FACULTY OF INFORMATION TECHNOLOGY CTU IN PRAGUE

Assignment of master's thesis

Title:	Concept drift and model degradation in network traffic classification
Student:	Bc. Lukáš Jančička
Supervisor:	Ing. Dominik Soukup
Study program:	Informatics
Branch / specialization:	Knowledge Engineering
Department:	Department of Applied Mathematics
Validity:	until the end of summer semester 2024/2025

Instructions

Study the problematics of network traffic classification, including machine learning approaches. Focus on dataset creation and model maintenance issues, such as Concept Drift [1]. Illustrate the aforementioned issues on network traffic datasets.
Study the concept of Active Learning and the technology of Active Learning Framework (ALF) [2], designed for deploying, autonomous retraining, and evaluating machine learning models.
The thesis should analyze existing methods for concept drift detection and explore their use in network monitoring. Focus on comparing their applicability for network traffic classification.
Based on the analysis, create a software prototype that identifies shifts in the underlying distribution of network traffic classification models and guides the dataset update incorporated into ALF. Evaluate the implementation using network traffic datasets supplied by the supervisor.

References

[1] J. Lu, A. Liu, F. Dong, F. Gu, J. Gama and G. Zhang, "Learning under Concept Drift: A Review," in IEEE Transactions on Knowledge and Data Engineering, vol. 31, no. 12, pp. 2346-2363, 1 Dec. 2019, doi: 10.1109/TKDE.2018.2876857.
[2] Pešek J.; Soukup D.; Čejka T. Active Learning Framework For Long-term NetworkTraffic Classification 2023 IEEE 13th Annual Computing and Communication Workshop and Conference (CCWC)

Electronically approved by Ing. Magda Friedjungová, Ph.D. on 14 June 2023 in Prague.

FACULTY
OF INFORMATION
TECHNOLOGY
CTU IN PRAGUE

Master's thesis

Concept drift and model degradation in network traffic classification

Bc. Lukáš Jančička

Department of Applied Mathematics
Supervisor: Ing. Dominik Soukup

January 10, 2024

Acknowledgements

I would like to express my gratitude to my supervisor, Ing. Dominik Soukup, for his valuable guidance during this research. I would also like to express my appreciation to various other members of the Network Traffic Monitoring Lab for their assistance. Finally, my thanks go to my friends and family for all their support and encouragement during the creation of this thesis.

Declaration

I hereby declare that the presented thesis is my own work and that I have cited all sources of information in accordance with the Guideline for adhering to ethical principles when elaborating an academic final thesis.

I acknowledge that my thesis is subject to the rights and obligations stipulated by the Act No. 121/2000 Coll., the Copyright Act, as amended. In accordance with Article 46 (6) of the Act, I hereby grant a nonexclusive authorization (license) to utilize this thesis, including any and all computer programs incorporated therein or attached thereto and all corresponding documentation (hereinafter collectively referred to as the "Work"), to any and all persons that wish to utilize the Work. Such persons are entitled to use the Work in any way (including for-profit purposes) that does not detract from its value. This authorization is not limited in terms of time, location and quantity.

In Prague on January 10, 2024 . . . . . .

Czech Technical University in Prague
Faculty of Information Technology

Citation of this thesis

Jančička, Lukáš. *Concept drift and model degradation in network traffic classification.* Master's thesis. Czech Technical University in Prague, Faculty of Information Technology, 2024.

Abstract

Machine learning represents a highly effective and currently popular approach for network traffic classification. However, network traffic represents a challenging domain, and trained models may degrade quickly after the deployment. Other than biases present during the data capturing and model creation, concept drift represents a major source of model degradation. As the distributions evolve, the trained data patterns may stop being accurate. Because of that, the thesis focused on creating a basis for a framework for concept drift detection and analysis tailored to the domain of network traffic. The behaviour of network traffic was examined using a variety of experiments studying the development of distributions, simulating model deployment and observing the degradation over time. The presence of multiple recurring concepts was discovered with weekend traffic differing from the one of the working week. When concept drift wasn't addressed, the test F1 scores dropped from 0.92 to around 0.7 in a matter of days. Sometimes, only a few severely drifted features were the source of model degradation, so a novel approach of weighing the drift result by the feature importances was invented. The created drift detector may be enhanced by modules for additional analysis of the detected drift. A novel idea of classifying types of drift for better drift understanding is introduced. The created detector was tested to guide the model retraining and was able to not only prevent the model from degrading but also improve its performance over time.

Keywords network traffic classification, concept drift, active learning, machine learning model robustness, machine learning

Abstrakt

Strojové učení představuje vysoce efektivní a v současnosti oblíbený přístup ke klasifikaci síťového provozu. Vytvořené modely ale mohou po nasazení rychle degradovat, jelikož síťový provoz představuje náročnou doménu. Kromě zkreslení přítomných během sběru dat a vytváření modelu (tzv. bias) představuje concept drift hlavní zdroj degradace modelu. Vzory v datech objevené při trénování mohou přestat být přesné kvůli vývoji distribucí. Z tohoto důvodu se práce zaměřila na vytvoření základů frameworku pro detekci a analýzu driftu na míru pro doménu síťového provozu. Chování síťového provozu bylo zkoumáno pomocí různých experimentů studujících vývoj distribucí a simulujících nasazení modelu a zkoumajících jeho degradaci modelu v čase. Byla zjištěna přítomnost opakujících se konceptů s víkendovým provozem odlišným od provozu v pracovním týdnu. Když se drift neřešil, F1 skóre kleslo z 0,92 na přibližně 0,7 během několika dní. Jelikož byly případy kdy zdrojem degradace modelu bylo pouze několik silně driftovaných příznaků, byl vynalezen nový přístup vážení výsledků testů driftu podle důležitostí příznaků. Vytvořený detektor může být rozšířen o moduly pro dodatečnou analýzu detekovaného driftu. Je představena nová myšlenka klasifikace typů driftu pro lepší pochopení vývoje provozu. Vytvořený detektor byl testován na experimentu, kde sloužil k přetrénování modelu po detekci a byl schopen nejen zabránit degradaci modelu, ale také zlepšit jeho výkon v průběhu času.

Klíčová slova Klasifikace síťového provozu, concept drift, aktivní učení, robustnost modelů strojového učení, strojové učení

Contents

List of Figures

List of Tables

Introduction

Network traffic classification is a crucial aspect of network security and monitoring. Machine learning (ML) models have recently become widely adopted for this task due to their ability to identify complex patterns in the modelled data. However, these models are not immune to the challenges of concept drift and biases present in the dataset, which can lead to the model underperforming or degrading over time.

Concept drift refers to the phenomenon where the underlying data distribution changes over time, making the learned model less effective in making accurate predictions. Various factors, including changes in network usage patterns, the introduction of new protocols, or network infrastructure modifications, can cause this. Biases can be present because of how the dataset was captured, and the dynamic nature of network traffic patterns can be problematic for the classification. New applications emerge, existing applications change their traffic patterns, and adversaries adapt their techniques to evade detection. We aim to mitigate these effects and create a robust model whose performance doesn't deteriorate over time.

Several approaches have been proposed to address this issue, such as the Active Learning Framework (ALF) [1] that can maintain and retrain models autonomously. However, this approach could be vastly improved when extended with the ability to detect and adapt to concept drift in a timely manner and to understand how the monitored traffic develops over time. The implemented concept drift detector could thus make the model update guided and retrained precisely when needed.

Given these challenges, this thesis aims to explore the existing methods for concept drift detection and design a novel concept drift detector focused on the domain of network traffic classification, which can enhance the existing ALF. On top of that, the detector can prove to be an effective analysis tool that can help people become acquainted with an unknown dataset or assess its quality. Real long-term network traffic datasets provided by CESNET are used to develop the detector and to demonstrate the aforementioned challenges.

Structure of the Thesis

The thesis is structured into four chapters, with the first one focusing on the theoretical background and research, which can be divided into several topics. The first one provides an overview of network traffic classification and showcases the various techniques currently used, with an emphasis on ML-based classification of network flows. The following sections revise the corresponding machine learning paradigms with the main focus put on describing how concept drift or biases can impair the accuracy of machine learning models. Concept drift, the main focus of this thesis, is properly defined and examined, and the approaches to its detection are outlined. The idea of active learning is then introduced. The workings of the existing Active Learning Framework are explored.

The second chapter presents a practical analysis of real-world network traffic datasets, with many being captured at CESNET's network. The emphasis is put on exploring the behaviour of network traffic and providing evidence of concept drift in these datasets, showcasing how it can lead to the model degrading over time. Existing solutions for concept drift detections are examined and analysed for use in the development of the novel drift detector.

The core contribution of this thesis, the concept drift detector itself, is described in the third chapter. The chapter serves to describe the design choices made, analysis modules of the detector, its usage and the various implemented statistical tests. The ways the model can enhance ALF or become a useful analysis tool are discussed.

All the findings on how to create robust network traffic classification models with the help of understanding how the properties of the modelled traffic change over time are discussed in the last chapter. The created drift detector is properly examined using a simulation of long-term model deployment with updates based on drift detections. Various possible improvements and discoveries leading to possible future research are discussed.

Chapter 1

Theoretical background

The research into the theory corresponding to the goals of the work is presented in this chapter. It is divided into multiple parts, with the first one presenting the current approaches and challenges in the area of network traffic classification. The theoretical background of the main topic of concept drift is then established, together with other causes of machine learning (ML) model degradation, such as biases. The paradigm of active learning is then explored, together with the framework utilising those paradigms in the area of network traffic classification called Active Learning Framework (ALF).

1.1 Network traffic classification

Network traffic classification presents an important part of network traffic monitoring and analysis. It can be a crucial step in the network administrators' work for managing network resources, ensuring the quality of service or discovering malicious behaviour. It is important not only for internet service providers but also for corporations and countries, as it can be used to enforce policies, block the usage of specific applications, or fulfil lawful interception regulations.

The core idea of network traffic classification is identifying the traffic and classifying it by the applications, protocols, or services used. The granularity of the classes depends on the specific use case. It may be general internet traffic types such as web browsing, streaming video, file transfers, or individual applications, such as YouTube, Netflix, or Spotify. As the field is highly evolving and the approaches depend on the specific use case, there are many ways to classify network traffic. The following sections further describe the various approaches, ranging from the simple ones historically used to the current state-of-the-art techniques using machine learning.

1.1.1 Packet-based analysis

One of the approaches to network traffic classification is examining the content of data packets. Visualised in Figure 1.1 is the structure of the IPv4 packet. The simplest but the least reliable method is using the information present in the packet header, specifically the port numbers, as they are assigned to the common services by the global standards organisation IANA (Internet Assigned Numbers Authority). For example, HTTP traffic is assigned to port 80, with HTTPS being assigned to port 443. One may classify email traffic by SMTP servers commonly using TCP on port 25 or 857. However, there are many reasons why this approach isn't reliable. Not only is the granularity of the classes and the accuracy of the whole method low but many applications are deployed randomly or may use masquerading (allocation of common ports to deliver other traffic than expected on that specific port), making this method infeasible in those cases. The only advantages of this approach would be the simplicity and low computational cost [2].

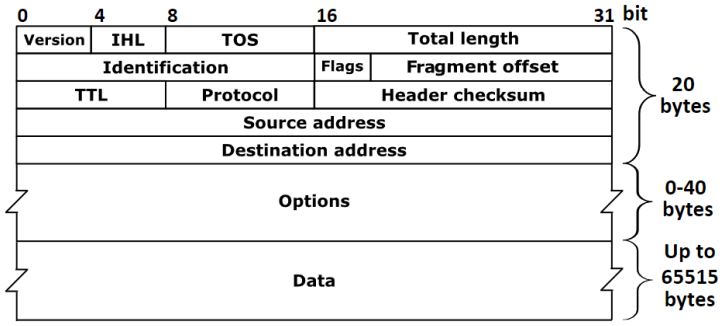

Figure 1.1: Structure of the IPv4 packet. Source: [3]

Deep packet inspection (DPI) represents a more powerful approach that combats the issues of the port-based methods by also utilising the payloads in their classification process. DPI presents a highly effective and widely used tool as it extracts a larger range of metadata, which is then used to match against known patterns. Today, the trend is to use encrypted traffic, with the usage of HTTPS overcoming that of HTTP in recent years. The latest report from Internet Security Research Group [4] notes that HTTPS made up 84% of page loads on average in 2023. Because encryption provides a challenge to the traditional DPI methods, more complex methods using decryption have to be used.

This raises a major ethical question about the use of DPI and its possible breach of the privacy of the users. There are concerns about how it may be misused by Internet service providers or governments and used for surveillance and censorship, especially in countries with less democratic regimes. The ethical sides of using DPI are a complex topic warranting its own interdisciplinary research, such as [5].

Another downside of DPI methods is their computational insensitivity. In some cases, they may negatively impact the performance of the network or may not be feasible for real-time network monitoring. However, the methods are usually highly accurate and offer fine granularity as patterns of many popular services are known. There exist not only paid corporate solutions but also those publicly available under open source licenses, such as the nDPI library [6].

1.1.2 Flow-based analysis

There are many advantages of working with a different structure than the packets themselves. Said structure should aggregate the raw traffic data from a single data transfer and provide information about it. Methods working with it wouldn't analyse the payloads themselves and thus wouldn't present a possible breach of privacy. The idea of aggregating packets and working with traffic statistics would also require vastly less computational resources. The mentioned structure is called the network flow.

The terminology concerning network flows (also called IP flows) was laid by the Internet Engineering Task Force (IETF) working group in their publication RFC 3917 [7] and further developed in RFC 5103 [8] and RFC 7011 [9]. According to RFC 3917 [7], "*A flow is defined as a set of IP packets passing an observation point in the network during a certain time interval. All packets belonging to a particular flow have a set of common properties.*" Source and destination IP address, source and destination port, and the used protocol are the frequently chosen set of common properties from the definition, also called a flow key. Packets matching the same flow key get aggregated into the network flow until one of the following reasons for flow termination occurs. The connection may terminate naturally, the longest allowed time between two packets may be exceeded, or the flow duration may reach the maximal allowed duration of a single flow.

The first network flow standard called *NetFlow* was designed by the Cisco company. After that, IETF published their definition of *IPFIX* standard, which we used as a basis for a general definition of network flow. The definition implies that the network flows aggregate packets heading in the same direction, this approach later being defined as yielding unidirectional flows (*uniflows*). As it is useful to monitor the whole communication, not only the one from source to destination, bidirectional flows (*biflows*) were introduced. Biflows merge two corresponding uniflows together and then calculate flow statistics in each direction.

