = 2$. This parameter setting is similar for all examined procedures.

GMM sub-procedure parameters

Algorithm 5 considers 10 feature subsets based on the original column each feature was created. Together, we assume 10 feature subsets each consisting of 10 features.

We set initialization period $t_b = 400$, outlier percentile $\beta = 0.05$, due to strong label imbalance and possible complex nature of samples the upper and lower bound for the sum of sp are set to $SP^U = 16000$, $SP^L = 8000$. From the same reason we drastically lowers incorporation parameters $n_{last} = 22$, $c^{id} = 15$. Since we scaled data to follow zero mean and unit variance we set initial covariance matrices to unit, scaled by 0.01. Similarly as in previous sections, we consider the prior threshold given by 3 flat Dirichlet distribution (Subsection 4.3.5).

Hellinger sub-procedure parameters

Algorithm 4 considers similarly to GMM-based procedure 10 feature subsets based on the original column each feature was created.

We set parameter of standard deviation $\gamma = 4$, window size $w = 1000$, B_{max} and B_{min} to 5% and 95% percentile of the data for all features. A number of minimal collected epsilons before concept drift detection is set to 100. After drift is detected classifier does not use the drifted feature subset for 2000 time steps. The drift parameter is set high (to 2000) since due to the great window size (1000) we do believe that the Gaussian Naive Bayes classifier needs a long time to adapt to a new concept.

5.6.3 Evaluation results and discussion

The main results of this evaluation are presented in Table 5.3, which displays the 95% confidence intervals for the differences in the performance of the DWM algorithm (Algorithm 3) with and without the developed sub-procedures (Algorithm 4 and Algorithm 5). These intervals represent the mean improvement over 10 runs, as described in Subsection 5.6.2.

The Hellinger sub-procedure improved the DWM algorithm's performance in two experiments, meeting the evaluation hypothesis, which requires the confidence interval to be positive and not include zero. Notably, all confidence intervals for the Hellinger sub-procedure have non-negative means.

Conversely, the results for the GMM-based sub-procedure are ambiguous and do not meet the evaluation hypothesis. While this sub-procedure primarily enhances Recall (Attachment A.12), indicating better malware detection, the overall improvements are inconsistent.

In the following subsection, we will briefly discuss the results and present Table 5.4, which displays the confidence intervals of output performance for each procedure and downsampling method.

Table 5.3: Improvement of sub-procedures over DWM algorithm using four various downsampled DREBIN dataset. Confidence intervals of results difference, as described in Subsection 5.6.2. Values of single algorithms statistics are presented in Table 5.4. p_d stands for probability of sample being discarded from the dataset.

	F1-diff.	Recall-diff.	Precision-diff.	Accuracy-diff.
Downsampling all samples, $p_d = \frac{5}{6}$				
GMM	-0.006 ± 0.009	0.025 ± 0.029	-0.008 ± 0.005	-0.012 ± 0.003
Hellinger	$\mathbf{0.006 \pm 0.004}$	0.011 ± 0.015	$\mathbf{0.004 \pm 0.003}$	0.001 ± 0.004
Downsampling all samples, $p_d = 0.5$				
GMM	-0.016 ± 0.007	-0.002 ± 0.013	-0.015 ± 0.005	-0.015 ± 0.002
Hellinger	0.006 ± 0.011	0.012 ± 0.022	0.004 ± 0.007	0.0 ± 0.001
Downsampling benign samples, $p_d = \frac{5}{6}$				
GMM	0.007 ± 0.014	$\mathbf{0.04 \pm 0.022}$	-0.011 ± 0.01	-0.009 ± 0.006
Hellinger	0.002 ± 0.008	0.002 ± 0.012	0.002 ± 0.005	0.001 ± 0.003
Downsampling benign samples, $p_d = 0.5$				
GMM	-0.012 ± 0.009	$\mathbf{0.019 \pm 0.013}$	-0.018 ± 0.007	-0.019 ± 0.003
Hellinger	$\mathbf{0.013 \pm 0.01}$	$\mathbf{0.023 \pm 0.013}$	0.008 ± 0.008	0.001 ± 0.003

Results further discussion

In Table 5.4, there are presented confidence intervals of the procedures for various downsampling of DREBIN dataset (Subsection 5.1.3). In the table, we can see that intervals are rather wide, since results for each run (downsampling with the same p_d) differ significantly. This is also the reason why we chose to track the differences.

The results presented in Table 5.3 suggest that the performance of GMM-based sub-procedure is connected with the label balance. We can see that in the most balanced scenario the procedure achieves its best performance.

Unfortunately, we have not identified a single cause of inconclusive results of the GMM sub-procedure evaluation presented in this section.

One cause of the results can be in complexity of the dataset, particularly of malware samples. In [Ceschin et al., 2022, Fig. 3 (a)] authors identify a great number of malware families, which with connection to only 5560 malignant samples could cause that clusters created by malware samples have too negligible prior and were considered as noisy.

Another aspect of our evaluation is that due to computational demands, we created only 10 features out of each feature column. Especially considering complex data it need not to be sufficient a number.

The results on the GMM sub-procedure can be also connected with exact implementation, where if the whole feature vector is detected as unsuitable we generate labels uniformly at random. This could partially explain the improvement of recall and deterioration of precision but it is in contrast with relevant stable information which is generally indicated by the good performance of Hellinger sub-procedure.

To get better intuition behind the run of GMM-based sub-procedure, we present in Attachment A.13 some results on downsampling using $p_d = \frac{5}{8}$. We

Table 5.4: Algorithms statistics for 4 various downsampled DREBIN dataset. 95% confidence intervals for the mean of 10 runs. p_d stands for probability of sample being discarded from the dataset.

	F1-score	Recall	Precision	Accuracy
Downsampling all samples, $p_d = \frac{5}{6}$				
GMM	0.214 ± 0.017	0.472 ± 0.053	0.139 ± 0.011	0.85 ± 0.008
Hellinger	0.226 ± 0.013	0.457 ± 0.034	0.151 ± 0.01	0.864 ± 0.007
DWM	0.221 ± 0.013	0.446 ± 0.025	0.147 ± 0.009	0.863 ± 0.006
Downsampling all samples, $p_d = 0.5$				
GMM	0.156 ± 0.02	0.309 ± 0.045	0.104 ± 0.013	0.857 ± 0.005
Hellinge	0.178 ± 0.024	0.323 ± 0.049	0.123 ± 0.016	0.872 ± 0.005
DWM	0.172 ± 0.023	0.311 ± 0.046	0.119 ± 0.016	0.872 ± 0.005
Downsampling benign samples, $p_d = \frac{5}{6}$				
GMM	0.523 ± 0.02	0.631 ± 0.037	0.448 ± 0.016	0.755 ± 0.001
Hellinge	0.518 ± 0.015	0.593 ± 0.028	0.461 ± 0.016	0.765 ± 0.01
DWM	0.516 ± 0.016	0.591 ± 0.029	0.459 ± 0.017	0.764 ± 0.01
Downsampling benign samples, $p_d = 0.5$				
GMM	0.277 ± 0.009	0.437 ± 0.015	0.203 ± 0.007	0.812 ± 0.003
Hellinge	0.302 ± 0.014	0.441 ± 0.023	0.229 ± 0.011	0.832 ± 0.004
DWM	0.289 ± 0.011	0.418 ± 0.019	0.221 ± 0.008	0.83 ± 0.003

specifically examine the evolution of the F1-score for the single run of GMM-based sub-procedure in time.

6. Future work

In this chapter, we present proposed directions of possible future work. This directions mainly focus on two aspects of problematic described in the thesis.

The first proposed direction stands in improvement of formalization which would separately be able to provide us with non-trivial properties and results. We see the possibility for this improvement in field of *Domain generalization*, which appears to have many similar aspect as our scenario and is able to produce solution directly based on the theoretical boundaries. We describe this field together with discussion of similarities and show of theoretical work in this domain in Attachment A.9.

The second proposed direction aims on better connection between classifier and the algorithm determining drift of feature vector parts. This idea of enhancing the impact of a classifier on obtaining information about drift in feature vector appears to well fulfilled by algorithm called *Gaussian process*. Despite the fact that we will not describe this algorithm in the thesis, we believe that it could be possible to use it for tracking the drift in feature subsets by testing whether accordingly restricted feature vector improves a prediction certainty or not.

Conclusion

In this thesis, we investigate a scenario tailored to real-life malware detection, characterized by data streams significantly impacted by concept drift and label delay. This scenario exhibits unique properties: the presence of highly decisive but severely drifting features, such as specific filenames or mutexes, and the existence of relevant stable information, such as connection types or monetization methods, which remain consistent over time.

We systematically study this environment. We present a formalization of this scenario and highlight the shortcomings of current techniques, particularly how models tend to over-rely on highly indicative drifting features, leading to unnecessary performance drops when these features drift. To address this issue, we propose two solutions designed to run alongside the underlying model, providing information about drift in specific feature subsets. The first solution utilizes concept drift detection based on the Hellinger distance, while the second employs a modified incremental Gaussian Mixture Model algorithm to identify drifted parts of the incoming feature vector.

We introduced a custom dataset to capture the properties of the overall scenario. Using this dataset and the Dynamic Weighted Majority (DWM) algorithm as the underlying model, we demonstrated that our proposed techniques can enhance this robust algorithm. Furthermore, we present promising results of our procedures on the real-life DREBIN dataset (Arp et al. [2014]).

The main takeaway of this thesis is that in scenarios exhibiting properties of severely drifting features and relatively stable information, it is beneficial to address the problem by excluding severely drifting features, as done in the proposed solutions. This approach improves the performance and reliability of malware detection systems in dynamic environments.

Bibliography

Juan M Acevedo-Valle, Karla Trejo, and Cecilio Angulo. Multivariate regression with incremental learning of gaussian mixture models. In *CCIA*, pages 196–205, 2017.

Isabela Albuquerque, João Monteiro, Mohammad Darvishi, Tiago H Falk, and Ioannis Mitliagkas. Generalizing to unseen domains via distribution matching. *arXiv preprint arXiv:1911.00804*, 2019.

Giacomo Aletti and Alessandra Micheletti. A clustering algorithm for multivariate data streams with correlated components. *Journal of Big Data*, pages 1–20, 2017.