Tools called flow meters (or flow exporters) are used to accumulate packets into flows. Examples of flow meters would be Joy [10] from Cisco, the original creator of NetFlow, Flowmon Probe [11], CICFlowMeter [12], or ipfixprobe [13]. Ipfixprobe flow exporter is a part of the NEMEA open-source network traffic analysis framework developed by CESNET. It is designed to create

biflows and can both work on live networks or process captured PCAP files. Ipfixprobe was used to export flows that created most of the datasets used in this thesis. The aggregated flow metadata exported by ipfixprobe in its default configuration can be seen in Table 1.1. On top of that, ipfixprobe allows the use of various plugins to export additional information. For example, PSTATS plugin is used to export the metadata such as payload lengths, timestamps and TCP flags of the first n packets.

Table 1.1: Fields exported by the ipfixprobe flow exporter in its default configuration without additional plugins [13]

Field	**Type**	**Description**
DST_MAC	macaddr	destination MAC address
SRC_MAC	macaddr	source MAC address
DST_IP	ipaddr	destination IP address
SRC_IP	ipaddr	source IP address
BYTES	uint64	number of bytes (src to dst)
BYTES_REV	uint64	number of bytes (dst to src)
LINK_BIT_FIELD	uint64	exporter identification
TIME_FIRST	time	first time stamp
TIME_LAST	time	last time stamp
PACKETS	uint32	number of packets (src to dst)
PACKETS_REV	uint32	number of packets (dst to src)
DST_PORT	uint16	transport layer destination port
SRC_PORT	uint16	transport layer source port
DIR_BIT_FIELD	uint8	determines outgoing/incoming traffic
PROTOCOL	uint8	transport protocol
TCP_FLAGS	uint8	TCP protocol flags (src to dst)
TCP_FLAGS_REV	uint8	TCP protocol flags (dst to src)

1.1.3 Use of machine learning

The exported metadata itself can be useful for network traffic monitoring and indicative of incidents happening on the network. However, for the task of network traffic classification, additional steps have to be taken. Machine learning models provide powerful tools that can classify the captured flows. However, the metadata usually exported by the flow meters isn't in the final form suitable for training the models on. Because of that, feature engineering represents one of the most crucial steps in the process of creating ML network classifiers. Its goal is to extract statistical features from flow data, which are well-suited to distinguishing the various services or protocols. There are many approaches to selecting the features, ranging from the global statistics of the whole flow to monitoring the behaviour of the first few packets [2].

How to choose the best-performing feature vector represents a complex topic. As an example of commonly used features for the classification of network flows, look at the features used in the CESNET DataZoo [14] project, listed in table 1.2. DataZoo provides access to various large-scale, long-term datasets with the flows being preprocessed into features suitable for training ML models. Because of that, CESNET's datasets are the primary source of data for analysis and testing of the created concept drift detector.

Table 1.2: Examples of some of the features used in datasets available with DataZoo [15]; *The transmission direction from the client to server is shortened as forward, reverse denotes the direction from server to client. A per-packet information (PPI) sequence describes the first 30 packets of a flow.*

Feature	**Description**
DURATION	Duration of the flow in seconds
BYTES	Number of transmitted bytes – forward
BYTES_REV	Number of transmitted bytes – reverse
PACKETS	Number of transmitted packets – forward
PACKETS_REV	Number of transmitted packets – reverse
PPI_LEN	Number of packets in the PPI sequence
PPI_DURATION	Duration of the PPI sequence in seconds
PPI_ROUNDTRIPS	Number of roundtrips in the PPI sequence
FLOW_ENDREASON	The reason of flow termination
PHIST_SRC_SIZES	Histogram of packet sizes – forward
PHIST_DST_SIZES	Histogram of packet sizes – reverse
PHIST_SRC_IPT	Histogram of inter-packet times – forward
PHIST_DST_IPT	Histogram of inter-packet times – reverse

The paradigm of supervised learning is widely used for network traffic classification as it is better suited than unsupervised. However, this presents a challenge to the creation of said models – the need for quality dataset labelling. The training procedure of providing the sets of input variables together with the corresponding target variable requires the ability to annotate the modelled services or protocols when capturing the flows for the training data. Another challenge is the uncertain generalisation of the knowledge gained when the capturing process was done on a specific network in a specific state at one point in time. This can lead to incorrect patterns in data, called biases. On top of that, network traffic represents a highly evolving domain where the behaviour of classes evolves, and the models can become degraded over time. This effect is called concept drift. Both of these challenges are further described in detail in the following sections. Nevertheless, even if those challenges should be addressed, it still provides high accuracy and reasonable computational cost and adheres more to privacy and ethical norms [2].

1.2 Overview of related machine learning concepts

In recent years, machine learning techniques have proved to be more than capable in a large number of fields and thus have become widely used, not only in the examined domain of network traffic classification, such as [16]. However, several challenges still exist that can lead to the degradation of machine learning models. This can be due to biases and concept drift, which should be addressed when creating robust, long-lasting models.

A basic understanding of the core machine learning concepts is expected of the reader. This section focuses on how biases present in the dataset can yield a model with unsatisfactory performance and how concept drift can make the model worsen in time. Before exploring these ideas, a short overview of key machine learning concepts is presented.

Machine learning projects usually come through complex life cycles before resulting in fully deployed and maintained models. Several analytic models have been proposed to describe the life cycle of the project, with CRISP-DM (Cross-industry standard process for data mining) [17], visualised in Figure 1.2, being generally used as an example. It is first necessary to understand the target domain and the modelled data. Before creating the models, data has to be collected and processed in a way suitable for training machine learning models. This means selecting or designing an appropriate feature vector. Its creation in the case of network traffic classification was discussed in the previous section.

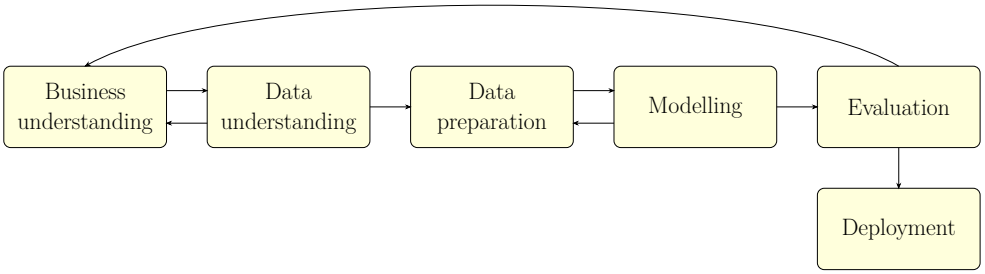

Figure 1.2: Broad overview of the life cycle of machine learning project based on the CRISP-DM model

If the process isn't thorough enough, incorrect assumptions or patterns in data, so-called biases, further described in the following section, may exist, which lead to the models underperforming. After a model is trained, evaluated and deployed, another source of model degradation presents itself – concept drift. Because of that, models have to be appropriately maintained to prevent their performance from deteriorating, especially in online learning tasks in highly evolving domains, such as network traffic classification.

In the case of a supervised learning task, the goal is to find a function that satisfies the best $y \approx f(X)$ where X denotes the set of input variables, y is the

target variable, and f represents the trained model. Given a feature vector X, the model provides predictions of $\hat{y} \equiv \hat{y}(X)$. Statistically speaking, the underlying distribution of the modelled domain is unknown and is modelled using empirical observations – the pairs of target variable samples with the corresponding feature vector. We aim to create a model of the underlying probability distribution that generated the training data, as this model should be able to generalise to new, previously unseen data [18].

A model's performance degrades when the training data is no longer representative of the current distribution or when there exist additional patterns in the training data. The issue can be with the model overfitting to those incorrect patterns in the training data when they aren't properly addressed. Several metrics are used to enumerate the model's performance. In the case of classification, accuracy, the estimate of the probability $P(\hat{y} = y)$, is often used. However, it is not a suitable metric in the case of imbalanced datasets. In general, F_1 score is a more suitable metric, defined as $F_1 = \frac{2*TP}{2*TP+FP+FN}$, where TP denotes the count of true positive samples, FP false positive and FN false negative [19].

1.3 Biases

When creating robust models, the presence of biases in data should be analysed and addressed. Bias refers to the systematic errors in the model that affect its predictions. There exists a plethora of bias sources, ranging from technical aspects of the models to the societal structure and human behaviour. It may arise from preferences or assumptions of the used model infrastructure or from the way data was collected or processed [20].

Another aspect of bias is from the point of view of artificial intelligence (AI) fairness, which is becoming a more and more pressing issue every day as AI models get used in high-stakes scenarios, such as hiring and giving loans. In this context, the definition of bias shifts from having systematic errors in prediction to providing decisions which are *unfair*. Mehrabi et al. [21] define *Fairness* as the "*absence of any prejudice or favouritism toward an individual or group based on their inherent or acquired characteristics.*" However, at that point, the discussion becomes interdisciplinary and without proper consensus in the scientific community because this definition is viewed from the point of ethics and not statistics. This thesis will discuss the former point of view on bias and present the perspective of addressing bias in the domain of network traffic classification.

1.3.1 Examples of bias

Many types of bias have been defined, some described by Hellström et al. [20] Following examples may occur in the domain of network traffic classification:

Sampling bias (sometimes called *selection bias*) occurs when the training data sample is chosen in a way that is not representative of the real-world distribution. Specific classes may be under or over-represented, or only a subset of some class is present in the training sample. This differs from a class imbalance in a way that not only classes may be over-represented but also trends in data, which aren't general and present in the real-world distribution.

For example, in the area of image classification, if all the objects in the training images are placed in the centre, the classifier may underperform when they are placed in the corners. In the studied domain of network traffic, sampling bias can occur when selecting only one or a small number of services to represent the labelled class, for example, sampling only Spotify traffic for the class of "*Audio streaming*". Another issue could arise with how the network traffic patterns evolve over time, sometimes called *temporal bias*. For example, the model could be biased toward night traffic, or if the different classes were sampled at different times, the model would be biased towards deciding by the time patterns instead of the differences between the classes. An example of incorrect sampling resulting in wrong models can be seen in Figure 1.3.

Measurement bias arises when there exist inconsistencies or inaccuracies in the data capturing. Apart from unreliable or noisy measurements, this issue can present itself when the inference procedure differs from the training one. For example, a model trained on network traffic collected on a backbone network may underperform when deployed on a local network with low throughput.

Label bias presents another source of model underperformance, brought in by the inconsistencies in the labelling process. There may be human error in the labelling or a systematic one. The models may learn the training sample perfectly, but these errors may bring in patterns not present in the underlying distribution.

Algorithmic bias may be introduced when choosing an algorithm which isn't well-suited for the modelled distribution or domain. The different algorithms make different assumptions or may be more sensitive to some distributions. For example, linear regression makes the assumption that the modelled distribution is linear, homoscedastic (having the same variance), and the errors are independent. The illustrations of examples of algorithmic bias can be seen in Figure 1.4.

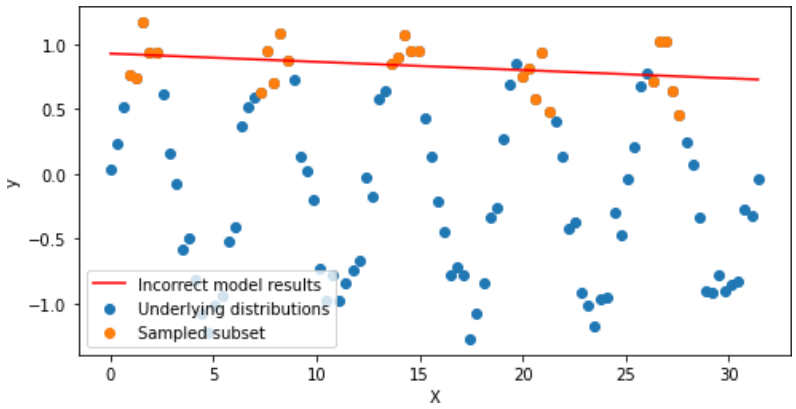

Figure 1.3: Visualisation of sampling bias: Presume we have a simple dataset of a single feature x and the distribution of the target variable y has periodic behaviour. If we sampled the distribution incorrectly, instead of discovering that the underlying distribution follows a sine function, we would observe a linear relationship.

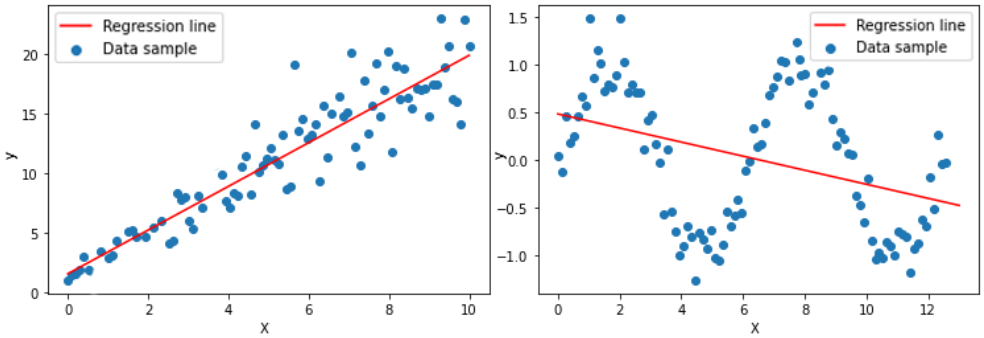

Figure 1.4: Visualisation of algorithmic bias sources: Presume we have a simple dataset of a single feature x and observed these distributions of target variable y. Linear regression wouldn't be well-suited in the case of heteroscedasticity (left) or higher-order distributions (right).

One should be aware of such biases when designing machine learning models. The models may underperform when deployed if not created robustly or have a negative impact on society, considering the second meaning of the word bias. The goal is always to create models that generalise the modelled domain well and continue performing well when deployed in real life. Having robust procedures where the conditions are the same in model creation and deployment and potential sources of bias are addressed usually yields a better model than simply looking for the best validation score.

1.4 Concept drift

Concept drift presents a challenge causing model degradation over time, especially in dynamic domains such as network traffic. Concept drift can be defined as a change in the statistical properties of the target variable that the model predicts, properly defined in the following section. In the studied domain of network traffic classification, this can be caused by changes in user behaviour or network protocols, the emergence of new applications, etc. As mentioned earlier, when developing machine learning models, we should aim to create a model that properly generalises the target domain and behaves well when presented with new and unseen data. However, concept drift can pose a challenge if not dealt with. Even if we created a model that perfectly reflected the state of the network at a certain point in time, the models might underperform when deployed in a different time period.