Daniel Arp, Michael Spreitzenbarth, Malte Hübner, Hugo Gascon, and Konrad Rieck. Drebin: Effective and explainable detection of android malware in your pocket. 2014.

Aya Ayadi, Oussama Ghorbel, Abdulfattah M Obeid, and Mohamed Abid. Outlier detection approaches for wireless sensor networks: A survey. *Computer Networks*, pages 319–333, 2017.

Peter Bajorski. *Statistics for imaging, optics, and photonics*, volume 808. 2011.

Daniel Barbara and Ping Chen. Tracking clusters in evolving data sets. In *FLAIRS Conference*, pages 239–243, 2001.

Anna Bartkowiak. Should normal distribution be normal? the student's t alternative. In *6th international conference on computer information systems and industrial management applications (cisim'07)*, pages 3–8, 2007.

Firas Bayram, Bestoun S. Ahmed, and Andreas Kassler. From concept drift to model degradation: An overview on performance-aware drift detectors. 2022.

Christopher M Bishop. Pattern recognition and machine learning. *Springer google schola*, pages 5–43, 2006.

Ane Blázquez-García, Angel Conde, Usue Mori, and Jose A Lozano. A review on outlier/anomaly detection in time series data. *ACM Computing Surveys (CSUR)*, pages 1–33, 2021.

Abdelhamid Bouchachia and Charlie Vanaret. Incremental learning based on growing gaussian mixture models. In *2011 10th International Conference on Machine Learning and Applications and Workshops*, pages 47–52, 2011.

Ander Carreño, Iñaki Inza, and Jose A Lozano. Analyzing rare event, anomaly, novelty and outlier detection terms under the supervised classification framework. *Artificial Intelligence Review*, pages 3575–3594, 2020.

Fabrício Ceschin, Marcus Botacin, Heitor Murilo Gomes, Felipe Pinagé, Luiz S. Oliveira, and André Grégio. Fast and furious: On the modelling of malware detection as an evolving data stream. *Expert Systems with Applications*, 2022.

Tony F Chan, Gene H Golub, and Randall J LeVeque. Updating formulae and a pairwise algorithm for computing sample variances. In *COMPSTAT 1982 5th Symposium held at Toulouse 1982: Part I: Proceedings in Computational Statistics*, pages 30–41, 1982.

Girish Chandrashekar and Ferat Sahin. A survey on feature selection methods. *Computers and Electrical Engineering*, pages 16–28, 2014.

Tamraparni Dasu, Shankar Krishnan, Suresh Venkatasubramanian, and Ke Yi. An information-theoretic approach to detecting changes in multidimensional data streams. 2006.

Javier Diaz-Rozo, Concha Bielza, and Pedro Larrañaga. Clustering of data streams with dynamic gaussian mixture models: An iot application in industrial processes. *IEEE Internet of Things Journal*, pages 3533–3547, 2018.

Yu Ding, Lei Wang, Bin Liang, Shuming Liang, Yang Wang, and Fang Chen. Domain generalization by learning and removing domain-specific features. In *Advances in Neural Information Processing Systems*, pages 24226–24239, 2022.

Gregory Ditzler and Robi Polikar. Hellinger distance based drift detection for nonstationary environments. pages 41–48, 2011.

Paulo Martins Engel and Milton Roberto Heinen. Incremental learning of multivariate gaussian mixture models. In *Advances in Artificial Intelligence–SBIA 2010: 20th Brazilian Symposium on Artificial Intelligence, São Bernardo do Campo, Brazil, October 23-28, 2010. Proceedings 20*, pages 82–91, 2010.

Elaine R Faria, Isabel JCR Gonçalves, André CPLF de Carvalho, and João Gama. Novelty detection in data streams. *Artificial Intelligence Review*, pages 235–269, 2016.

Igor Goldenberg and Geoffrey Webb. Survey of distance measures for quantifying concept drift and shift in numeric data. *Knowledge and Information Systems*, 2019.

Heitor Murilo Gomes, Jean Paul Barddal, Fabrício Enembreck, and Albert Bifet. A survey on ensemble learning for data stream classification. *ACM Computing Surveys (CSUR)*, pages 1–36, 2017.

Cyril Goutte and Eric Gaussier. A probabilistic interpretation of precision, recall and f-score, with implication for evaluation. pages 345–359, 2005.

David A Harville. Matrix algebra from a statistician's perspective, 1998.

Milton Roberto Heinen. A connectionist approach for incremental function approximation and on-line tasks. 2011.

Hammer Hinder, Vaquet. Suitability of different metric choices for concept drift detection. In *Advances in Intelligent Data Analysis XX*, pages 157–170, 2022.

Thorsten Joachims et al. A probabilistic analysis of the rocchio algorithm with tfidf for text categorization. pages 143–151, 1997.

D Keehn. A note on learning for Gaussian properties. *IEEE Transactions on Information Theory*, pages 126–132, 1965.

J. Zico Kolter and Marcus A. Maloof. Dynamic Weighted Majority: An ensemble method for drifting concepts. 2007.

Jogikalmat Krithikadatta. Normal distribution. *Journal of Conservative Dentistry and Endodontics*, pages 96–97, 2014.

Vipin Kumar and Sonajharia Minz. Feature selection: a literature review. *SmartCR*, pages 211–229, 2014.

David M Lane, David Scott, Mikki Hebl, Rudy Guerra, Dan Osherson, and Heidi Zimmer. Introduction to statistics, online edition. *Rice University, University of Houston Clear Lake, and Tufts University*, 2017.

Da Li, Yongxin Yang, Yi-Zhe Song, and Timothy M Hospedales. Deeper, broader and artier domain generalization. In *Proceedings of the IEEE international conference on computer vision*, pages 5542–5550, 2017.

Jiayu Lin. On the dirichlet distribution. *Department of Mathematics and Statistics, Queens University*, pages 10–11, 2016.

Anjin Liu, Jie Lu, and Guangquan Zhang. Concept drift detection via equal intensity K-Means space partitioning. pages 3198–3211, 2021.

Jie Lu, Anjin Liu, Fan Dong, Feng Gu, João Gama, and Guangquan Zhang. Learning under concept drift: A review. pages 2346–2363, 2019.

Goeffrey J McLachlan. Mahalanobis distance. *Resonance*, pages 20–26, 1999.

Richard D Morey, Rink Hoekstra, Jeffrey N Rouder, Michael D Lee, and Eric-Jan Wagenmakers. The fallacy of placing confidence in confidence intervals. *Psychonomic bulletin & review*, pages 103–123, 2016.

Kevin P Murphy et al. Naive bayes classifiers. *University of British Columbia*, pages 1–8, 2006.

Andrew Ng and Michael Jordan. On discriminative vs. generative classifiers: A comparison of logistic regression and naive bayes. In *Advances in Neural Information Processing Systems*. MIT Press, 2001.

F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. 2011.

Rafael Pinto and Paulo Engel. Scalable and incremental learning of gaussian mixture models. *arXiv preprint arXiv:1701.03940*, 2017.

Rafael Coimbra Pinto and Paulo Martins Engel. A fast incremental gaussian mixture model. *PloS one*, page e0139931, 2015.

Gerard Salton, Anita Wong, and Chung-Shu Yang. A vector space model for automatic indexing. pages 613–620, 1975.

Jack Sherman and Winifred J Morrison. Adjustment of an inverse matrix corresponding to a change in one element of a given matrix. *The Annals of Mathematical Statistics*, pages 124–127, 1950.

Shuping Sun, Yaonan Tong, Biqiang Zhang, Bowen Yang, Long Yan, Peiguang He, and Hong Xu. A novel adaptive methodology for removing spurious components in a modified incremental gaussian mixture model. *International Journal of Machine Learning and Cybernetics*, pages 551–566, 2023.

Marco Taboga. Functions of random vectors and their distribution, lectures on probability theory and mathematical statistics. Kindle Direct Publishing, 2021.

Jinita Tamboli and Madhu Shukla. A survey of outlier detection algorithms for data streams. In *2016 3rd International Conference on Computing for Sustainable Global Development (INDIACom)*, pages 3535–3540, 2016.

Hongzhi Wang, Mohamed Jaward Bah, and Mohamed Hammad. Progress in outlier detection techniques: A survey. *Ieee Access*, pages 107964–108000, 2019.

Kapil K Wankhade, Snehlata S Dongre, and Kalpana C Jondhale. Data stream classification: a review. pages 239–260, 2020.

Osamu Watanabe. Simple sampling techniques for discovery science. *IEICE Transactions on Information and Systems*, pages 19–26, 2000.

William J Welch. Algorithmic complexity: three np-hard problems in computational statistics. *Journal of Statistical Computation and Simulation*, pages 17–25, 1982.

Thomas H Wonnacott and Ronald J Wonnacott. Introductory statistics for business and economics. 1990.

CF Jeff Wu. On the convergence properties of the em algorithm. *The Annals of statistics*, pages 95–103, 1983.

Jingkang Yang, Kaiyang Zhou, Yixuan Li, and Ziwei Liu. Generalized out-of-distribution detection: A survey. *arXiv preprint arXiv:2110.11334*, 2021.

A. Attachments

In Attachments, we introduce several topics that are frequently referenced throughout the thesis. However, they are not deemed essential for the development of the primary techniques.

A.1 Other types of concept drift

In Subsection 2.2.2 we described sudden concept drift as essential scenario we are interested in. In this section, we will follow up on this subsection and present other frequent types of concept drift. Specifically, we will define gradual, incremental and reoccurring concept drift.

Generally speaking we can describe the remaining three types as follows. For gradual it holds that original concept is replaced in longer period of time and we can observe period of overlay of these two concepts. The incremental is typical by slowly change, where the original concept incrementally changes up to new concept. For the reoccurring drift, original concept reoccurs in the future. Illustration of this division can be seen in Figure A.1, which follows [Lu et al., 2019, Fig. 4].

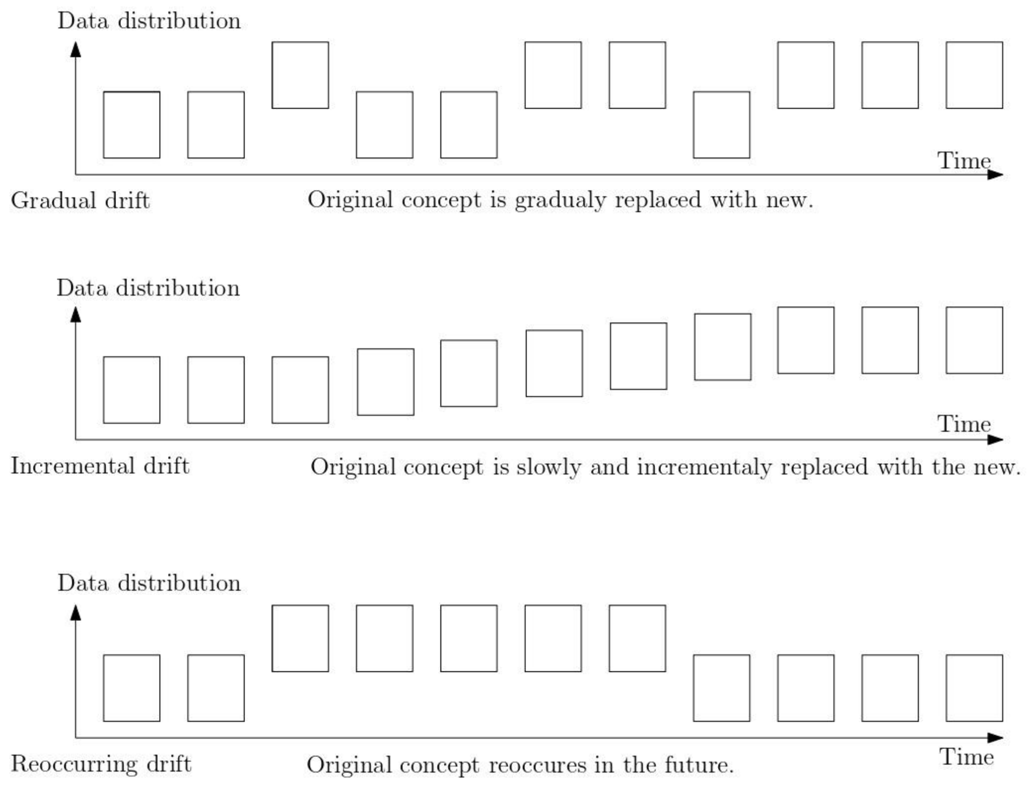

Data distribution
Time
Gradual drift
Original concept is gradualy replaced with new.
Data distribution
Time
Incremental drift
Original concept is slowly and incrementaly replaced with the new.
Data distribution
Reoccurring drift
Original concept reoccures in the future.
Time

Figure A.1: Illustration of the concept drift types according to way the distribution of samples in the stream changes over time.

For Definition 11 of the concept drift types we assume setting presented in Subsection 2.2.2.

Definition 11. *Considering notation similar as in Definition 2 and problem setup as in Subsection 2.2.2, for given sample set S_{0,t_n} we call a concept drift*

- *gradual if the drift is not sudden and there exist time points $t_0, t_1 \in \mathbb{N}$, $0 < t_0 < t_1 < t_n$ and distinct distributions $P_{<t_0}(X, y)$ and $P_{>t_1}(X, y)$ such that $P_t(X, y) = P_{<t_0}(X, y)$ for all $t < t_0$ and $P_t(X, y) = P_{>t_1}(X, y)$ for all $t > t_1$ and for all t satisfying $t_0 \leq t \leq t_1$ it holds that $P_t(X, y)$ is equal to either $P_{<t_0}(X, y)$ or $P_{>t_1}(X, y)$.*

- *incremental if the drift is not sudden and there exist time points $t_0, t_1 \in \mathbb{N}$, $0 < t_0 < t_1 < t_n$ and distinct probability distributions $P_{<t_0}(X, y)$ and $P_{>t_1}(X, y)$ such that $P_t(X, y) = P_{<t_0}(X, y)$ for all $t < t_0$ and $P_t(X, y) = P_{t>t_1}(X, y)$ for all $t > t_0$. For t satisfying $t_0 \leq t \leq t_1$ it holds that $\delta(P_{<t_0}(X, y), P_t(X, y)) \leq \delta(P_{<t_0}(X, y), P_{t+1}(X, y))$ and $\delta(P_{>t_1}(X, y), P_t(X, y)) \geq \delta(P_{>t_1}(X, y), P_{t+1}(X, y))$, for some metric δ.*

- *reoccurring if for some given minimal time period $m \in \mathbb{N}$ there exists time points $t_0, t_1 \in \mathbb{N}$ and probability distribution $P_{reoccur}(X, y)$ such that $0 < t_0 < t_1 < t_n$, $t_1 - t_0 > m$, for all t satisfying $0 \leq t < t_0$ or $t_1 < t < t_n$ it holds $P_t(X, y) = P_{reoccur}(X, y)$ and for all t such that $t_0 \leq t \leq t_1$ it holds $P_t(X, y) \neq P_{reoccur}(X, y)$.*

Knowledge of the concept drift type of the dataset can be very beneficial in real world scenarios since we can modify our technique accordingly. Example of modified approaches for different concept drift types can be seen in Subsection 1.3.1.

A.2 Location of drift lemma

In this attachment we present proof of Lemma 1 presented in Subsection 1.1.3. This lemma is as follows.

Lemma 6. *Let $X = \{X^1, \dots, X^d\}$ be a feature vector, where $d \in \mathbb{N}$ and all features in the vector are mutually independent, conditional on the label $y \in \mathcal{Y}$. That means $P(X^i | X^1, \dots, X^{i-1}, y) = P(X^i | y)$. Then it holds that $P(X^1, \dots, X^d, y) = P(y)^{1-d} \prod_{i=1}^{d} P(X^i, y)$.*

Proof. First, we show basic probabilistic lemma about decomposition of joint probability mostly known as chain rule, which we will use in Lemma 1.

Lemma 7. *For $n \in \mathbb{N}$ let $X^1, \dots, X^n$ be random events for which it holds that their joint probability is strictly greater than zero. Then it holds that $P(X^1, \dots, X^n) = P(X^1) \prod_{i=2}^{n} P(X^i | X^1, \dots X^{i-1})$.*

Proof. For $i = 2$ equality comes from the definition of joint probability

$$P(X^1, X^2) = P(X^1)P(X^2 | X^1)$$

For induction step let assume that the equation holds for $n = j - 1$, for $n = j$ we can see that

$$P(X^1, \dots, X^j) = P(X^1, \dots, X^{j-1}, X^j) =$$
$$P(X^j | X^1 \dots, X^{j-1}) P(X^1 \dots, X^{j-1}) = P(X^1) \prod_{i=2}^{j} P(X^i | X^1, \dots X^{i-1}).$$

First we used definition of joint probability and then induction hypothesis for $P(X^1, \dots, X^{j-1})$. □

The main lemma comes from

$$P(X^1, \dots, X^d, y) = P(X^1, y) \prod_{i=2}^{d} P(X^i | X^1, \dots X^{i-1}, y) =$$
$$P(X^1, y) \prod_{i=2}^{d} P(X^i | y) = P(y)^{1-d} \prod_{i=1}^{d} P(X^i, y).$$

Where for the first equality we used Lemma 7, the second is obtained from assumption of independence and the last expression is derived by applying equality $P(X^i | y) = \frac{P(X^i, y)}{P(y)}$ which comes from definition. □

A.3 Feature selection

As highlighted in Section 3.3, our general solution relies on selecting features suitable for the prediction process. This task closely resembles feature selection. To provide a broader context, we describe the objectives of this field and briefly discuss its connection to our solutions.

According to [Chandrashekar and Sahin, 2014, Section 1], feature selection aims to find a subset of features that can efficiently describe the original data, reduce noise and irrelevant variables, and still provide good prediction results. A more detailed description of feature selection can be found in Kumar and Minz [2014], where a definition is provided in [Kumar and Minz, 2014, Definition 7], which we present as Definition 12.

Definition 12. *Feature selection is the following process. Let A be an original set of features and $L()$ be an evaluation criterion $L : A' \subseteq A \to \mathbb{R}$ to be maximized (optimized). The candidate subset of features can be considered under the following considerations*

- *Let $|A| = m$ and $|A'| = n$, then $L(A')$ is maximized, where $m > n$ and $A' \subset A$.*
- *Set a threshold Θ such that $L(A') > \Theta$ to find a subset of features with the smallest number, where $m > n$.*
- *Finding the optimization function $L(A')$ with optimal feature subsets $|A'|$, where optimal feature subset refers to subset of features for which accuracy of the induced classifier is maximal.*

From Definition 12, it is clear that feature selection is highly relevant to our problem. Particularly, if we consider the evaluation criteria $L()$ to reflect concept drift and reevaluate the feature selection process at each time step, we see a direct connection. This approach is applied across all our proposed solutions, with each defining different evaluation criteria $L()$. In the first solution, presented in Section 4.2, we measure drift based on changes in distribution using the Hellinger distance. In the second approach, described in Section 4.3, the criteria assess how well each arriving sample fits into the distribution modelled by the Gaussian Mixture Model algorithm.

However, our objectives and assumptions diverge significantly from typical feature selection. This divergence is primarily due to our strong assumption of feature dependency knowledge and the fact that our primary goal is not to identify the smallest subset of relevant features by excluding the least relevant ones. Instead, we often aim to discard highly relevant features at the appropriate time. Therefore, categorizing our problem strictly as feature selection could be misleading.

A.4 EM algorithms

In this section, we generalize clustering algorithms presented in Section 1.5 into more general class of algorithms called EM algorithms. Approach described in following text is followed when deriving the incremental algorithms presented in Section 1.6.

Both algorithms described in Subsection 1.5.4 and Subsection 1.5.2 are generalized by class of so called EM algorithms. This algorithms work in setting such that there is joint distribution $P(X, Z; w)$, where X is observed variable, Z is latent variable and w parametrizes both of them. We want to maximize the log-likelihood $\log P(X; w) = \log \sum_Z P(X, Z; w)$ with respect to w. For our process described in Subsection 1.5.4, X is feature vector, Z is assignment to the clusters and w are parameters. Note that Z can be any variable which we made (is not given) to improve our solution and the equality above is simple use of the law of total probability. There is difficult calculation of logarithm of the sums. Therefore, we split the computation into two steps, the evaluating step (E-step) and the log-likelihood maximization step (M-step). There we replace the log-sum by expectation of logarithm under the distribution $P(Z|X; w)$. In the algorithm we fix parameters w_{fix} and proceed by E/M-steps which looks as follows

- **E step** $Q(w|w_{fix}) = \mathbb{E}_{Z|X,w_{fix}}(\log P(X, Z; w))$
- **M step** $w_{fix} \leftarrow argmax_w Q(w|w_{fix})$.