1.4.1 Problem definiton

The phenomenon of concept drift was first proposed by Schlimmer et al. [22], where they called for a need for robust algorithms to react to concept change over time. This effect of the statistical properties of the target variable changing over time is commonly [23] [24] [25] [26] defined as follows:

Concept drift occurs between time steps t_0 and t_1 if

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

where $P_t(X, y)$ denotes the joint distribution between the set of input variables X and the target variable y at time step t. The concept itself is defined as the joint distribution $P_t(X, y)$ or $P_t(X)$ in the case of unsupervised learning where there is no target variable.

The joint distribution $P_t(X, y)$ defining concept can be rewritten as follows:

$$P_t(X, y) = P_t(X) \times P_t(y|X)$$

where $P_t(X)$ denotes the distribution of the features and $P_t(y|X)$ the posterior probability of the classes. When observing the two parts the joint distribution was decomposed into, three possible cases can describe how the distribution shifted, as outlined by Lu et al. [25] :

1. $P_{t_0}(X) \neq P_{t_1}(X) \wedge P_{t_0}(y|X) = P_{t_1}(y|X)$

   In this case, there is a shift in the distribution of the features, but no shift in $P_t(y|X)$ is present. This means that the decision boundary of the model remains unchanged, and thus, the performance doesn't degrade. Because of that, the term *virtual drift* was coined [23] for this phenomenon.

2. $P_{t_0}(X) = P_{t_1}(X) \wedge P_{t_0}(y|X) \neq P_{t_1}(y|X)$

   This is the case where concept drift leads to model degradation as the decision boundary changes. In other words, given the same feature distribution, the probability of observing the classes differs, and the predictions, once correct, may not be valid anymore.

3. $P_{t_0}(X) \neq P_{t_1}(X) \wedge P_{t_0}(y|X) \neq P_{t_1}(y|X)$

   It is often the case that both of the previous drift sources occur.

Opposed to the *virtual drift*, the last two cases where the $P_t(y|X)$ changes are sometimes referred to [27] as *actual drift* or *real drift*, as these cases result in model degradation as the decision boundary changes. Figure 1.5 demonstrates the various cases and their effect on the model performance (visualised by the change of decision boundary).

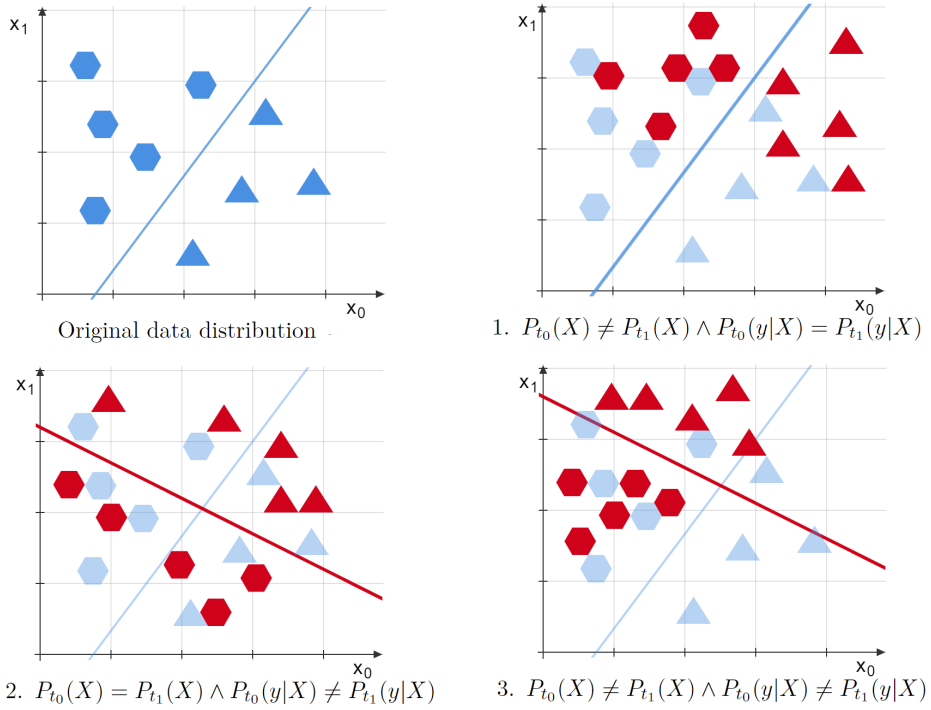

Figure 1.5: Visualisation of concept drift sources; Presume we have a feature space $X = \{x_0, x_1\}$ and two classes $y = \{y_0, y_1\}$ visualised as triangles and hexagons. The data distribution before shift (X_{t_0}) is visualised in blue, while the shifted distribution (X_{t_1}) is visualised in red.

Because of these three cases describing the source of concept drift, which is defined as a shift of $P_t(X, y)$, several other terms have been used in the literature [28] [29]. The terms *data drift* or *dataset shift* are sometimes used, which encompass various drift types, such as *covariate shift*, *prior probability*

shift and *concept drift*, which is only used to describe the second case of only the $P_t(y|X)$ changing. However, for better clarity and consistency with the commonly used terminology today [23] [24] [25] [26], we will regard concept drift as we defined earlier as the change of the joint distribution $P_t(X, y)$.

It can be noted how the previously studied construct of biases can be considered interconnected with concept drift. For example, selection bias, the phenomenon where the dataset is created in a way that is not reflective of real-world distribution, can be one of the sources of concept drift as this can lead to the change of $P_t(X)$ during deployment.

1.4.2 Categorising concept drift

To further understand how the concept develops and how to react to the changes, several categories of the concept drift patterns have been introduced, as reported by Gama et al. [24] and Lu et al. [25]. The different forms of concept change are visualised on simplified one-dimensional data distributions and can be seen in Figure 1.6 and are further discussed.

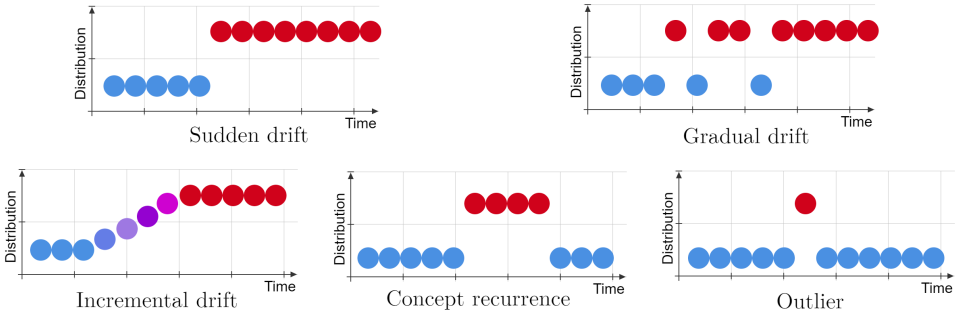

Figure 1.6: Visualisation of concept drift categories on simplified distributions.

The first type of drift, called *sudden* or *abrupt*, happens when there's an immediate switch from one concept to another. For example, in our studied domain of network traffic, this can occur with a change of certificates, which can make the corresponding features irrelevant.

Concept drift can also occur *gradually*, where the behaviour of the users of the network evolves and changes over time, for example. Similar to *gradual* drift is the behaviour of the *incremental* drift. In this case, the idea of an *intermediate concept* was introduced. Concept drift may last for a longer period of time, so this term describes the mixture of starting and ending concepts.

Concept recurrence describes the situation where the ending concept isn't a completely new one but was previously seen. Some of the causes of the drift can be temporary, and the previous concept can return. There is a possibility of an anomaly, so-called *outlier*, which doesn't present a new concept we should adapt to but a once-off random deviation instead.

The idea of recurring concepts presents novel approaches to concept drift adaptation. The approach of adapting to the first three drift types is minimising the model degradation by the faster re-learning of the new concept. In the case of recurring concepts, it is to find the closest historical concepts quickly. However, the behaviour of concept drift is often complex and presents a mixture of those drift categories.

1.4.3 Concept drift detection

The overall process of model maintenance when learning under concept drift can be seen in Figure 1.7. There are many aspects of working with data with concept drift present. The first is the *concept drift detection* itself, where a decision is made whether the drift is present in the current batch of incoming data. In that case, the area of *concept drift understanding* focuses on providing other information about when and where it happened, how severe it is, etc. This leads to the field of *concept drift adaptation*, which studies how to guide the retraining of the models effectively.

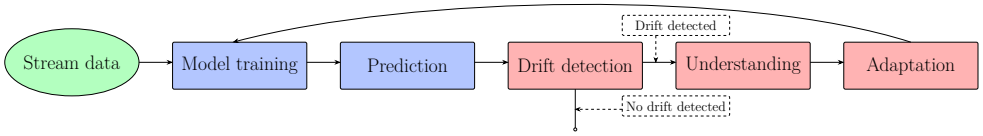

Figure 1.7: Overview of the framework of learning under concept drift

This section focuses on methods of detecting whether drift is present in the incoming data or not. Lu et al. [25] present a generalised overview of the pipeline of drift detection and how it can be divided into the following stages:

Data retrieval aims to retrieve relevant chunks from the data stream. It is accomplished by retrieving a pair of time windows, one representing the historical data and the second one representing the current data.

Data modelling represents an optional step of further pre-processing of the retrieved samples. For example, data can be further sampled, key features chosen, or a dimensionality reduction algorithm such as PCA can be used. This can be done to meet storage and speed requirements for tasks where we require quick inference, which is often the case in an online learning environment.

Test statistics calculation is the formation of test statistics for the hypothesis test and the corresponding dissimilarity or distance between the retrieved data samples. This presents the greatest challenge of drift detection: how to design a test or dissimilarity metric that is accurate and robust.

Hypothesis test represents the final part of the pipeline, where the statistical significance of the change observed in the previous stage is evaluated. The tests are usually designed to decide whether the two samples come from the same distribution, with test rejection leading to drift detection. The p-value of the test (the probability of observing the same or more extreme test results if the null hypothesis were true) is sometimes used to decide if drift is present with statistical significance. Other methods, such as using Hoeffding's inequality, permutation tests or estimating the distribution of the test statistics by maximising the likelihood, can also be used.

These four stages work together to make a statistically significant decision of whether the drift is present at that specific point in time. In that case, an alarm signal is sent, and the model is retrained to adapt to the new concept. There are many variations of this generalised pipeline, which differ by the tests they use, their sampling approaches, etc. These variations can be divided into the following categories, as described by Lu et al. [25]:

Error rate-based drift detection works by tracking the changes in the model error rate. Drift Detection Method (DDM) [30], a representative of this approach, serves as one of the first commonly used algorithms for drift detection. DDM works by comparing the online error rates of two learners. The first is the historical one, and the second one is created by adding the new data to the training dataset. A statistically significant increase in the error rate of the new model, which was trained on both the historical and the current data, leads to the detection of concept drift. Many variations of this method exist, differing mainly in how the hypothesis test is designed.

Data distribution-based drift detection models work by addressing the root cause of the concept drift itself instead of monitoring the model degradation. The distributions of the historical sample and the current one are compared using a test or distance metric to prove their dissimilarity. While these models usually incur higher computational costs than the error rate-based approaches, their benefits are a better understanding of where the drift is happening and the usefulness of the dissimilarity metric to act as a measure of the severity of the drift. On top of that, their results are interpretable as they directly describe how the data samples are different. For example, Reis et al. [31]. use the two-sample Kolmogorov–Smirnov test as the core of their drift detection algorithms. The null hypothesis presumes that the two samples were drawn from the same underlying distribution. Rejection of the null hypothesis leads to the detection of concept drift.

Multiple hypothesis test approaches are similar to the idea of ensemble learning. Multiple tests are carried out, and the final decision is a mixture of those weak decisions. The test can be either run in parallel or hierarchically. In the latter case, some tests can have lower precision, but also lower computational cost and their drift detection leads to a validation by a test which is higher quality but slower.

1.4.4 Understanding concept drift

For the most suitable reaction to the occurring concept drift, it is optimal to have a thorough understanding of how it develops. Detection frameworks can present information about when the drift took effect, how severe it is, and where the drift regions are. Lu et al. [25] discuss and categorise the following aims and methods of concept drift understanding:

Time of detection: Concept drift is defined as $\exists X : P_{t_0}(X, y) \neq P_{t_1}(X, y)$, occurring between the time steps t_0 and t_1. The most trivial ability of the concept drift detector is to identify the timestamp of drift appearing. The aforementioned approaches to drift detection are used to decide whether drift occurred on the current timestamp based on the historical and current windows. In that case, an alarm signal is sent, which can be used to force the model to update itself and react to the new concept. This presents two challenges: minimising both the false detection rate and the speed of drift discovery.

The drift alarm should have a statistical guarantee with a predefined false alarm rate. The aforementioned Kolmogorov–Smirnov test, with the null hypothesis being that the two samples come from the same distribution, can be used as an example. The false alarm rate is derived from the significance level of the test α, the probability of rejecting the null hypothesis when it is true. Drift detection, in this case, means the p-value of the test being lower than α and knowing that the probability of the historical and current samples coming from the same distribution is significantly low.

The second issue is that there exists a delay between the actual drifting timestamp and the timestamp of the alert, as the detectors often require some amount of new data to process. One of the approaches can be specifying two thresholds, one for the detection itself and a second for a preemptive warning. For example the warning level of $2\sigma = 95\%$ and alarm level of $3\sigma = 99\%$ is sometimes used, with the corresponding significances $\alpha_{warn} = 0.05$ and $\alpha_{alarm} = 0.01$.

Drift severity: On top of detecting the drift, some algorithms can provide additional information about the severity of the occurring drift. Severity can be a metric that quantifies the dissimilarity between two concepts.

Data distribution-based approaches are generally better suited to this discipline than error rate-based ones as the core of their tests is to find the dissimilarity of the distributions compared to monitoring the drop in performance of the learning system. The difference in performance on historical and current samples can be used to estimate the severity of the drift. These estimates aren't well explainable, though.

Distribution-based approaches often work with test metrics that can be reasonably explainable and suitable for determining the severity of the drift. In the case of the previously mentioned Kolmogorov–Smirnov test, the p-values of the test can serve as drift severity. The stronger the rejection of the test, the more severe the drift is. Following are examples of other metrics of similarity between distributions that are well suited for drift detection and severity measurement.