When considering the validity of this algorithm, it can be seen in Wu [1983] that for EM algorithms it holds that log-likelihood increases in every step but whether it converges to local or global optimum generally depends on initial setting.

A.5 Adaptive window size

In this section, we present the theory of adaptive sampling and adaptive window size in relation to tracking concept drift. This theory is closely linked to a potential solution for determining the duration for which an outlier sample should be retained in the retention set, as discussed in Subsection 4.3.4. Although we reference this theory multiple times throughout the thesis, we do not directly apply it in our solution. Instead, it often serves to illustrate a more theoretically robust approach to addressing issues that arise during the process.

First, we introduce the basic theory of sampling based on Watanabe [2000], and then we discuss its potential application to problems associated with concept drift.

Consider a random variables $\mathcal{X}_i$ which is 1 if the i-th sample is not detected as outlier and 0 if otherwise. Using these, we can construct another random variable $\mathcal{X} = \sum_{i=1}^{n} \mathcal{X}_i$. Assuming that all $\mathcal{X}_i$ have probability of being 1 equal to p, in other words, prior probability that arriving sample is clustered appropriately equals to p. Then the expected value of $\mathcal{X}$ is equal to np. The problem is that typically we do not know p, we can only estimate this value by $\hat{p}$ which is generally set to a number of non-outliers divided by a number of observations. Therefore, we determine minimal amount of samples we need to get estimate of p close enough. This we do according to [Watanabe, 2000, Theorem 2.5].

Theorem 8. *Using above notation, for any δ, $0 < \delta < 2$ and relative error ϵ, $0 < \epsilon < 1$, if n satisfies*

$$n > \frac{3p}{\epsilon^2} \ln\left(\frac{2}{\delta}\right),$$

then it holds that

$$P(|p - \hat{p}| \leq \epsilon) > 1 - \delta,$$

where $\hat{p}$ is estimated as $\hat{p} = \frac{\sum_{i=1}^{n} \mathcal{X}_i}{n}$.

In above, we worked with an absolute error of approximation p. It can be beneficial to consider also relative error, where the bound for number of samples n is presented in next theorem, which follows [Watanabe, 2000, Theorem 3.1].

Theorem 9. *Using above notation, for any δ, $0 < \delta < 2$ and relative error ϵ, $0 < \epsilon < 1$, if n satisfies*

$$n > \frac{3}{\epsilon^2 p} \ln\left(\frac{2}{\delta}\right),$$

then it holds that

$$P(|p - \hat{p}| \leq \epsilon p) > 1 - \delta,$$

where $\hat{p}$ is estimated as $\hat{p} = \frac{\sum_{i=1}^{n} \mathcal{X}_i}{n}$.

Presented sampling is based on idea that at the beginning we determine number of samples we need. This can be potentially improved by taking into account the values we receive during the process. This idea leads to the so called *adaptive sampling*. There we count number of positive samples and check whether it exceeds predefined threshold A, if it does, then we stop and calculate mean by dividing the number of positive samples by the overall number of examined samples. One of such thresholds is presented in [Watanabe, 2000, Theorem 3.2], which we formulate as Theorem 10.

Theorem 10. *Using above notation, for any δ, $0 < \delta < 2$ and relative error ϵ, $0 < \epsilon < 1$, if threshold for positive samples A in adaptive sampling satisfies*

$$A > \frac{3(1+\epsilon)}{\epsilon^2 p} \ln\left(\frac{2}{\delta}\right), \tag{A.1}$$

then it holds that

$$P(|p - \hat{p}| \leq \epsilon p) > 1 - \delta,$$

where $\hat{p}$ is estimated as $\hat{p} = \frac{\sum_{i=1}^{n} \mathcal{X}_i}{n}$.

This adaptive threshold can also provide an estimate on the number of sampled elements n. This is done in [Watanabe, 2000, Corollary 3.5] which is of the following form.

Theorem 11. *For any δ, $0 < \delta < 2$ and relative error ϵ, $0 < \epsilon < 1$, using adaptive sampling with threshold for positive samples A set to smallest integer satisfying Equation* (A.1)*, then with probability greater than $1 - \frac{\delta}{2}$, sample size n satisfies*

$$n \leq \frac{3(1+\epsilon)}{(1-\epsilon)\epsilon^2 p} \ln\left(\frac{2}{\delta}\right). \tag{A.2}$$

This theory is used in Barbara and Chen [2001] for tracking concept drift. Generally, they are provided with n samples, where n is the lower bound of Equation (A.2) computed using $\hat{p}$ estimated in previous step. After proceeding batch we compare the number of non-outliers to A computed as the lower bound of Equation (A.1) and if A is greater then observed number of well fitted samples, the concept drift is detected.

In [Diaz-Rozo et al., 2018, Section II C], the theory is applied to adaptive window size. The authors use Equation A.2 to determine the adaptive window size, optimizing the number of samples to be kept in memory. Intuitively, this makes sense because the probability of a non-outlier is in the numerator; thus, if the concept is well-represented, fewer samples are retained, and vice versa. However, their objective differs from ours, as they use this window to measure drift and retrain the model when significant changes are detected. We describe this approach because it determines the number of recently significant samples, which is closely related to the issue discussed in Subsection 4.3.4.

A.6 Confidence intervals

In this section we describe common method used for tests reliability and repeatability called *confidence intervals.* This method is used many times in this thesis. Therefore, we consider it appropriate to specify how exactly this method works.

An important part of the testing and generally observing any properties of the examined algorithm is to determine its statistic relevance which is crucial for possible result interpretation. One of the widely used approach for stating some statistical certainty is concept called confidence interval, which we will describe in this subsection according to [Lane et al., 2017, Chapter 10] and Morey et al. [2016].

The general definition of the confidence interval can be seen in [Morey et al., 2016, Definition 1]. This definition is presented as Definition 13. We note that according to Morey et al. [2016] the probability equality from the Definition 13 can be often swapped with: *at least* with the probability $P\%$.

Definition 13. *An $P\%$ confidence interval for parameter Θ is an interval (L, U) generated by a procedure that in repeated sampling has an $P\%$ probability of containing the true value of Θ, for all possible values of Θ.*

In this thesis, we will mainly estimate the mean and we will do it usually with only small number of samples, typically with only 10 and also without knowledge of the exact value of variance. This is scenario where it is common to approximate the random variable producing the observation by student t-distribution. This distribution has probability density function equal to

$$f(t) = \frac{\Gamma(\frac{\nu+1}{2})}{\sqrt{\pi}\Gamma(\frac{\nu}{2})}\Big(1 + \frac{t^2}{\nu}\Big)^{-\frac{\nu+1}{2}},$$

where ν is the number of degrees of freedom and Γ is the gamma function, which we already met in Section 1.4. We will denote this distribution as t_ν. More to this distribution especially with focus on comparison to Normal distribution can be seen in Bartkowiak [2007], but for our purpose it is sufficient to know that it is generalization of the Normal distribution controlled by parameter ν, which generally affects a tales of the probability distribution function. We illustrate this in Figure A.2 which is similar to [Bartkowiak, 2007, Figure 2]. For our use in confidence intervals, an idea of using t-distribution over normal distribution is in incorporation of a less certainty, which is directed by the parameter ν.

Now, we present just a sketch of deriving confidence interval computation. Generally, we are looking for interval (L, U) for which it holds that estimated mean $\hat{\mu}$ satisfies $P(\hat{\mu} \in (L, U)) = 0.95$. For estimating confidence interval, from n samples $X_1, \ldots, X_n$ and assuming $X_i \sim \mu + \sigma T$ for $T \sim t_{n-1}$, which is called location-scale t-distribution with location parameter μ and scale parameter σ. Hence for estimation of the mean we get

$$\frac{\hat{\mu} - \mu}{\frac{\hat{\sigma}}{\sqrt{n}}} \sim t_{n-1}, \tag{A.3}$$

where $\hat{\mu}$ is estimate of the mean μ and $\hat{\sigma}$ is sample variance estimating σ, both using samples $X_1, \ldots, X_n$. Now, we use concept which is called quantile. Let $qt_{n-1}(p)$ be a p percent quantile of the t_{n-1} distribution, then it means that it is a such number that if we sample from variable which follows t_{n-1} distribution with exactly p percent we sample value smaller or equal to the $qt_{n-1}(p)$. From this definition and Equation (A.3) we can see that

$$P(\frac{\hat{\mu} - \mu}{\frac{\hat{\sigma}}{\sqrt{n}}} \leq qt_{n-1}(p)) = p. \tag{A.4}$$

It also works on the other way and we can get

$$P(\frac{\hat{\mu} - \mu}{\frac{\hat{\sigma}}{\sqrt{n}}} \geq qt_{n-1}(1-p)) = p. \tag{A.5}$$

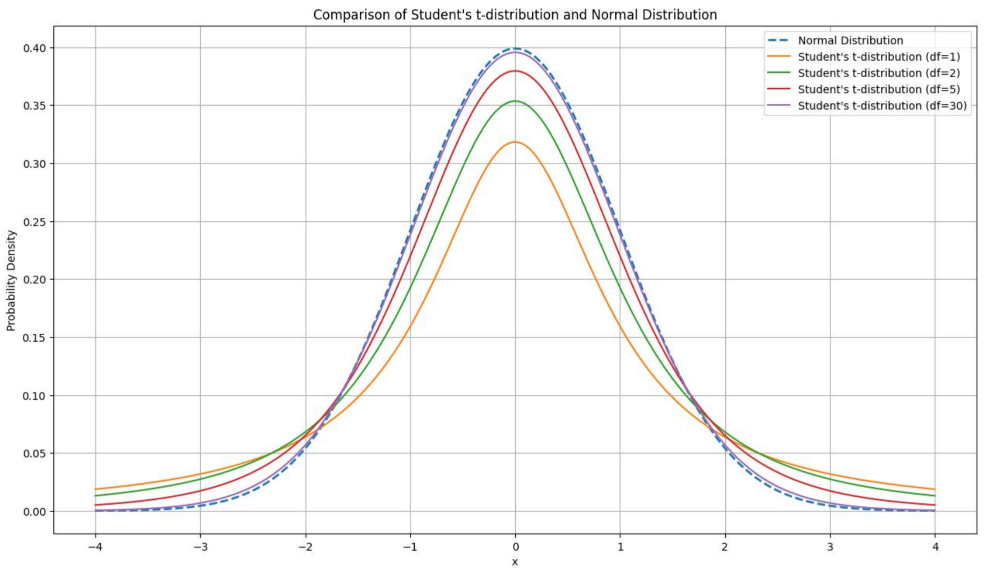


Figure A.2: Comparison of probability density functions of student t-distribution with various degrees of freedon (df) and Normal distribution. For all functions we consider 0 mean and standard deviation equal to 1.

Now, we will show how can be these equations rewritten. From Equation (A.4) we get

$$P(-\mu \leq -\hat{\mu} + \frac{\hat{\sigma}}{\sqrt{n}} qt_{n-1}(p)) = p$$
$$P(\mu \geq \hat{\mu} - \frac{\hat{\sigma}}{\sqrt{n}} qt_{n-1}(p)) = p.$$

Hence the interval given by these equations is

$$(\hat{\mu} - \frac{\hat{\sigma}}{\sqrt{n}} qt_{n-1}(p), \infty).$$

For Equation (A.5) using same approach we can see that

$$P(-\mu \geq -\hat{\mu} + \frac{\hat{\sigma}}{\sqrt{n}} qt_{n-1}(1-p)) = p$$
$$P(\mu \leq \hat{\mu} - \frac{\hat{\sigma}}{\sqrt{n}} qt_{n-1}(1-p)) = p.$$

This sets the second interval to

$$(-\infty, \hat{\mu} - \frac{\hat{\sigma}}{\sqrt{n}} qt_{n-1}(1-p)).$$

Now if we consider $p \geq 0.5$ it holds that

$$P(qt_{n-1}(1-p) \leq \frac{\hat{\mu} - \mu}{\frac{\hat{\sigma}}{\sqrt{n}}} \leq qt_{n-1}(p)) = 1 - (1-p) - (1-p) = -1 + 2p,$$

which can be considered as two inequalities, where we for each of them use above derived intervals, whose combination provide us with confidence interval of the form

$$(\hat{\mu} - \frac{\hat{\sigma}}{\sqrt{n}} qt_{n-1}(p), \hat{\mu} - \frac{\hat{\sigma}}{\sqrt{n}} qt_{n-1}(1-p)).$$

Note also that t-distribution is symmetric and therefore $qt_{n-1}(1-p) = -qt_{n-1}(p)$, which can provide us with interval given by

$$(\hat{\mu} - \frac{\hat{\sigma}}{\sqrt{n}} qt_{n-1}(p), \hat{\mu} + \frac{\hat{\sigma}}{\sqrt{n}} qt_{n-1}(p)). \tag{A.6}$$

At the end, we note that according to Morey et al. [2016] there are many possible fallacy about confidence intervals which we should be aware of. Therefore, we highlight that we strictly follow Definition 13, which for our case mean that main interpretation of the p-percent confidence interval (L, U) for mean value of process P is that if we reevaluate this process, create mean of same number of samples, then with probability p output value of our evaluation will be in interval (L, U).

A.7 Abnormality detection

In this section, we present wider context of outlier and novelty detection, which are important terms for our solution based on GMM procedure (Section 4.3). Discussion serves as extension of Section 1.7.

There are lots of terms highly connected with concept drift. Usually, these terms focus on specific tasks that are slightly different from ours, but their solutions in a modified form can be beneficial in solving our problem. This is the case of set of fields we generally called abnormality detection. We describe some basic directions we can find behind this term and which appear in Section 4.3 as sub-tasks of the final Algorithm 5. In following paragraphs we will present wider discussion of terms described in Section 1.7.

Major objective of this thesis is ability of tracking drift. In Section 4.2 we do this by studying change at the distribution level. But we might also want to be able to track the change at a sample level. From this point of view, we are interested in samples, which are abnormal to those we have seen so far. Generally, abnormality appears to be a reasonable assumption for samples that are influenced by the concept drift.

There are many fields dealing with study of abnormality detection. The most interesting for us are areas of *Outlier detection* and *Novelty detection*. Unfortunately, both of these areas lacks exact definitions and formalization but still we give some in Subsection 1.7.1.

Note that in our scenario novelty detection and outlier detection may meet and overlap. We will show this on an example, which also generally demonstrates a situation we deal with in Section 4.3. Suppose that we receive new sample which does not fit into distribution we have seen so far. In this moment, we mark this sample as an outlier but in time more similar samples arrive then our conclusion would be that the sample was in fact the first example of the newly arriving concept. Therefore, we are in the field of novelty detection scenario and we should consider changes in the model.

Our solution presented in Section 4.3 is based on technique which is common to both fields. It is clustering algorithm called Gaussian Mixture Model. Still each of the presented terms will affect the algorithm in its own way. In Section 1.7, we present brief survey of related outlier detection techniques from which we later derive a tailor-made solution for our problem. Novelty detection has specific role in Algorithm 5 presented in Section 4.3. This is because next of searching for pattern in detected outliers we will also deal with question whether is this pattern incorporated by related classifier (see Section 3.3) or not. We study these tasks across Section 4.3.

For sake of completeness, note that there are many not clearly defined terms which are highly related to ones we described in this section. These are for example *Out of distribution detection*, *Anomaly detection*, *Rare event detection* and many others which are used in various and often overlapping meanings as can be seen in Carreño et al. [2020]. Also techniques used in all mentioned detection field are usually same or very similar. We will not discuss them any further in this thesis since it would only unnecessarily complicate terminology.

A.8 Exponential prior probability threshold

In this section, we describe exponential probability threshold associated with Subsection 4.3.5.

In [Sun et al., 2023, Subsection 3.1], the study of π_{min} involved generating N_k random samples from a mixture of K bivariate normal distributions with unit covariance matrices and overlaps occurring in the mixture. The extensive search considered mixtures with 3 to 10 components, with the number of assigned samples ranging from 500 to 2000, assuming uniform prior probabilities for all mixtures. The data were sequentially generated and processed to create and update clusters. These processes, described in [Sun et al., 2023, Subsections 2.1 and 2.2], are briefly summarized in Subsection 1.8.2 and Subsection 1.8.1. At the end, spurious clusters were removed, and their maximum probabilities were recorded. These probabilities were used to compute a threshold curve $\frac{\pi}{10}e^{-\frac{\pi}{10}K}$, representing π_{min} as a function of the number of clusters K. Clusters were deemed spurious based on their small probabilities and significant distances from the centers of generated data.

We compare the thresholds obtained using Dirichlet distribution based threshold with various α and δ values to exponential threshold of Sun et al. [2023], which determined the desired parameter through extensive search. This comparison is illustrated in Figure A.3.

A.9 Domain generalization

Domain generalization is a field of study distinct from our work as it does not address concept drift, a crucial aspect for us. Despite this, it offers major similarities, particularly in its formalization, which is sophisticated enough to provide useful theorems for algorithm development.

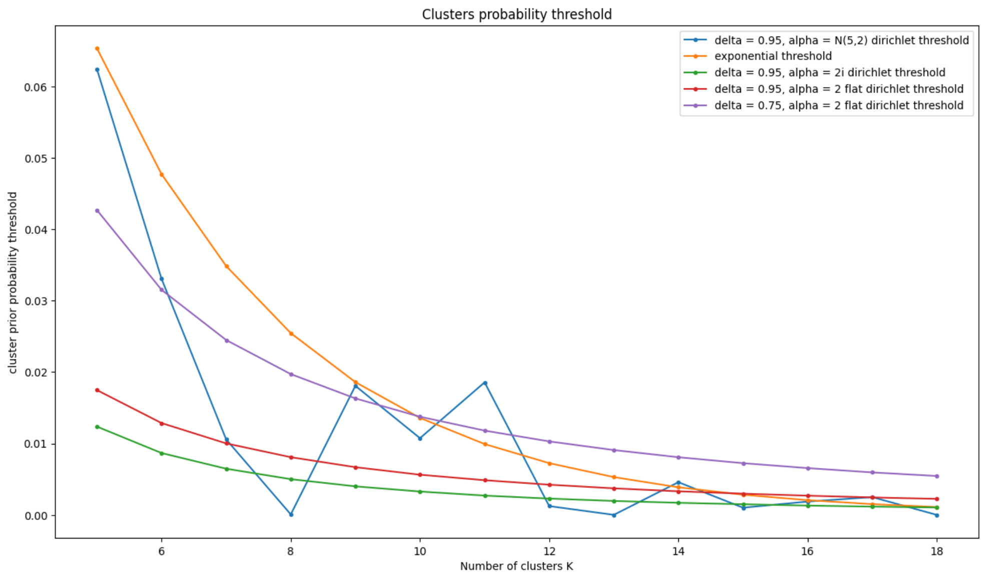


Figure A.3: Illustration of possibilities given creating probability threshold based on Dirichlet distribution with comparison to exponential threshold presented in [Sun et al., 2023, Subsection 3.1]. Thresholds are created according to Subsection 4.3.5, where $\alpha = N(5, 2)$ represents alphas taken as random samples from normal distribution with mean 5 and standard deviation 2 generated by numpy.random.normal(5, 2, K) for random seed 0. $\alpha = 2i$ represents alpha equal to $(2, 4, 8, \ldots, 2K)$.

In this section, we delve into the concept of domain generalization, its formalization, and current approaches based on risk bounds. We also present a lemma derived from this formalization. Finally, we highlight the similarities between this field and the problem addressed in this thesis.

All solutions proposed in this thesis share the goal of reducing the model's susceptibility to concept drift by identifying similarities between old and new concepts. This objective aligns with the domain generalization field, which focuses on finding similarities across training datasets to generalize to new, slightly different datasets. While domain generalization may not offer a direct solution to our problem, it provides valuable insights, particularly in formalizing aspects of our problem and offering useful theorems.