Kullback–Leibler divergence, also called relative entropy, is a popular measure of how two probability distributions, P and Q differ from each other. It is defined as $D_{KL}(P||Q) = \sum_x P(x) \log(\frac{P(x)}{Q(x)})$. However, it cannot be called a metric as it doesn't satisfy some of the axioms of metric space. It isn't symmetric, and it doesn't hold the triangle inequality. To improve on this measure, Jensen–Shannon divergence was introduced. Its square root is often called the Jensen-Shannon distance and is a metric ranging from zero to one, zero meaning identical distributions and one completely different. This makes it fairly explainable and a good fit for being a metric of drift severity [32].

Another metric used for detecting concept drift and judging its severity is called the Wasserstein metric, sometimes known as the earth mover's distance. To put it simply, it represents the minimal cost of transforming one distribution to another, as in moving piles of dirt over each other. Wasserstein distance can be normalised, resulting in a relative distance better suited for drift detection. After normalisation, it is fairly explainable but isn't well-defined for categorical data and can be relatively computationally expensive [27] [33].

Drift location: There are drift detection algorithms which can also provide where are the regions of concept drift located. In some systems, this information can be used for better concept drift adaptation. As some data regions can remain stable as others drift, the old model can be used for inference in the stable regions before the new model is retrained.

Generally, these methods work by partitioning the feature space by a decision tree or more complex tree structures. The logic of drift detection is contained in the leaf nodes, each corresponding to a hyper-rectangle in the data feature space. Because of that, the detection results are associated with the corresponding regions.

1.5 Active learning

When training machine learning models under the paradigm of supervised learning, large amounts of quality labelled data are needed for creating well-performing models. Another common challenge is keeping the model performing as the data distribution and patterns change over time. Both of these challenges can be addressed by the field of machine learning called active learning, where the models are created in a way that minimises the number of labelled samples needed and allows the model to be continually retrained. The following sections describe the main principles of active learning and how they are utilised in the current Active learning framework, which the drift detector aims to extend.

1.5.1 Principles of active learning

As the survey paper by Settles [34] describes, the core idea of active learning is minimising the amount of labelling that needs to be done by allowing the learning algorithm to train only on an intelligently chosen subset of data. The annotation process can be computationally expensive, time-consuming or require an annotator who is an expert in the field. For example, as described by El-Hasnony et al. [35], active learning can have a huge impact in the field of medicine, where the amount of human experts is limited and the annotations complex and costly. In the studied field of network traffic, detectors can be deployed on networks with hundreds of thousands of network flows generated every second, making the online annotation of each flow infeasible.

Two main approaches to active learning (AL) exist based on how the data is processed. *Pool-based* approaches work with processing a large pool of offline data, while *stream-based* approaches are designed for processing an online stream of data. The general overview of the process of active learning can be seen in the example of pool-based AL overview, visualised in Figure 1.8.

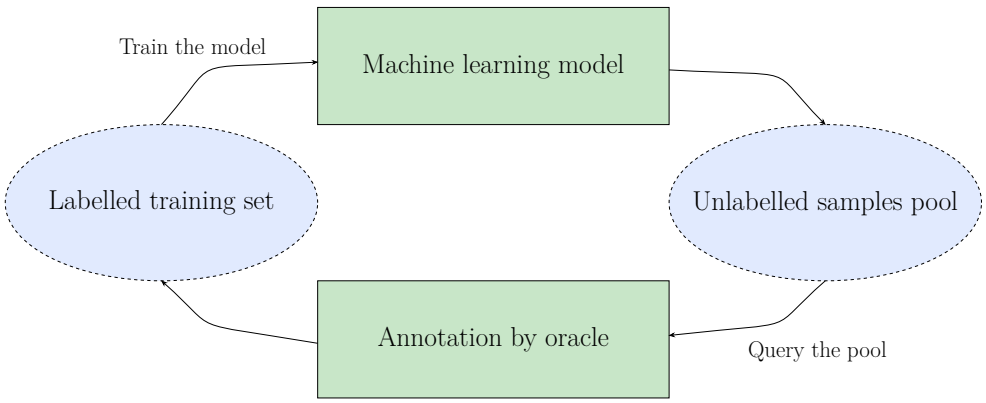

Figure 1.8: Overview of pool-based active learning

The core idea of sampling only a subset of the training data is applied using query strategies. Query strategies are methods to decide whether to query or discard the current instance in the case of stream-based AL or to choose the most informative subset in the pool-based case. For example, the strategy of *uncertainty sampling* selects the instances in which the model is the least certain about what label to give. The selected candidates for annotation are then presented to the oracle, the source of truth, which can be human experts or some more computationally expensive model, based on the modelled domain. The annotated data samples are then stored in the labelled training set, which is used to train the new iteration of the ML model. This process makes the models iteratively improve and provide the same performance with a vastly smaller amount of training data because query strategies can make the training process focus on instances closest to its decision boundary instead of irrelevant ones.

1.5.2 ALF: Active Learning Framework

Active Learning Framework (ALF) [1] incorporates the ideas of active learning into a robust framework for training and maintaining network traffic classification models. The concepts of active learning are used to enable the framework to be deployed for flow-based analysis of high-speed networks, while also presenting the benefits of continuously retraining a model in a highly evolving domain of network traffic. ALF is primarily focused on the paradigm of stream-based AL as it reduces the required storage needed, which is especially useful in the case of monitoring high-speed networks. However, it can also work in the pool-based scenario to evaluate offline datasets.

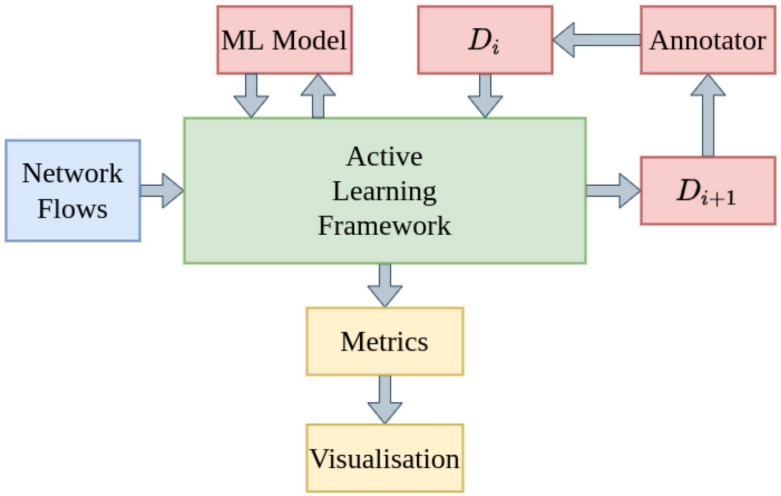

Figure 1.9: General overview of the whole pipeline of ALF; *Image source:* [1]

The broad overview of the ALF pipeline can be seen in Figure 1.9. The whole knowledge of the ML model is continuously updated and represented by the current training dataset D_i. In the beginning, the initial dataset (D_0) is presented, together with the chosen ML classifier, annotator, etc. For each retraining loop, a new version of the classifier (M_{i+1}) is created, and the performance of the new model is monitored. The one retraining loop generating the next iteration of the dataset can be seen in Figure 1.10 and can be divided into the following phases:

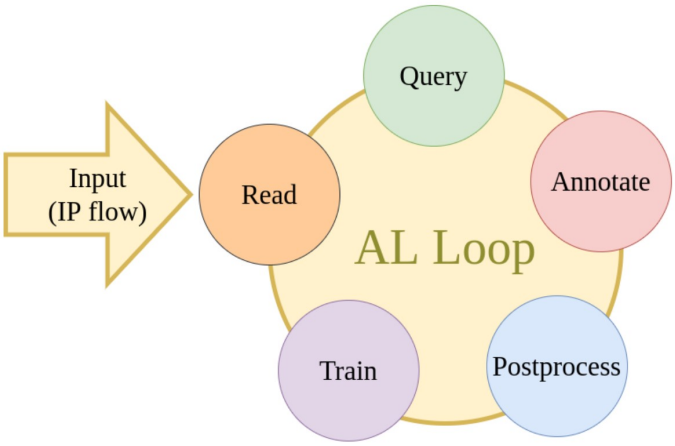

Figure 1.10: Overview of the ALF update loop. *Image source:* [1]

Read stage processes all the incoming network flows. Data gets preprocessed into features suitable for machine learning training and inference. Prediction of the current model M_i is done on the whole batch of incoming flows.

Query uses strategies to sample a subset of data to annotate and include in the next generation of the dataset.

Annotate stage provides the labels for the currently selected subset of data. The role of the oracle is represented by a software module that uses additional sources of knowledge (e.g., OSINT, system logs, service logs, or any monitoring/auditing tools running at end-point devices). These annotators may not be usable in all scenarios, or their time complexity may make them infeasible at large scale, which is the case in the decryption of encrypted traffic, for example.

Postprocess stage presents a novel addition to the usual AL pipeline. It allows the analysis of the quality of the current dataset and addresses possible quality issues. For example, undersampling is done to have a

dataset of manageable size, and class imbalance is handled. The detection of concept drift would provide an additional model-enhancing procedure that would help ALF update the datasets appropriately. These procedures are often time-consuming and may be done once per longer time period.

Train stage represents the training of the new generation of ML model (M_{i+1}) based on the latest version of the dataset (D_{i+1}).

Authors of ALF put great emphasis on creating a robust framework that goes beyond implementing the common active learning loop and focuses on dataset optimization and quality assessment. Detection of concept drift may present an additional step beyond the currently implemented performance monitoring and dataset optimization procedures. The retraining process could become guided by the drift detector and make the model avoid unnecessary retraining or increase it under a sudden change of the underlying distribution. Because of that, the authors perceived drift detection as an open research challenge. This thesis tackles the challenge by creating a tool that can implement these changes into ALF, among other use cases.

Chapter 2

Analysis of network traffic behaviour

The theoretical research introduced many challenges in creating robust, well-performing and long-lasting machine learning models for network traffic classification. It was described how the models may underperform when the capturing process doesn't address the possible biases and how concept drift can make the models become degraded over time. The goal of this thesis is to create a concept drift detector that can be a part of methods for combating those challenges. To create a solution for said problems, one should first analyse them and understand their behaviour. Because of that, this chapter provides an analysis of multiple network traffic datasets, documents how network traffic evolves over time and showcases the effect of concept drift and biases on the performance of machine learning models. The knowledge gained in this chapter aims to aid in creating a concept drift detector tailored to the domain of network traffic and provide useful insights into the best practices for creating network traffic classification models.

2.1 MAWI WIDE dataset

In order to properly study the evolution of network traffic, the first experiments analyse it from the long-term point of view. MAWI WIDE dataset [36] presents a perfect candidate for such long-term analysis as the MAWI Working Group has provided traffic samples since 1999. They provide daily traces from various sampling points, with sampling point F being operational since 2006, enabling us to study how the network traffic has evolved for 17 years. The mentioned sampling point monitors the transit link from the WIDE academic network in Japan to an upstream provider. With some exceptions, they sampled the traffic each day for 15 minutes, starting at 14:00. The whole trace can be downloaded as a tcdump file, and the website aggregates various statis-

tics describing the current sample. For the experiment, a website crawler was created that parsed statistics of the daily traces as they are used in studying the long-term evolution of network traffic.

The first analysis examines how network throughput evolves over time as it is a general statistic of the state of the network that can also have an impact on the possible statistical features used in the creation of ML models. The time series of the mean daily throughput is visualised in Figure 2.1. The following insights can be gained from observing it: The behaviour of network traffic is highly dynamic, with a large variance in the observed data. Other than this behaviour, there are other global and local trends present. Some of them may be outliers – daily anomalies whose source may be an unexpected load of the network that day or an error in their sampling process, but the time series was cleaned up from the most easily recognisable outliers before plotting it. There exist more intense network changes which could severely impact the following performance of the model trained on the data before the change, presuming throughput can be used as a general metric influencing the feature vector. Such change can be seen in 2015 when the values doubled on average in a matter of months.

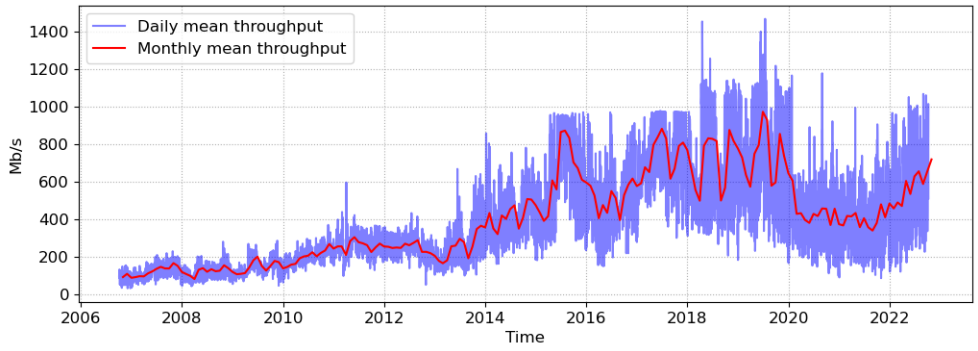

Figure 2.1: Mean daily and monthly throughput of the MAWI WIDE dataset.

Further analysis focuses on a six-month long time window, visualised in Figure 2.2. The noise in the data represents a recurring periodic behaviour, not some random variance. This implies that the weekend network traffic differs from the one during a working week. On top of that, the time series was well predictable when analysed using models for time-series modelling, illustrating the strength of the periodic behaviour. This presents the suggestion that network traffic periodically changes, which could impact the performance of classification models (assuming the relationship between the throughput and the used features), meaning that models trained on working week data would be less capable of recognising weekend data and vice versa. However, to fully explore that idea, an analysis of additional examples of traffic from other networks has to be made to assess whether the assumption is correct and whether this periodic trend can be seen in other datasets.

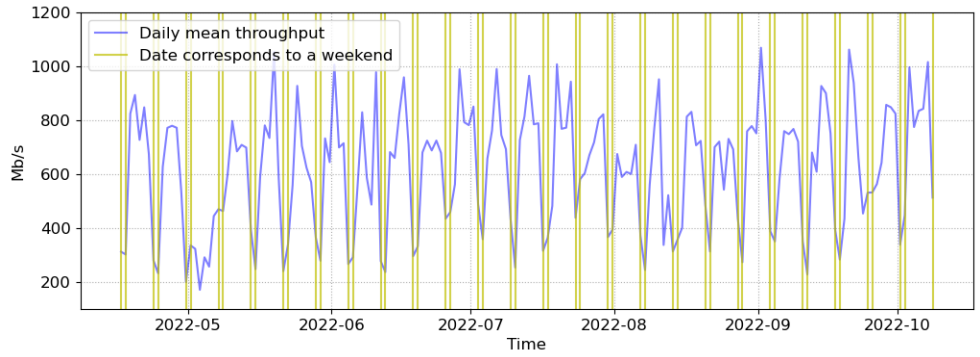

Figure 2.2: The time series of the mean daily throughput of the MAWI WIDE dataset, with a six-month subset of data being presented.