Domain generalization aims to develop models that generalize across different training datasets, referred to as domains. These domains are defined in this section. Notably, if we referred to them as concepts (see Chapter 2), the distinction would be minimal.

First, we present basic notation of the domain generalization based on risk bound from [Albuquerque et al., 2019, Subsection 2.1]. Suppose feature vectors $X \in \mathcal{X}$ and labels $y \in \mathcal{Y}$ connected by deterministic labelling function $f_{\mathcal{D}} : \mathcal{X} \to \mathcal{Y}$. Every pair (X, y) connected by labelling function is called a sample. For $\mathcal{D}$ a probabilistic distribution over $\mathcal{X}$ a pair $(\mathcal{D}, f_{\mathcal{D}})$ is called a domain. Now we define a risk in a way that for a mapping $h : \mathcal{X} \to \mathcal{Y}$ such that h is hypothesis from set of candidate hypothesis $\mathcal{H}$, a risk $R(h)$ on domain $(\mathcal{D}, f_{\mathcal{D}})$ is given by

$$R(h) = \mathbb{E}_{X\sim\mathcal{D}} l(h(X), f_{\mathcal{D}}(X)), \tag{A.7}$$

where the loss $l\ :\mathcal{Y}\times\mathcal{Y}\rightarrow\mathbb{R}_+$ quantifies accuracy of the hypothesis $h(X)$, or in the other words how far is $h(X)$ from true labelling function $f_{\mathcal{D}}(X)$. In our scenario, labelling function $f_{\mathcal{D}}$, which we are trying to approximate is labelling function of future drifted data and hypothesis h is generalizing model we learned from data we have already received.

The generalization of Formula (A.7), which gives a basic definition of the risk bound into multi domain scenario can be seen in [Li et al., 2017, Section 3.]. We assume S source domains, the domains we are provided for generalization, each of them has N_i data examples of a form $(X, y) \in \mathcal{X}\times\mathcal{Y}$. Consider the scenario where we want to reach the best performance on all the source domains. Then we would try to find parameters Θ_i for each domain S_i which provide our labelling function $h_{\Theta_i}(X)$ with the best performance on i-th source domain comparing to the true labelling function $f_{\mathcal{D}}(X)$. This can be rewritten to

$$argmin_{\Theta_1,\ldots,\Theta_S}\frac{1}{S}\sum_{i=1}^{S}\frac{1}{N_i}\sum_{j=1}^{N_i} l(h_{\Theta_i}(X_j^{(i)}), f_{\mathcal{D}}(X_j^{(i)})). \tag{A.8}$$

Problem is that Formula (A.8) does not provide us with the best possible generalization since each Θ_i can follow the specific aspects of S_i which harms the generality. On the other hand, if it would hold that $\Theta_1 = \Theta_2 = \ldots = \Theta_S$ we could say that parameters are general and not to dependent on specificity of each domain. This motivates us to rewrite the Formula (A.8) to

$$argmin_{\Theta}\frac{1}{S}\sum_{i=1}^{S}\frac{1}{N_i}\sum_{j=1}^{N_i} l(h_{\Theta}(X_j^{(i)}), f_{\mathcal{D}}(X_j^{(i)})). \tag{A.9}$$

More generalization and formalization is needed to work with this problem. Therefore, we present information based on [Albuquerque et al., 2019, Subsection 3.1 and 3.2]. The main problem is that we want to minimize the error of an unseen domain. Therefore the definition of meta-domain distribution $\mathfrak{D}$ which represents the probability distributions over countable set of possible domains. In the scenario where we are not able to say anything of the target distribution one of the best possible approaches is to minimize error generally on $\mathfrak{D}$. This provide us with the formula

$$argmin_{h\in\mathcal{H}}\mathbb{E}_{\mathcal{D}\sim\mathfrak{D}}\mathbb{E}_{X\sim\mathcal{D}} l(h(X), f_{\mathcal{D}}(X)). \tag{A.10}$$

Note the similarity of Equation (A.7) and Equation (A.10), in the latter one we add sampling domain from the meta distribution and afterwards sampling example from sampled domain.

The general setting that we have no information about the test distribution can show to be too general. According to [Albuquerque et al., 2019, Subsection 3.1] there is no free-lunch for risk bound, that means that for each fixed h it is possible to find domain $\mathcal{D}$ from $\mathfrak{D}$, where the risk is high. Therefore, it is reasonable to make some assumptions about the target distribution.

One of the most used assumption is that target domain $\mathcal{D}_T$ is in the convex hull of the source domains $\mathcal{D}_S^i$ which are provided to us for training. Under this assumptions we present [Albuquerque et al., 2019, Lemma 1.] together with detailed proof. This lemma gives the mathematical background for intuitive approach of the error minimization by minimizing the difference between each two source domains.

Lemma 12. *Given source domains $\mathcal{D}_S^i$ for $i \in \{1, \dots, S\}, 1 < S \in \mathcal{N}$ satisfying that for each $i, k \in \{1, \dots, S\}$ it holds that $d_{\mathcal{H}}(\mathcal{D}_S^i, \mathcal{D}_S^k) \leq \epsilon$, where $d_{\mathcal{H}}(\mathcal{D}_S^i, \mathcal{D}_S^k) = 2sup_{\eta \in \mathcal{H}} |Pr_{X \sim \mathcal{D}_S^i}(\eta(X) = 1) - Pr_{X \sim \mathcal{D}_S^k}(\eta(X) = 1)|$. Then the inequality*

$$d_{\mathcal{H}}(\mathcal{D}', \mathcal{D}'') \leq \epsilon$$

holds for every pair of domains such that $\mathcal{D}', \mathcal{D}'' \in \Lambda^2$, where Λ is convex hull satisfying $\Lambda = \{\mathcal{D} \mid \mathcal{D} = \sum_{i=1}^S \pi_i \mathcal{D}_S^i, \pi_1 + \dots + \pi_S = 1, \pi_i \geq 0 \, \forall i\}$.

Proof. Consider domains $\mathcal{D}', \mathcal{D}'' \in \Lambda$, which means $\mathcal{D}' = \sum_{i=1}^S \pi_i \mathcal{D}_S^i$ and $\mathcal{D}'' = \sum_{k=1}^S \pi_k \mathcal{D}_S^i$. Now we rewrite $d_{\mathcal{H}}(\mathcal{D}', \mathcal{D}'')$ as follows:

$$\begin{aligned}
d_{\mathcal{H}}(\mathcal{D}_S', \mathcal{D}_S'') &= 2sup_{\eta \in \mathcal{H}} |Pr_{X \sim \mathcal{D}_S'}(\eta(X) = 1) - Pr_{X \sim \mathcal{D}_S''}(\eta(X) = 1)| \\
&= 2sup_{\eta \in \mathcal{H}} |\mathbb{E}_{X \sim \mathcal{D}_S'}(\mathbf{I}(\eta(X))) - \mathbb{E}_{X \sim \mathcal{D}_S''}(\mathbf{I}(\eta(X)))| \\
&= 2sup_{\eta \in \mathcal{H}} \left| \int_\Omega \mathcal{D}_S'(X) \mathbf{I}(\eta(X)) dX - \int_\Omega \mathcal{D}_S''(X) \mathbf{I}(\eta(X)) dX \right| \\
&= 2sup_{\eta \in \mathcal{H}} \left| \int_\Omega \sum_{i=1}^S \pi_i \mathcal{D}_S^i(X) \mathbf{I}(\eta(X)) dX - \int_\Omega \sum_{k=1}^S \pi_k \mathcal{D}_S^i(X) \mathbf{I}(\eta(X)) dX \right|
\end{aligned}$$

At beginning we write definition then we went from probability to expected value and this expected value we rewrote according to their definition for continuous case, where Ω is support of a source domain convex hull Λ. Tn the end we use definition of belonging to Λ. Now we use that

$$1 \sum_{i=1}^S \pi_i \mathcal{D}_S^i(X) = \sum_{k=1}^S \pi_k \sum_{i=1}^S \pi_i \mathcal{D}_S^i(X) = \sum_{k=1}^S \sum_{i=1}^S \pi_k \pi_i \mathcal{D}_S^i(X)$$

since $\sum_{k=1}^S \pi_k = 1$ which holds for both i and k. Now we will continue with rewriting

$$\begin{aligned}
&= 2sup_{\eta \in \mathcal{H}} \left| \int_\Omega \sum_{k=1}^S \sum_{i=1}^S \pi_k \pi_i \mathcal{D}_S^i(X) \mathbf{I}(\eta(X)) dX - \int_\Omega \sum_{i=1}^S \sum_{k=1}^S \pi_i \pi_k \mathcal{D}_S^i(X) \mathbf{I}(\eta(X)) dX \right| \\
&= 2sup_{\eta \in \mathcal{H}} \left| \sum_{i=1}^S \sum_{k=1}^S \pi_i \pi_k \left(\int_\Omega \mathcal{D}_S^i(X) \mathbf{I}(\eta(X)) dX - \int_\Omega \mathcal{D}_S^i(X) \mathbf{I}(\eta(X)) dX \right) \right| \\
&\leq 2sup_{\eta \in \mathcal{H}} \left| \sum_{i=1}^S \sum_{k=1}^S \pi_i \pi_k \left| \int_\Omega \mathcal{D}_S^i(X) \mathbf{I}(\eta(X)) dX - \int_\Omega \mathcal{D}_S^i(X) \mathbf{I}(\eta(X)) dX \right| \right. \\
&\leq \sum_{i=1}^S \sum_{k=1}^S \pi_i \pi_k \left| 2sup_{\eta \in \mathcal{H}} \int_\Omega \mathcal{D}_S^i(X) \mathbf{I}(\eta(X)) dX - \int_\Omega \mathcal{D}_S^i(X) \mathbf{I}(\eta(X)) dX \right| \\
&\leq \sum_{i=1}^S \sum_{k=1}^S \pi_i \pi_k d_{\mathcal{H}}(\mathcal{D}_S^i, \mathcal{D}_S^k) \leq \sum_{i=1}^S \sum_{k=1}^S \pi_i \pi_k \epsilon \leq \epsilon
\end{aligned}$$

For first inequality we used the triangle inequality which can be easily translated for our case as absolute value of the sum is less then or equal to sum of the absolute values. Then we used sub additivity of the supremum and at last we use assumption $d_{\mathcal{H}}(\mathcal{D}_S^i, \mathcal{D}_S^k) \leq \epsilon$. □

The assumption that a target domain lies within the convex hull of source domains is commonly used, naturally leading to data augmentation and specific projections into latent space. Data augmentation involves generating new data by modifying existing data, often used in image processing. Specific projections into latent space, generally based on results similar to Lemma 12, typically employ neural networks and require a large amount of training data. An example of an effective projection function can be found in [Ding et al., 2022, Section 2]. While these approaches often have specific structural demands and assume a large training dataset, they provide the intuition that if the target domain is within the convex hull of the source domains, it is not significantly different from the training domains.

The formalization presented reveals strong similarities with our scenario. In our problem, we assume a sequence of concepts forming a stream, with each concept change termed as drift. Our goal is to minimize the negative impact of new, unseen concepts, closely aligning with the scenario discussed here. Both environments assume the existence of stable common information throughout the process (Definition 8). These similarities suggest that aspects of domain generalization formalization could be highly beneficial for our scenario.

However, it is important to note that despite these similarities, our scenario is not strictly domain generalization. In our case, the objective is not to develop a model that performs best in the future, as the sequence of models described in Section 2.3 allows for the use of adaptive models.

A.10 Evaluation of Hellinger detection

In this section we present graphs of epsilon evolution during the single run of Algorithm 4 on 12 Concepts dataset (Subsection 5.1.2). The graphs in this section are presented to give a better intuition behind evaluation in Section 5.4.

Figure A.4 represents part of evaluation where we study single overall feature set. We note that as stated in Section 1.3 values of epsilon in stable concept are not zero but close to constant. In Figure A.5 we present evolution of epsilons with depicted concept drifts according to 3 feature subsets based on dependencies. This evaluation is presented in Section 5.4.

A.11 TF-IDF

In this section, we briefly describe TF-IDF process, which is used for feature extraction in Section 5.6.

TF-IDF introduced in Joachims et al. [1997] is approach presented in Salton et al. [1975] and is based on calculation of words importance using term frequency (TF) and inverse document frequency (IDF). General formula for term t on set

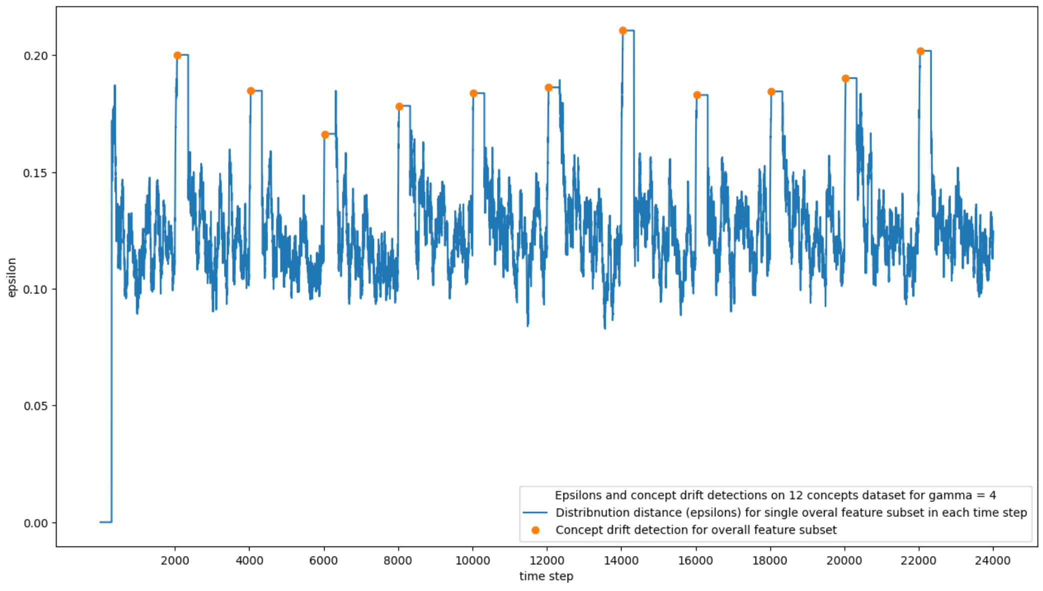


Figure A.4: Concept drift detection and epsilon evolution of Algorithm 4 in time, for single run on 12 concepts dataset. We consider one overall feature subset and $\gamma = 4$.

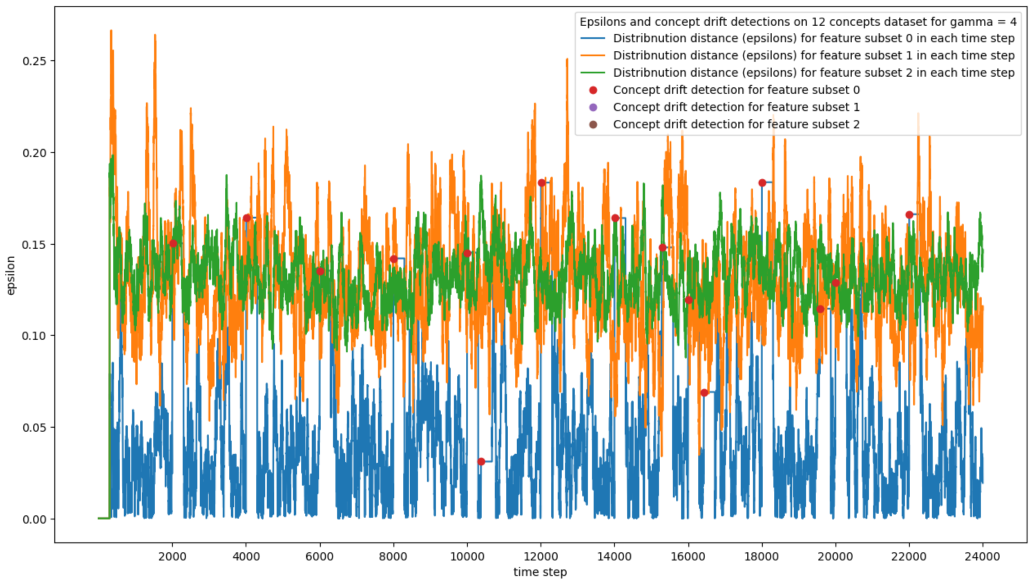


Figure A.5: Concept drift detection and epsilon evolution of Algorithm 4 in time, for single run on 12 concepts dataset. We consider three feature subsets according to the dependencies and $\gamma = 4$.

of documents D is

$$\mathrm{TF}(t, d \in D) * \mathrm{IDF}(t),$$

where $\mathrm{TF}(t, d)$ is frequency of term t in document d and

$$\mathrm{IDF}(t) = \log \left(\frac{|D|}{|d : d \in D, t \in d|} \right),$$

which in other words mean that we compute logarithm of overall number of documents divided by number of ones which contains term t. Note that it is usual (it is default in our experiments) to slightly adjust formula to

$$\mathrm{IDF}(t) = \log \left(\frac{|D| + 1}{|d : d \in D, t \in d| + 1} \right) + 1.$$

Ones in the logarithm prevent formula from division by zero and can be understood as if we added one more document consisting every term. One added to logarithm provide us with property that we consider also terms which are present in every document.

A.12 F1-score, recall and precision

In this section, we briefly describe methods for measuring the accuracy of datasets with highly unbalanced labels, following the approach outlined in [Goutte and Gaussier, 2005, Section 2]. We will apply these methods to evaluate a specific dataset in Section 5.6, using the scores introduced here.

When handling strongly imbalanced datasets, using regular percentage prediction accuracy can be misleading. For example, in a dataset with two labels where one label is 20 times more frequent than the other, a model that always predicts the more frequent label would achieve over 95% accuracy without being useful. Therefore, it is common to consider *recall* and *precision*. In our scenario, these metrics assume two labels: positive and negative. Given this context, each prediction will fall into one of the following categories. In our scenario of malware detection, a malware presents positive sample and benign software negative sample.

True positive (TP) - correctly classified positive samples.

False positive (FP) - negative samples classified as positive ones.

True negative (TN) - correctly classified negative samples.

False negative (FN) - positive samples classified as negative.

The usual percentage accuracy is defined as $\frac{TP+TN}{TP+FP+TN+FN}$. Recall, defined as $\frac{TP}{TP+FN}$, represents the number of correctly classified positive samples divided by the total number of true positive samples. This metric focuses on minimizing the misclassification of positive samples and is particularly important in contexts such as malware and fraud detection. Precision, defined as $\frac{TP}{TP+FP}$, represents the number of correctly classified positive samples divided by the total number of samples classified as positive. Precision primarily aims to minimize the incorrect classification of negative samples.

F-score connects recall and precision in a way that

$$F\beta\text{-}score = (1+\beta)\frac{recall * precision}{\beta^2 precision + recall}.$$

We use F1-score which weights both recall and precision evenly. The according formula is of the form $2\frac{recall*precision}{precision+recall}$.

A.13 GMM-based sub-procedure and label imbalance

In this section, we aim to provide a deeper understanding of the GMM-based sub-procedure on the downsampled data. We build on our previous evaluation of sub-procedure performance using the downsampled DREBIN dataset (Section 5.6).

In Figure A.6 and Figure A.7 we see the evolution of label imbalance and comparison of the DWM algorithm with and without proposed GMM-based sub-procedure. Both figures depict a scenario with the DREBIN dataset using downsampling of benign samples with probability $p_d = \frac{5}{6}$. In Figure A.8 and Figure A.9 we can see the same procedures on DREBIN dataset with downsampling of all samples with probability $p_d = \frac{5}{6}$. For better clarity, we use single results of a run of the procedures. These results correspond to the initial run used for mean computation in Section 5.6.

The results indicate a strong correlation between the F1-score performance and the number of malware samples. As shown in Figure A.7, the difference in performances remains relatively consistent, with the one exception of enhancing performance at the end of the dataset. Figure A.9 suggests that the performance deficiencies of the sub-procedure are likely due to the discarding of important information during brief periods of new malware arrivals.

A.14 Implementation and used software

In this section we briefly review the structure of supplementary code and used software.

A.14.1 Used software

In accordance with the code of ethics of Charles University article VIII[1] we state that we used AI based software Chat GPT[2] for rewriting and rephrasing sentences throughout the thesis.