2.2 CESNET TLS dataset

The following experiments further explore the idea of network traffic evolving periodically through the week and the performance of classifiers differing on working week versus weekend data. In other words, the experiments aim to indicate whether concept recurrence can be observed with weekend traffic representing a different concept than working week traffic.

The aforementioned theory was tested on a year-long dataset capturing a large number of web services (180) running in HTTPS communications called CESNET-TLS-Year22 [37]. With some exceptions, that dataset provides network flows that were captured continually through the year 2022. Some experiments have been conducted on a reduced sample set provided by the supervisor. The whole dataset was later published on their DataZoo platform [14].

2.2.1 Simulated model deployment

The goal of the following experiment is to support the discovery of the periodic behaviour of network traffic. In order to achieve that, a classifier was created, and the dataset was then used for simulated model deployment. The gradual decline of the model over time was observed. The whole first day was used as a training dataset, which was the 1st of January 2022 – a national holiday and Saturday. For the remainder of the dataset, a five-minute sample beginning at 10:00 was used to represent the day because of the computational and storage requirements of such a large experiment. This decision was made to emphasise the periodical behaviour and showcase the effect of biases, but it shouldn't be made when designing actual models as it represents an example of sampling bias. The model is trained on a subset of data and may thus overfit to data patterns only correct at that point in time instead of learning how the modelled network traffic generally behaves.

This reduced version of the dataset was available in the form of flows captured by the ipfixprobe flow exporter with the PSTATS plugin used. To generate a feature vector from the captured flows, Feature Exploration Toolkit [38] was used. It uses a variety of statistics, such as mean and standard deviation regarding packet lengths, inter-arrival times, etc. Instead of services themselves, general categories, such as streaming media, file sharing, etc., were used as classes. Histogram-based Gradient Boosting classifier from the sklearn library was used as the model for this experiment as Gradient Boosting Machines (and XGBoost used in following experiments) represent highly popular and effective model architectures currently used.

The results of the experiment simulating model deployment can be observed in Figure 2.3. F1 scores of the model prediction for the daily data sample represent the quality of the model's predictions to judge how it would degrade over time. We can gain many interesting insights into the behaviour of models for network traffic classification. First is how quickly the model's performance deteriorated, with the F1 score dropping from 0.92 on the first test day to around 0.7 in the following working days. This implies that the behaviour of weekend traffic indeed differs, and the model may have overfitted to the patterns only present in the weekend data and failed to generalise properly. This presents a strong practical example of sampling bias and urges following more robust procedures when designing models for network traffic classification. If not properly addressed, one would create a model with satisfactory validation scores that would nonetheless fail to perform in the deployment phase on a different day.

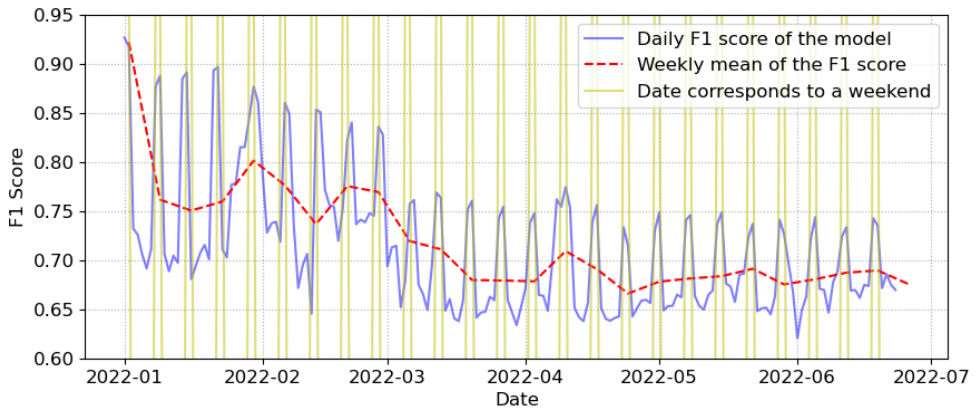

Figure 2.3: Results of the simulated model deployment experiment on the TLS dataset. F1 scores indicate how the model degraded over time.

The strong differences in performance continued throughout the whole dataset, which urges further investigation of how the behaviour changes when training on the dataset sampled on the whole first week. When assessing the general trends other than the periodic behaviour, one can see an example

of a possible strong concept drift at the beginning of March, indicated by an additional drop in performance. Some distribution changes probably impacted the performance of the model. This behaviour was explored and analysed in more detail in the following experiments. As with the previous experiment, the resulting time series was modelled and analysed, and the observations were the same. The change in the model's performance is well-predictable and thus deterministic, which one may not expect.

In order to study how to make models more robust and provide examples of challenges already present in the usual process, further experiments studied the causes of model degradation and looked for proof of the suspected distribution change at the beginning of March. The second experiment with this dataset was designed to observe the development of the most correlated features with the F1 Score of the models. The comparison of the time series of model quality and mean values of the two most correlated features is visualised in Figure 2.4. Many features were studied in this way, all showing similar behaviour, with these two plotted – *lengths_std* (standard deviation of packet lengths) and *fwd_pkt_iat_max* (maximal packet inter-arrival time in the forward direction) – showcasing the strongest trends.

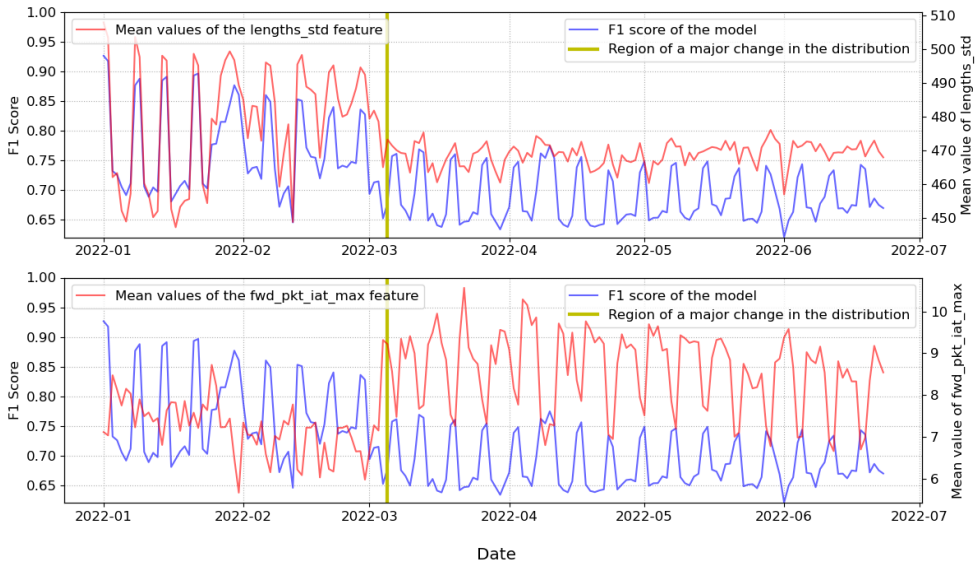

Figure 2.4: Comparison of time series of model quality indicated by its F1 score on a daily subset of data and mean value of *lengths_std* and *fwd_pkt_iat_max* features. A change in the distribution of both features can be observed in March.

The experiment results provide evidence of the distribution change in March. After the change, the variance of values through the week either dramatically decreased or increased. This provides an example of concept drift and provides an explanation of the model degradation. On top of that, it can

be noted how heavily correlated the time series are. Before the process change in March, the time series of *lengths_std* and F1 scores are almost perfectly correlated. In creating robust models, the ability of one feature to impact the quality of the model this heavily would not be wanted.

Further experiments explored the robustness of the used feature vector. It was indicated that they are often correlated with throughput, meaning that just a higher load on the network would impact the quality of the model, on top of the aforementioned weekly period. There exists a need to study the robustness of feature vectors used for training network traffic classification models further. However, this falls out of the scope of the researched topic of concept drift and is thus left as a future research challenge.

2.2.2 Concept drift analysis

The main research topic of this thesis is the concept drift itself as the goal is the creation of a novel concept drift detector tailored to the needs of network traffic classification. Because of that, the focus shifted from analysis of the features and simulated model deployment to using existing solutions for concept drift detection and observing how to tailor the created drift detector. The goal of these experiments is first to explore how the drift detection approaches react to the previously described periodic development and to the major process change in March. Other experiments then serve to gain insights used in the development of the drift detector.

The library for evaluating machine learning models called Evidently [39] was used to detect drift in this section. In the case of numerical data, the default drift tests Evidently uses are the Two-sample Kolmogorov-Smirnov test (KS test) or the test based on the Wasserstein distance in samples with more than a thousand data instances. A test is built for each feature and the whole sample is considered drifted if more than half of the features are detected as drifted. A feature is considered drifted with the rejection of the null hypothesis of the KS test that the two samples come from the same distribution, with the default value of $\alpha = 0.05$. In the latter case, normalised Wasserstein distance is used, with the default detection threshold set at 0.1, meaning that drift is detected when the cost of transforming one distribution to another is more than 10% of standard deviation. This presumes the default thresholds are used.

Experiments with both approaches have been made – undersampling the dataset and using the KS test or working with normalised Wasserstein distance. The results from both of these methods were similar, with the KS test being more sensitive as it usually marked more features as drifted. Creators of Evidently suggest [33] that this sensitivity of the KS test increases even more in the case of large datasets. Because of that, the following experiments are based on normalised Wasserstein distance, as the created drift detector should be designed to work with hundreds of thousands of data instances.

Another question that presented itself is how to deal with the fact that the usual feature vector can have up to a hundred features, and some of them have a vastly larger impact on the model's decision than others. It was later observed in an experiment with a different feature vector that one of the features was static and had no use whatsoever. This feature should be thrown away during preprocessing, but this work aims to create a robust drift detector that can also be used for quick analysis of unknown datasets. It was experimented with choosing a subset of the most prominent features and making the final detection in other ways than by shares of drifted features, such as counting the mean drift severity across all features. More complex methods will be later used in the final developed detector but in the case of the first analysis of the drift present in this dataset, a subset of the ten most prominent features (based on the feature importance provided by the trained model) was used. The whole daily sample is considered drifted if half or more of those features are drifted. The historical data comes from the time window of the last seven days and the new daily data represents the current one. The results of this drift detection experiment can be seen in Figure 2.5.

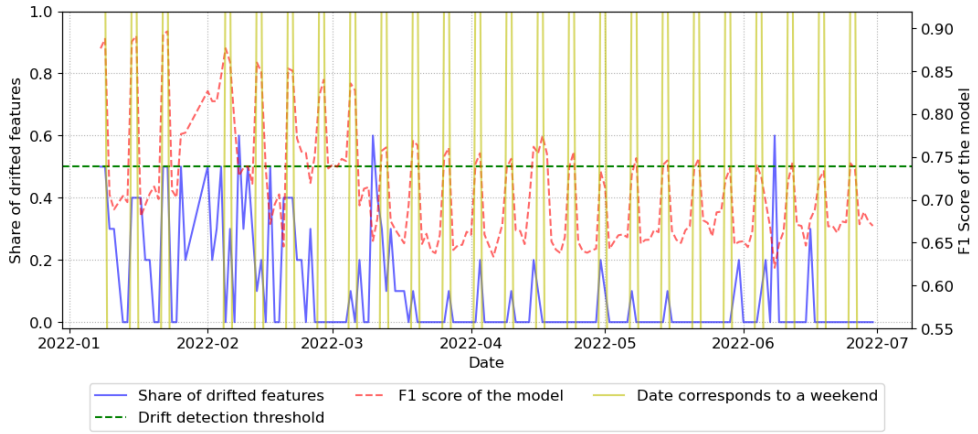

Figure 2.5: Results of the TLS drift detection experiment; *Subset of ten most important features was chosen and concept drift detection test based on normalised Wasserstein distance was used. Share of the features detected as drifted is shown with a share larger than half or equal would mean that drift was detected at the whole daily sample (with a historical window of last week).*

It can be observed that the distribution varied the most before the process change in March, where simply a difference between working week and weekend traffic could trigger drift detection or be close to triggering it. There was a data outage at the end of January associated with multiple drift detections. Another drift detection occurs in March, accompanying the previously discussed process change, providing evidence that drift detection algorithms would detect it and may thus guide the dataset update and prevent the model

from deteriorating. Another drift detection happened in June, following the drop in performance to the lowest measured F1 score. The performance improved the next day and no drift was detected, meaning it may have been a temporary anomaly.

The previous experiment showcased the traditional approaches to drift detection in choices of historical and current windows that would detect concept drift at the current timestamp. In the next experiment, the idea of concept drift is further generalised to study the development of the distribution, which can be used for additional analysis and a better understanding of the behaviour of network traffic in this dataset. Drift is detected for each day of the test data (data from said day – D_i – representing the current window) and four tests are carried out, each with a different choice of historical window.

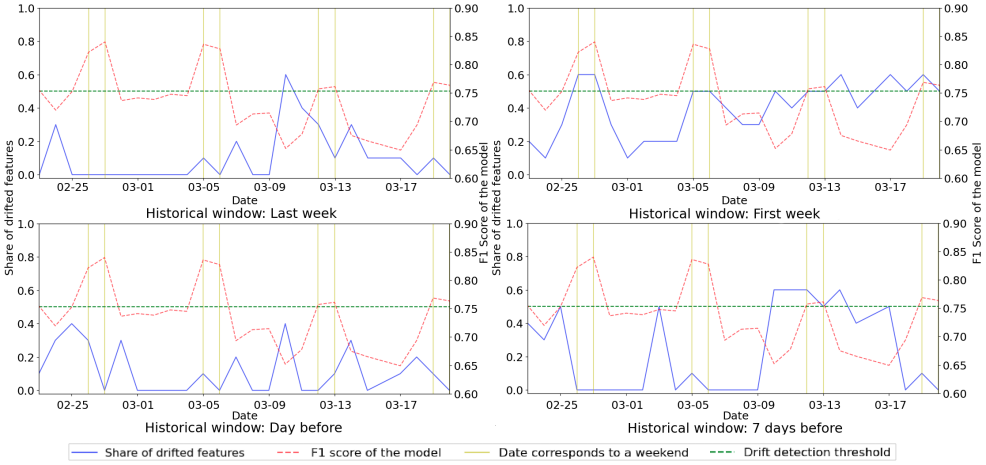

Figure 2.6: Analysis of the development of the TLS dataset by additional historical window choices; *Drift detection test using normalised Wasserstein distance is carried out for new daily data subset (D_i). The following choices of historical windows have been made: Last week – D_{i-7} to D_{i-1}, First week – D_0 to D_6, Day before – D_{i-1} and seven days before – D_{i-7}.*

The results of this experiment zoomed at the area of process change in March can be seen in Figure 2.6 and are analysed below:

Last week represents the usual approach to drift detection, also used in the previous experiment. Data from days D_{i-7} to D_{i-1} are used for the historical window. It can be studied that the process change was detected in one day, after which the performance worsened, but no more detections were made after that.

First week uses data from days D_0 to D_6 for the historical window. This experiment represents the cumulative development of the distribution compared to the beginning of the dataset. Compared to the first week, each weekend's data would be considered drifted. After the process

change, the data would be considered drifted (or nearly drifted) for each day. This provides evidence of the model being degraded as the distribution has substantially changed and retraining was needed.

Day before compares the days D_i and D_{i-1}. The differences between the consecutive days aren't strong enough to detect the process change, and spikes in the share of drifted features are located around the weekends.

7 days before analyses the changes cleaned from the periodical effects, comparing D_i and D_{i-7}. Some detections occurred on days when no other window choices had an increased share of drifted features and the effect on the performance isn't visible. However, the most revealing is the behaviour around the process change, where the distribution vastly differed from the one from the same day last week for seven consecutive days, providing further proof of this idea of a more complex process change happening.

2.3 CESNET QUIC dataset

To properly assess whether the patterns observed on the previous datasets represent general behaviour, similar experiments were carried out on another dataset. CESNET-QUIC [40] dataset provides four weeks of data samples captured in November 2022 and is comprised of 102 captured services using the QUIC protocol. It is also available on the CESNET DataZoo platform [14] in the form of already preprocessed flows with features suitable for training machine learning models.

With this new dataset and feature vector, the experiment of simulated model deployment was repeated. In this case, a Histogram-based Gradient Boosting classifier was used to classify the dataset into a chosen subset of general categories, such as streaming media, file sharing, etc. The first week was used as a training dataset and the remaining three weeks were used for testing. The trained model was used to classify daily subsets of the testing dataset, showcasing how the model would degrade over time. The results of this experiment can be seen in Figure 2.7. The previously observed periodic behaviour is also present in this dataset, although the differences in performance seem to be less severe. On top of observing differences in weekend traffic, a change in behaviour can be seen during the Czech national holiday on November 17, when the performance improved, similarly to the weekend traffic. Outside of this period, slight degradation can be observed during the first week of testing, after which the downward trend stabilised. This periodic behaviour was observed with the choice of different datasets and different feature vectors, so it can be presumed that this issue generally exists in the area of network traffic classification.

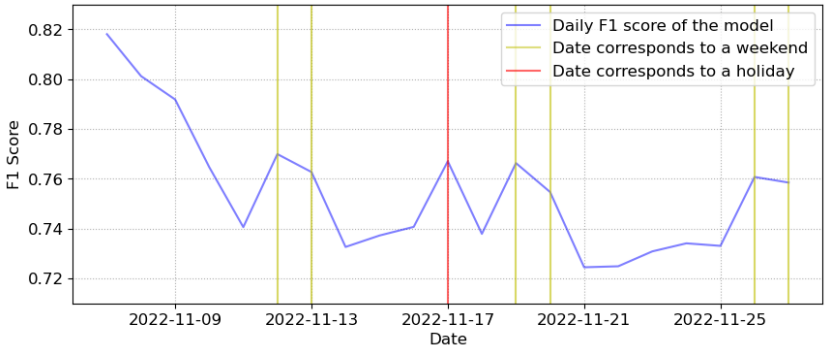

Figure 2.7: Results of the simulated model deployment experiment on the QUIC dataset. F1 scores indicate how the model degraded over time.

As with the previous dataset, the Evidently library was used to detect concept drift present in the dataset. The test based on the normalised Wasserstein distance was performed and its results can be seen in Figure 2.8. For the current time window of data from day D_i, a historical time window is created from the data from the last seven days (D_{i-7} to D_{i-1}). For this choice of classes and undersampling used, no drift detection would occur under the default thresholds Evidently uses. It can be observed that usually, no features or only a minor number were detected as drifted, with slight increases during the degradation in the first week and during the holiday on November 17. Because of these observations, it can be presumed that the majority of features were drifted but slightly under the detection threshold, or the few features detected as drifted had severe changes in distribution and could cause the model degradation themselves. A further analysis of this behaviour is needed.

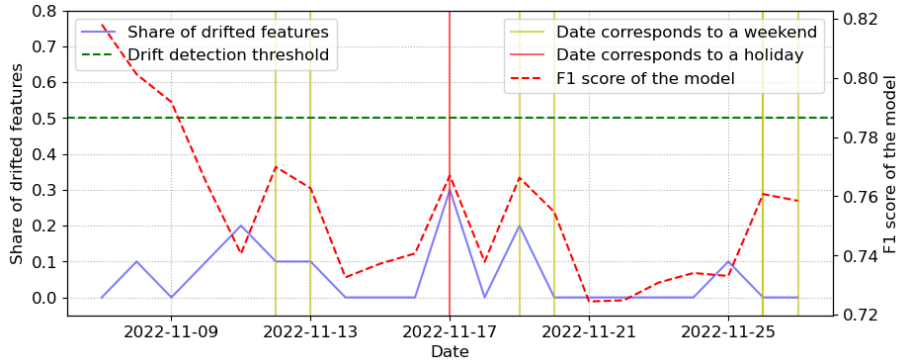

Figure 2.8: Results of the QUIC drift detection experiment; *Subset of ten most important features was chosen and concept drift detection test based on normalised Wasserstein distance was used. Share of the features detected as drifted is shown with a share larger than half or equal would mean that drift was detected at the whole daily sample (with a historical window of last week).*

Because of these discoveries, an experiment was designed to study the drift of the features separately. Drift detection tests, historical windows, etc., were the same, but the features were then analysed independently, with the most illustrative chosen for visualisation in Figure 2.9. It was discovered that a single feature, *PSIZE_REV_4*, was consecutively detected and, in some cases, more severely drifted. In some cases, the features presented sudden spikes in drift severity or were slightly under the detection threshold. Because of this observation, the created drift detector should have the ability to fine-tune the detection thresholds and work with a more robust approach of combining the final decision than by the share of drifted features.

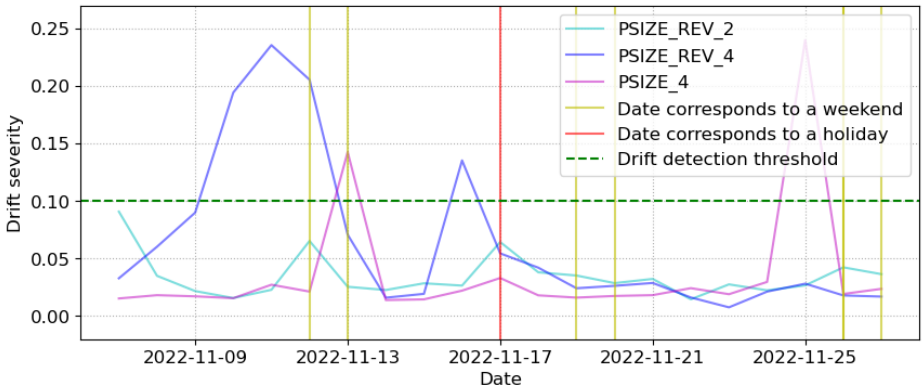

Figure 2.9: Independently detected drift of the features of the QUIC dataset; *Drift detection test based on normalised Wasserstein distance was used to detect drift independently on each feature and the development of three chosen illustrative features is shown.*

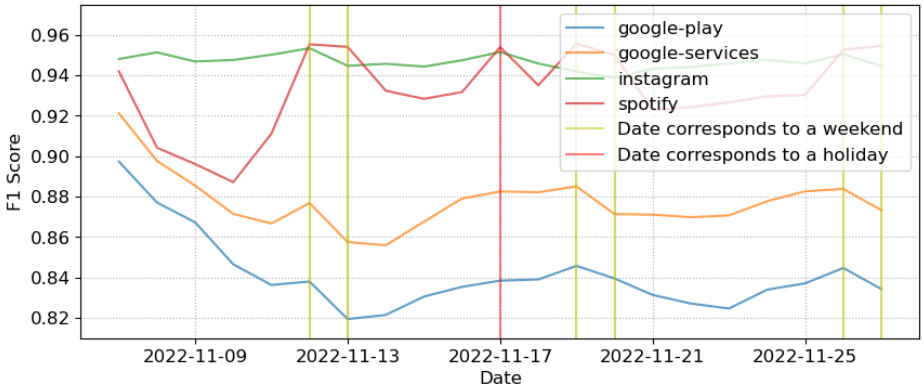

Figure 2.10: Results of the simulated model deployment experiment on the QUIC dataset. A subset of services was chosen. It can be observed how the model degrades for each class.

The following experiment analysed how the individual classes can drift in different ways. A new model was created to classify the subset of the dataset of several chosen services, and the same simulation procedure was replicated to test the model, with the F1 scores measured independently for each service and the experiment results visualised in Figure 2.10. It can be observed how some of the classes (e.g. *Instagram*) present no visible decline in performance, and some classes (e.g. *Spotify*) recovered after a short underperformance period. Another case was when the drift occurred and the performance never recovered (e.g. *Google-services*). Luxemburk et al. [16] provided a study of QUIC traffic classification using the same dataset and discovered concept drift present in various services from Google. Because of that, a drift detection experiment was performed specifically on data from *Google-services* class. In this case, the KS test was performed at it is more suitable when presented with a lower amount of data instances. The chosen subset of features and windows were the same. Results of this detection can be seen in Figure 2.11, and the share of drifted features is higher, especially during the first week. However, drift detection would only occur in one day, further showcasing the importance of a more robust method of providing the final decision of whether the drift was present in the whole sample.

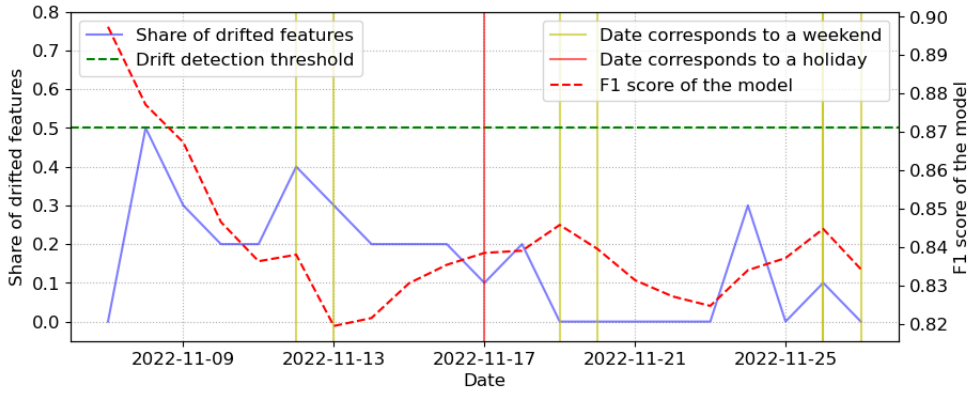

Figure 2.11: Drift detection of the Google services class; *Subset of ten most important features were chosen and the KS test was used for the detection. Share of the features detected as drifted is shown with a share larger than half or equal would mean that drift was detected at the whole daily sample (with a historical window of last week).*

The authors of the study suspect that a change in the certificate could have been the source of the observed drift. The experiment of independently detecting drift for each feature was repeated on *Google-services*, visualised in Figure 2.12. In this case, the training dataset from the first week was fixed as a historical window as this helps discover whether the changes are permanent. It can be seen that the previously discovered feature, *PSIZE_REV_4*,

drifted and never recovered and may be the source of drift. Another feature, *PSIZE_4*, was severely drifted during the first couple of days but then recovered to the same training distribution. These discoveries support the hypothesis of change in a certificate being the source of the drift as it would correlate with the change of the *PSIZE_REV_4* feature.

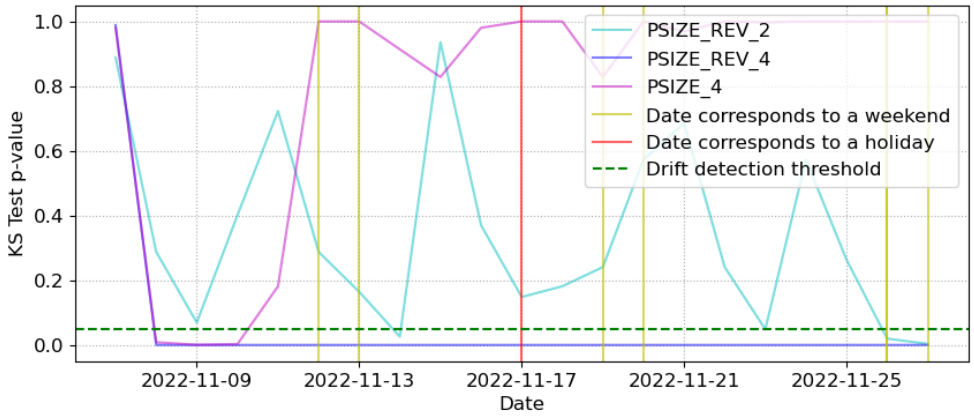

Figure 2.12: Independently detected drift of the features of the Google services class; *KS test was used for the detection and differences between the training and current dataset were compared. P-values of the test are shown, with values under the threshold meaning drift presence.*

2.4 Conclusions of the analysis

This chapter provided three main points of view on the topic of creating robust network traffic classifiers. One was to analyse the behaviour of network traffic and practically observe the challenges that need to be addressed. Subsequent experiments showcased how the models would degrade if those challenges were not handled. The last experiments worked directly with existing methods of concept drift detection and observed how they would detect drift in the datasets they used. The conclusions gained from those experiments can be summarised as follows:

Behaviour of network traffic: The periodic development of network traffic represents the first major discovery about the behaviour of network traffic. The paradigm of multiple recurring concepts was observed, with working week traffic having a different distribution than the weekend one. The statistical features used as a basis of ML models are often correlated with throughput and thus depend on the current load on the network, which can differ not only during weekends but also holidays.