We utilized several Python libraries throughout this work. However, aside from Scikit-learn (Pedregosa et al. [2011]), which was employed for the TF-IDF

[1]https://www.mff.cuni.cz/en/internal-affairs/regulations/code-of-ethics
[2]https://openai.com/chatgpt/

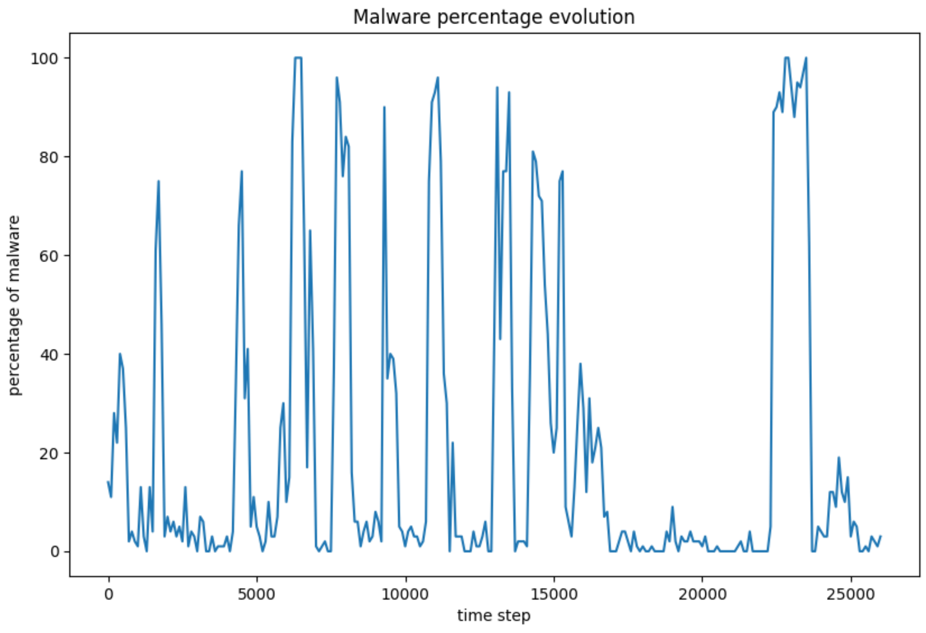


Figure A.6: The percentage of malware in the last 100 samples, it was calculated using a dataset where benign samples were downsampled with a probability of $p_d = \frac{5}{6}$. These results correspond to the initial run used for mean computation in Section 5.6.

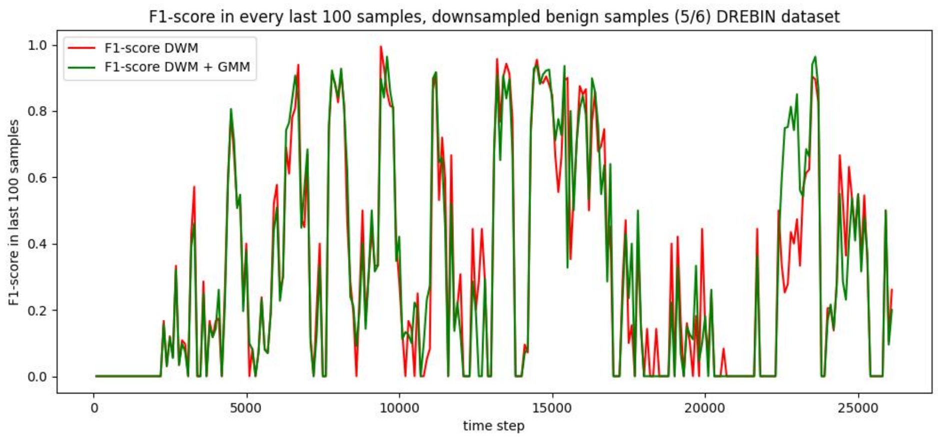


Figure A.7: Evolution of the F1-score for the DWM algorithm, both with and without the GMM-based sub-procedure, on the downsampled DREBIN dataset where benign samples were downsampled with a probability of $p_d = \frac{5}{6}$. Graph depicts F1-score in the last 100 samples. These results correspond to the initial run used for mean computation in Section 5.6.

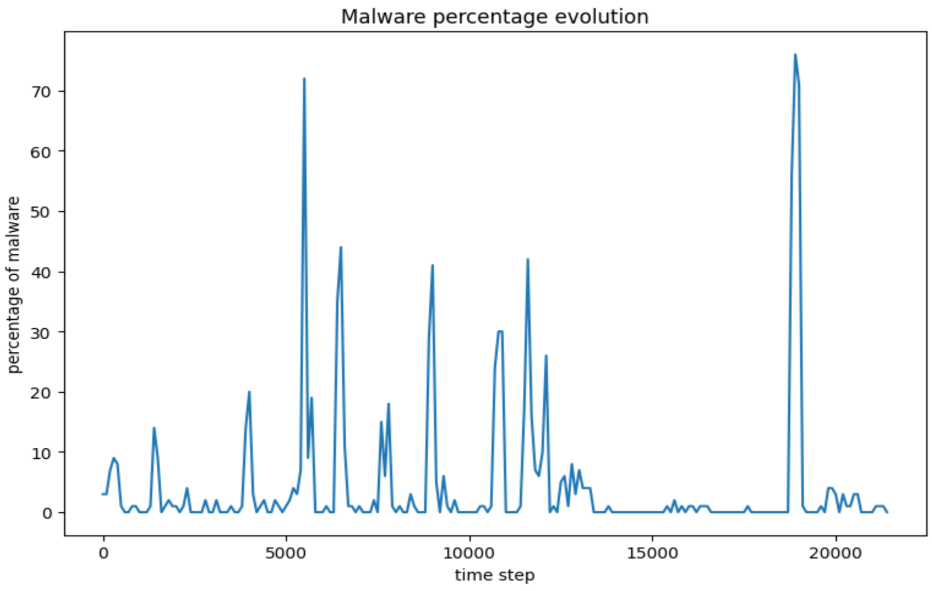


Figure A.8: The percentage of malware in the last 100 samples, it was calculated using a dataset where all samples were downsampled with a probability of $p_d = \frac{5}{6}$. These results correspond to the initial run used for mean computation in Section 5.6.

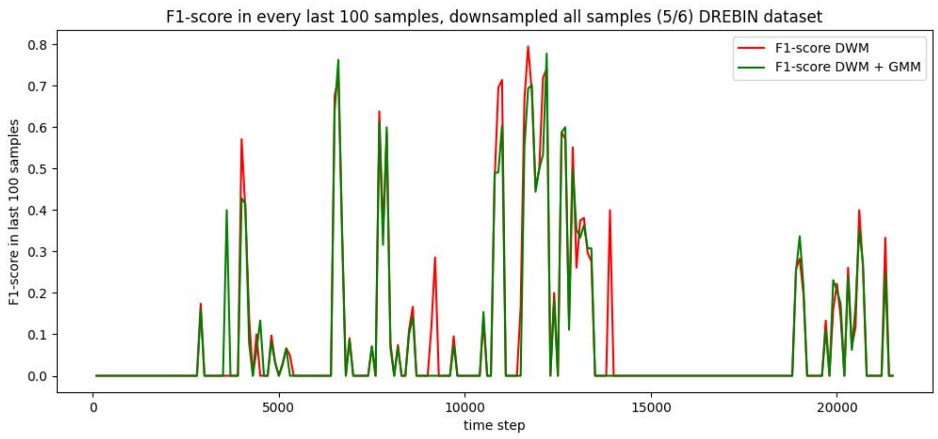


Figure A.9: Evolution of the F1-score for the DWM algorithm, both with and without the GMM-based sub-procedure, on the downsampled DREBIN dataset where all samples were downsampled with a probability of $p_d = \frac{5}{6}$. Graph depicts F1-score in the last 100 samples. These results correspond to the initial run used for mean computation in Section 5.6.

process in Section 5.6, we do not specify the others by name, as their usage was limited to addressing minor auxiliary tasks.

A.14.2 Code structure

Attached file consists of implementation and evaluation scripts, which can be also find on my GitHub[3].

Supplementary material consists of

- *README.md* - markdown file providing structure of the directory and necessary information to code and evaluation.
- classes.py - python script representing implementation of developed procedures (Algorithm 4, Algorithm 5 and Algorithm 3)
- postprocessing_functions.py - python script consisting of postprocessing functions used for plotting graphs or generating datasets.
- evaluation_source_code.ipynb - Jupyter notebook containing all evaluations of Chapter 5 except the one presented in Section 5.6.
- DREBIN directory contains two sub-directories:
  - evaluations - files needed for the evaluation of procedures presented in Section 5.6
  - results - exact outputs of the evaluations presented in Section 5.6 (also incorporated in evaluation_source_code.ipynb file)

DREBIN directory contain files, which follow that

- ..._main.py are python scripts representing the run of particular evaluation
- ..._test.py are python scripts, which run according evaluation with setting given by the abbreviation in the file name.
- ..._result.out are textual files, which were created by according ..._test.py scripts

Name of each _test.py file contains information about type of downsampling:

- ...d2... represents downsampling on all samples with probability 0.5
- ...d6... represents downsampling on all samples with probability 5/6
- ...db2... represents downsampling on benign samples with probability 0.5
- ...db6... represents downsampling on benign samples with probability 5/6

[3]https://github.com/MartinProchazka/Masther-thesis-supplementary-code

To run the evaluation it is sufficient to run the _test.py file in directory, where is according _main.py, classes.py, postprocessing_functions.py and the dataset. Example: command

Python GMMd2_test.py

runs GMM-based evaluation with 50% random downsampling (Section 5.6).

Note that for the run of the experiment it is necessary to download DREBIN dataset used in Ceschin et al. [2022], which is available online[4]. This dataset needs to be placed into sub-directory where evaluation takes place, or adjust path in ..._main.py scripts accordingly. All evaluation also uses procedures from files classes.py and postprocessing_functions.py. Both need to be in the same directory as the launching script.

Due to computational complexity all evaluations of Section 5.6 took place on metacentrum cluster. Computational resources were provided by the e-INFRA CZ project (ID:90254), supported by the Ministry of Education, Youth and Sports of the Czech Republic.

[4]https://www.kaggle.com/datasets/fabriciojoc/fast-furious-malware-data-stream