Model degradation: If the previously discovered challenges aren't properly addressed, the model may quickly degrade. An experiment illustrating sampling bias showcased that a model trained only on weekend traffic from a single day may provide significantly worse predictions only a couple of days later (dropping from F1 score of 0.92 to 0.7). A long-term experiment simulated half a year of model deployment and showcased that concept drift may severely and permanently change the underlying distribution. The trained patterns in data may not be accurate anymore and the model may become ineffective if it is not retrained. This section provided evidence that network traffic indeed represents a challenging and highly evolving domain, and further work is needed in creating robust network traffic classifiers. It was also showcased that there may be more sources of model degradation, and the performance of the individual classes may evolve differently.

Concept drift: The last section experimented with existing methods of concept drift detection to demonstrate how they react to the periodic behaviour and the previously discovered concept drift. The relationship between drift being detected and model underperforming was illustrated. It was discovered that in some cases, the methods discover drift in places where it is expected by the underperformance of the model, but in other cases, a low number of severely drifted features seemed to be the cause of degradation. Because of that, a search for a more robust approach of combining the final decision than by the share of drifted features is advisable. The idea of detecting concept drift was then generalised for additional analysis of the distribution change, where various choices of time windows enabled different observations of how the distributions develop. For example, if drift is calculated compared to the reference training dataset, it may stack up, and the model may further degrade.

Chapter 3

Concept drift detector

The main goal of this thesis is the creation of a prototype for a novel concept drift detector tailor-made for the domain of network traffic. The observed patterns from the thorough analysis of multiple network traffic datasets, such as the periodic behaviour or a single heavily drifted prominent feature being the cause of model degradation, serve as a basis to fine-tune the detection method. The primary use case of the drift detector is incorporating it into the existing infrastructure of the Active Learning Framework, where it can guide the model retraining by the drift detected. During the development of the detector, the focus was also given to the secondary use case of a tool for offline analysis of existing datasets, and additional features were implemented to help further with this use case.

3.1 Design choices

The main approaches to designing concept drift detectors are the *Error rate-based* detectors, *Data distribution-based* detectors and those utilising *Multiple hypothesis tests*. *Data distribution-based* methods are currently implemented as they are usually based on well-explainable metrics, which address the root cause of the concept drift, and the detector may be used for analysing the drift in datasets without having a machine learning model to observe its error rate. Lu et al. [25] categorise various capabilities and aims of the detectors for better concept drift understanding, which were addressed as follows:

Time of detection: The primary ability of the concept drift detector is to identify whether the drift is present for the current timestamp. This is done by providing the detector with two distributions – historical and current. Drift is detected if the tests provide a statistically significant decision that the two samples come from different distributions, and a signal is sent, which can be used to make the model update itself to react to the change of concept.

There exist multiple ways of choosing the historical and current time windows and their lengths. As the primary goal of this detector is to expand the existing infrastructure of the Active Learning Framework, this process is deliberately left out of the detection method. ALF works with continuously maintaining its knowledge in an evolving dataset. It is presented with a stream of new data to classify and choose which to incorporate into the latest version of the dataset. ALF could thus supply the detector with valid window choices. If the concept of the current stream of data doesn't match the concept of the knowledge base in the dataset, there is a need to update it. In the case of using the detector as a tool for analysing unknown datasets, the user should supply the detector with the choice of windows suitable for the aims of the specific experiment.

Drift severity: The importance of having a representative metric that quantifies the dissimilarity between two concepts was observed during the analysis. Such metric, usually called drift severity or strength, should be well explainable and representative of the observed dissimilarities. The work focuses on finding a suitable severity candidate as it could be used to modify the intensity of the retraining, replacing larger subsets of the dataset in cases of more severe drifts.

In the case of network traffic, the feature vectors often have up to a hundred features, some of them providing negligible information, and thus, a shift in their distribution may not lead to the decision boundary change. Another observed behaviour was a couple of important shifted features leading to major model degradation. There exist multiple methods of factoring in the various feature distributions into the final decision of whether drift is present. One may adopt multivariate two-sample tests which regard the whole feature vector as a multidimensional distribution, each feature corresponding to one dimension. Some methods use dimensionality reduction first. Another approach is running a test for each feature distribution and then combining the decision, usually by the share of the drifted features.

However, none of these approaches address the previously described issue of various features differently contributing to the final model degradation. Because of that, the tests are run independently for each feature distribution and their drift severities stored. They may be used for further analysis of the most drifted features, features that are often drifted, etc. The final drift strength is then calculated as the weighted arithmetic mean of those individual severities, where the feature importances supplied by the classifier serve as weights. Because of that, the detector will be triggered by cases such as observed in the TLS-QUIC dataset, where a couple of important and heavily drifted features were the cause

of model degradation. When analysing a dataset without the presence of an ML model to supply the weights, the unweighted mean is used. The default threshold corresponds with the commonly used drift detection if half or more features are considered drifted. The detector considers the whole sample drifted when the final drift strength exceeds 50% of the detection threshold of a single test, presuming default thresholds are used. This presents a novel approach to tackle these challenges.

Drift location: Some methods exist which can also provide where are the regions of concept drift located by partitioning the feature space, running multiple tests, and associating the detection results with the corresponding regions. However, different approaches were chosen to analyse where the drift is located. Firstly, calculating the drift severity independently for each feature has the advantage of making the monitoring of which features are often drifted possible. Secondly, the detector is designed to run secondary tests for each class independently and provide information on which classes are the most drifted. The workings of this class detection will be further described in the following sections.

3.2 Implemented tests

As reasoned above, *Data distribution-based* methods were implemented as they are well suited for our use case. The detector was designed to be modular and can be easily expanded by implementing new tests. They have to follow the general interface of a drift detection test, which means providing the decision if a single feature should be considered drifted and calculating the final drift severity for the whole sample. Commonly used default detection thresholds are implemented in the tests but may be changed during the construction of the test instance. Currently, there are three drift tests implemented:

Kolmogorov–Smirnov test is built around the null hypothesis that the two samples were drawn from the same underlying distribution. Drift severity is represented by the p-value of the test, meaning the lower the value stronger the rejection of the null hypothesis and the stronger the drift. The default α level used is 0.05, so features with lower p-values are considered drifted. The KS test is best suited for comparing lower sample sizes (e.g., under 1000) as it can be overly sensitive for larger sample sizes. It is defined for continuous distributions, so it isn't suitable for detecting drift on categorical data.

Wasserstein distance measures the dissimilarity between two distributions and represents the minimal cost of transforming one distribution to another. The absolute Wasserstein distance is then normalised by dividing it by the standard deviation, resulting in a fairly explainable drift severity. The default detection threshold is 0.1, meaning features with larger

normalised distances are considered drifted. It is advised to be used as the primary method for the global drift detector. However, it is also only well-defined for continuous distributions, so another test should be used for categorical data.

Jensen-Shannon distance is the square root of Jensen–Shannon divergence (symmetrised and smoothed version of the Kullback–Leibler divergence). It is a metric ranging from zero to one, zero meaning identical distributions and one completely different. The default behaviour of the test is that features with a distance larger than 0.1 are considered drifted. As it is calculated by comparing two probability vectors, binning has to be performed on the distribution. The main advantage of this approach is that it may be used for categorical data.

3.3 Detector infrastructure

The general overview of the drift detector infrastructure can be seen in Figure 3.1. The design of the drift detector is modular and some modules may be used for further analysis. The detector may be tuned to the needs of the user by specifying the various modules. After the detector and its modules are initialised, the intended usage is to provide it with the historical and current data samples for a single round of drift detection. The primary and mandatory module is the global drift detector, which provides the drifted features, drift strength and signal to retrain the model. Further analysis may be done by running tests independently for each class and studying which classes are the most drifted. The drift analyser classifies the occurring drift into categories to further explain how the distributions develop by running separate tests for its own time windows. Each of these modules may be configured to work with different tests independently of each other. The logger module may be used to store the test results and prevent the need for outside monitoring of the detection results.

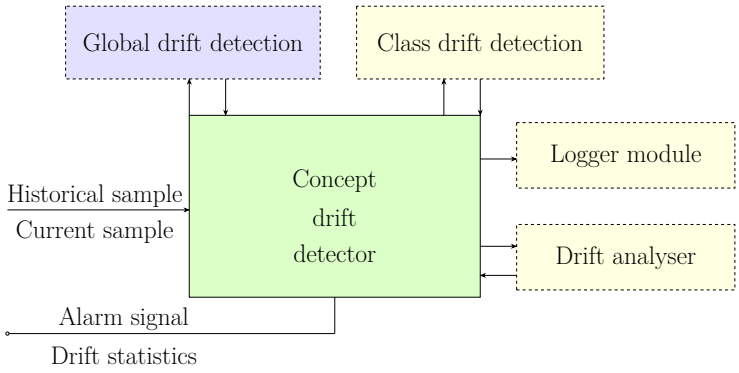

Figure 3.1: Overview of the infrastructure of the created drift detector

The detector is initialised by supplying it with the configuration for the primary global detection and optionally for the class drift detection. Logger and drift analyser are then supplied as instances of their individual classes, with the analyser being set up by the same `Config` class as the other modules. It encapsulates the following possibilities for configuring the detector:

chosen_features expects a list of feature names specifying the subset of features to run the drift detection on. Some of the tests are not suitable for categoric features, so this argument may be used to exclude them.

feature_importances expects a Pandas series of model's feature importances indexed by the feature names. It is used for supplying the weights to the final drift strength calculation. It is an optional argument and an unweighted mean is used if it is not specified.

drift_test expects an instance of `Test` class with the test used to provide the decision on which features are drifted and calculate the final drift strength.

class_name is an optional argument for the class drift detector and expects the name of the column which contains the class labels.

The process of a single round of detection is performed by calling the `detector.detect()` method and providing it with the two samples to compare, presuming `detector` is an instance of `DriftDetector` class. The expected format of the samples is a Pandas dataframe. The single round of detection carried out by said function call returns a boolean value of whether the drift was detected and an alarm signal was sent. Presuming all of the implemented modules are used, the round of detection is comprised of the following procedures:

Global drift detection provides the primary decision on whether drift was detected for the current choice of data samples. For each of the chosen features, the chosen detection test is run and the drift severities are measured. They are then used to decide which of these features are considered drifted and for calculation of the final drift strength. Global drift statistics are returned by calling `detector.get_drift_statistics()` and the Pandas series of drifted features with their severities by calling `detector.get_drifted_features()`.

Class drift detection was implemented to present additional information about the drift location. It requires the existence of class labels, which are available when analysing an existing dataset or may be provided by the oracle in the case of active learning. Each sample is grouped by the specified column containing the class labels, and a detection is run on each group corresponding to a class that is available in both samples.

The same process of independent tests for each feature and calculation of final drift strength is performed for each group. The results of this analysis may be returned by calling `detector.get_class_drift()` and are presented as a Pandas dataframe containing the drift strength, the share of drifted features and a decision whether the class is considered drifted for each class.

Drift analyser presents a novel approach to concept drift understanding by classifying the drift into multiple classes, somewhat inspired by the theoretical categories used to describe various types of drift. Implementation of this idea is done using a process of combining detection from several separate windows, where the analyser itself stores its own data subset to perform the analysis on. The current implementation uses a week-long time window, which is then divided into additional subsets and multiple detection tests are performed. The overview of the window choices for the specific test used for the drift classification may be seen in table 3.1.

Table 3.1: Window choices for analysis module tests

Test name	Historical window	Current window
Last week	D_{i-7} to D_{i-1}	D_i
Yesterday	D_{i-1}	D_i
Week ago	D_{i-7}	D_i

There are currently five drift classes given by the analyser. *Unknown drift* label is given before sufficient time for collecting the analysis data sample passes. *No drift* label is given when there is no strong chnage present. It was observed how network traffic may shift between recurring concepts, especially between the concept of working week and weekend. *Periodic drift* label is for cases when the drift may be explained by the switch of these concepts and behaviour is similar to what it was a period ago. When the drift is not expected and some unforeseen change in the underlying distribution has occurred, *Sudden drift* label is given. *Incremental drift* describes the case where the distribution is in the process of changing between two concepts, and the intermediate concept is present.

The classification is currently implemented by logic inferred from the analysis of the datasets. For each test corresponding to the choice of windows, it is decided whether the drift occurred, and the final classification is done by combining these results. The decision tree representing the classification can be observed in Figure 3.2. The `Analyser` is an abstract class, so more advanced classifiers of drift types may be implemented in the future.

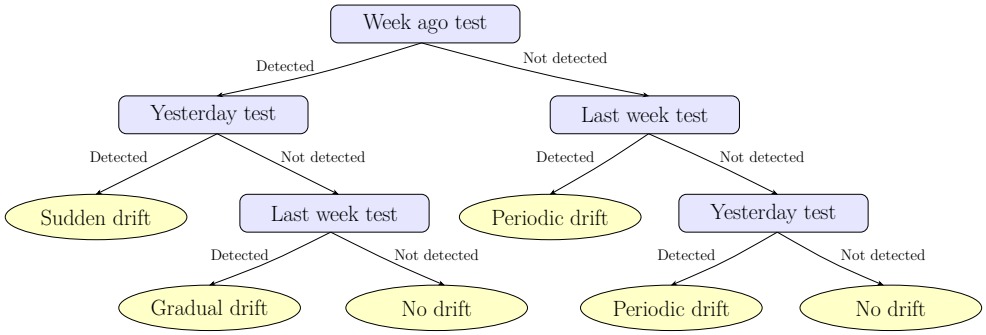

Figure 3.2: Decision tree represents the workings of the currently implemented drift type classifiers. Multiple tests are performed, and their results correspond to the tree nodes. The decision of the classifier is presented in the leaves.

Logger module provides a useful tool for storing the drift results without the need for outside logging of the drift development. It is available in the form of an instance of `Logger` class passed to the detector during initialisation. Its contents may be returned by calling the `get_logs()` method and are available as a Pandas dataframe. An example of a detection log may be seen in table 1.8.

Table 3.2: Example of a log monitoring the test results

Date	Detection	Strength	Features share	Type
3.1.2024	False	0.027	0.025	Unknown
4.1.2024	False	0.021	0.0	No drift
5.1.2024	False	0.029	0.025	No drift
6.1.2024	True	0.057	0.35	Sudden drift

3.4 Detector showcase

Many experiments have been performed on the available datasets from CESNET while developing the various modules of the detector and testing its performance. The following experiments demonstrate how the detector may be used for the analysis of an unknown dataset. A similar simulation of the model being deployed and monitored how it degrades over time was performed, this time using the feature vector supplied by CESNET's DataZoo package. Currently highly popular XGBoost classifier was chosen as the model infrastructure and the model was trained to classify all the available services. The first week was used as a training dataset, and the following months were used for testing. One may use the detector to analyse how the dataset evolves over time and choose the training dataset as the historical sample given to the detector. Another possibility is using the floating time window of the last week.

Both of these approaches are compared in the experiment results visualised in Figure 3.3. It may be observed how the individual daily drifts would stack up compared to the training dataset if no retraining were to be done. This presents another example of the importance of concept drift detection.

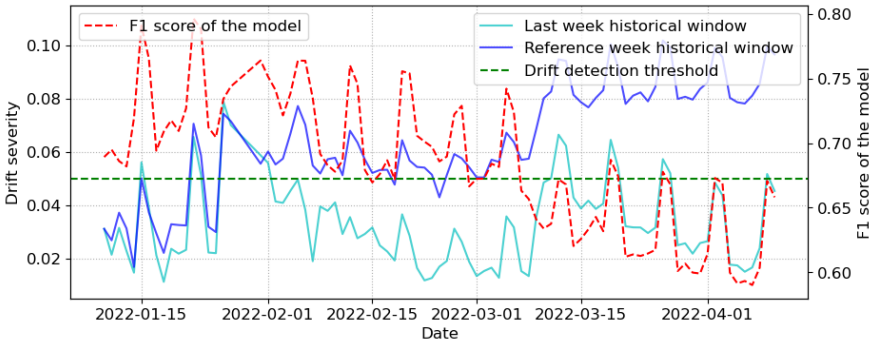

Figure 3.3: Detector showcase on the simulated deployment of a model using the TLS dataset. The choice of the historical window of either the last week of data or the reference training week may be compared.

The decisions of the drift analysis module can be seen in Figure 3.4. The module needs some time to gather enough data to make the decision. After that, the classes given by the analyser are as expected. On Mondays, the drift is analysed as periodic, where a change between the two recurring concepts was spotted but is expected. When the change was larger than expected, the drift was classified as sudden. In the case of drift continuing the following day, it was labelled as incremental, as it probably represented an intermediate concept during the change between two concepts happening.

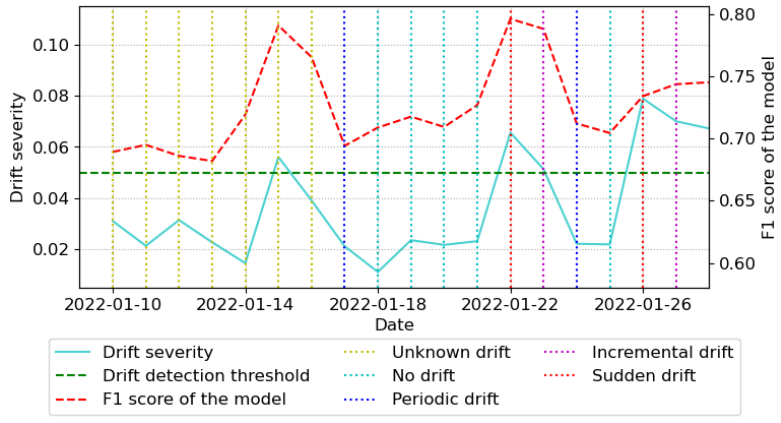

Figure 3.4: Workings of the drift analysis module using the TLS dataset. The model's performance and the global drift severity may be observed together with the predicted classes represented by the dotted lines of specific colours.

Chapter 4

Thesis outcomes

This thesis extensively studied various aspects of creating more robust models for network traffic classification. Many challenges making network traffic a particularly difficult domain to classify were described, and approaches to addressing those challenges were tried. Some challenges, such as biases present in the creation of the models, should be handled by a more thorough training process. One of the primary sources of model degradation over time is concept drift, which was addressed by the creation of a detector that can guide the retraining of the models to prevent the model's performance from deteriorating.

4.1 Best practices

Many experiments were performed to study the behaviour of network traffic on a variety of long-term real-world datasets. Commonly used feature vectors use features whose distributions largely vary over time, as the statistics used may correlate with the throughput of the network. Because of that, the same class can be observed to have different distributions during different states of the network. The paradigm of multiple recurring concepts was observed, with working week traffic having a different distribution than the weekend one. This periodic behaviour was observed on multiple datasets and feature vector choices.

Because of that, a bias may be introduced when capturing network traffic datasets. One should capture the dataset through longer time periods and avoid capturing different classes during different states of the network or different times. In that case, the model may overfit to differences between the time-specific behaviour instead of generally correct patterns. Another source of bias may be training the model on a small subset of data only, possibly also overfitting to the time-specific behaviour. It was showcased that a model trained this way managed to substantially degrade in a manner of days. In

some cases, the created models may be highly performing on minor experimental datasets, but may not be suitable for use for real-world traffic. Proper long-term testing and model monitoring is thus encouraged.

Concept drift represents the main source of model degradation. The distributions evolve over time and the trained data patterns may not be correct anymore. This presents a major challenge to model deployment as the model's performance decreases over time. It is highly advisable to use methods for model maintenance which monitor the model's performance and have the ability to retrain the models, such as the Active Learning Framework. The detection of concept drift represents a further increase in the quality of model maintenance as the retraining process may react to the observed changes in the distributions. The created concept drift detector may be used to enhance the existing model maintenance frameworks and the results of tests of its abilities may be seen in the following section.

4.2 Testing

The created drift detector was analysed on the QUIC and TLS datasets to provide proof of how it may aid model retraining and prevent the models from degrading because of concept drift. Experiments simulating model deployment and retraining guided by the drift detector's alarm signals were performed. The behaviour of the Active Learning Framework was simulated with a process corresponding to the strategy of random sampling being used. In both cases, the data from the first week was used to train the reference model. In the case of drift detection, the model's dataset is updated and the model is retrained. The current data subset is used to replace the corresponding number of oldest instances in the dataset, which was circa 30% of the dataset. The knowledge base of the model is present in this dataset, which simulates the way ALF would operate. The dataset also represents the historical sample supplied to the drift detector. XGBoost classifier is used as the model as it presents one of the model infrastructures currently often used. The feature vector supplied by CESNET DataZoo was used and the goal of the model is to classify all of the captured services, which is more than a hundred classes. The goal of the experiments is to compare the performance of the reference model with the performance of the model retrained after drift detection. The results of the experiment on the year-long TLS dataset may be observed in the table 4.1.

Table 4.1: Model performance, measured using F1 Score, on the TLS dataset when utilising model update guided by the drift detector. Validation dataset was a subset of the first week of the data, while the test was performed on the rest of the year-long dataset.

Validation F1	**Reference test F1**	**Retrained test F1**
0.725	0.602	0.748

It can be seen how the performance over the whole year-long dataset vastly improves. Surprisingly, the maintained model not only keeps its performance, but it even increases compared to the validation performance in the first week. The possible explanation of the observed phenomenon is that the concept drift led to the simplification of the problem by the differences between the classes being more recognisable or the distributions more stable. The visualisation present in Figure 4.1 may be used to judge how the model reacted to the dataset update. It shows when the drift detection happened, and the F1 scores of the reference and the maintained models may be compared to understand how the model degradation was prevented. In some cases, a single retraining was sufficient to keep the model performing for several months. In other, several retrainings were needed. This may be due to the incremental drift being present, where the change of concepts may take a longer time. The single retraining may only lead to the learning of the intermediate concept and another is needed when the new concept stabilises. The reference drift severity may be observed to understand how strongly the model would drift if it weren't updated. The periodic behaviour can also be observed, where the severity increases each weekend, sometimes getting close to the drift detection threshold.

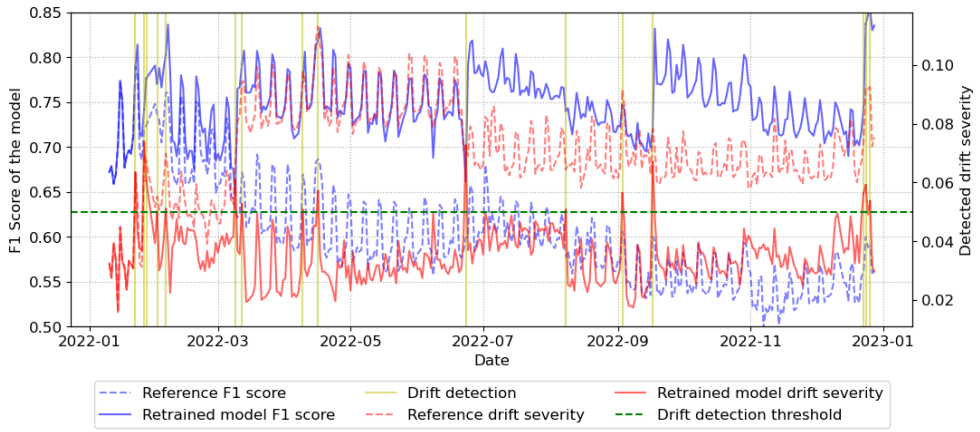

Figure 4.1: Simulated model maintenance on the TLS dataset guided by the drift detection. In the case of drift detection, visualised by a yellow vertical line, a model is retrained. The model performance and drift severities may be compared between the reference and the continuously retrained model.

During the analysis, a severe drift accompanied by a major change in most of the feature distributions was observed in March. This knowledge can be used for judging how the drift detection would react to this change. Figure 4.2 presents the view selecting only March to study the detection and dataset update corresponding to the known area of severe drift. It may be seen that the model degradation was successfully prevented, and the performance

drastically improved. This change seemed to happen under the paradigm of incremental drift, where a couple of days passed between the concepts fully changed. Because of that, the first detection happened at the start of the process change, and the second one retrained the model when the new concept stabilised. This simulation showcased how the model update guided by the drift detection may lead to a successful and long-lasting model performance on the year-long TLS dataset.

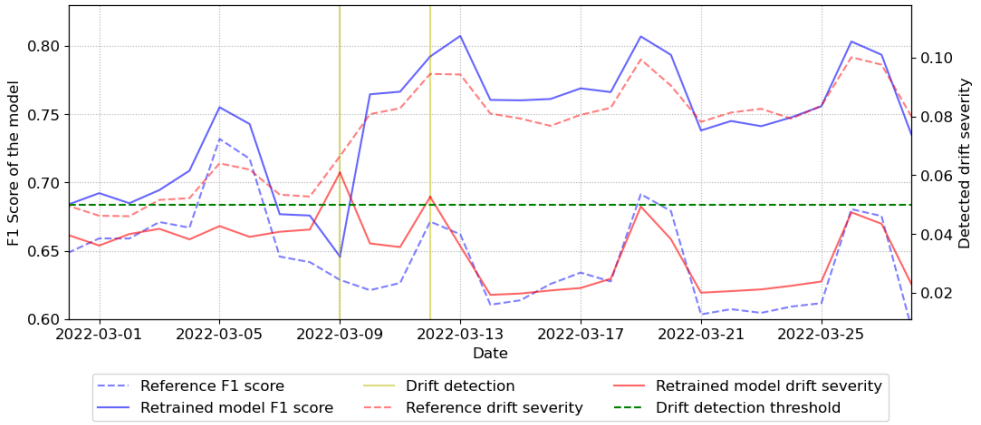

Figure 4.2: Simulated model maintenance on the March subset of the TLS dataset guided by the drift detection. In the case of drift detection, visualised by a yellow vertical line, a model is retrained. The model performance and drift severities may be compared between the reference and the continuously retrained model.

Compared to the year-long TLS dataset, CESNET QUIC was captured during four weeks. As with the previous experiment, the first week was used for training and validation, and the remaining three weeks for testing. As seen in table 4.2, this experiment provides another example of successful model retraining preventing model degradation, where the test performance remained in line with the validation performance. The development of drift severity and model performance can be observed in Figure 4.3. It can be seen that retraining was needed several times to adapt to the change of concepts.

Table 4.2: Model performance, measured using F1 Score, on the QUIC dataset when utilising model update guided by the drift detector. The validation dataset was a subset of the first week of the data, while the test was performed on the remaining three weeks.

Validation F1	**Reference test F1**	**Retrained test F1**
0.748	0.631	0.743

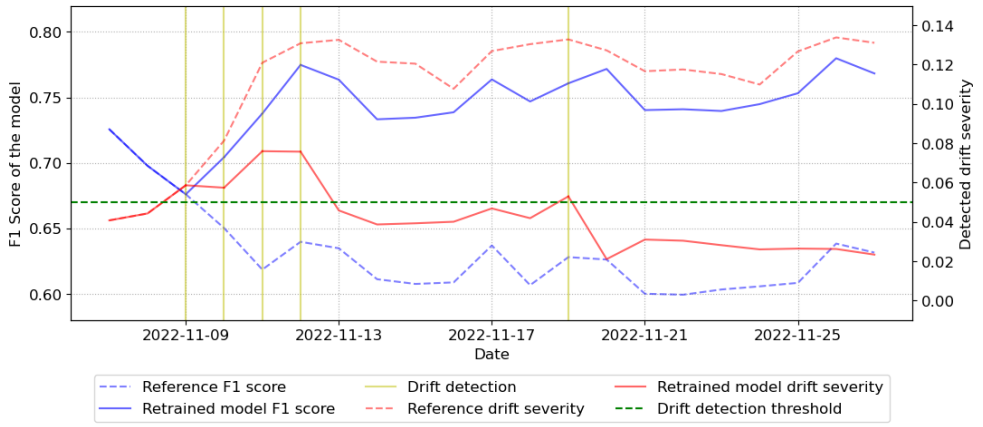

Figure 4.3: Simulated model maintenance on the QUIC dataset guided by the drift detection. In the case of drift detection, visualised by a yellow vertical line, a model is retrained. The model performance and drift severities may be compared between the reference and the continuously retrained model.

In both cases, model retraining guided by the drift detector resulted in vast improvements in model F1 scores. The capabilities of this method were showcased and further incorporation of it into Active Learning Framework is advised. On top of the drift detection itself, the drift detector exceeded the assigned requirements by implementing various other tools for concept drift analysis. The drift detector is implemented as a basis for a more complex modular detection framework instead of a simple prototype and the primary goal of this thesis can be considered more than successful.

4.3 Future research

While using concept drift detection to guide the retraining of ML models yielded massive performance improvements, there still exist unanswered questions about dealing with the periodic nature of network traffic. When working under the paradigm of multiple recurring concepts, simply retraining the models may not be enough to gain the best-performing model possible. Increasing the sensitivity of the tests isn't the answer, as then the models would simply learn and then forget the recurring concept of weekend traffic at the beginning and end of each weekend.

There exists the approach of concept matching, utilising multiple models and using the one representing the closest concept to current data for inference. This seems to be a promising technique to develop an even more robust classification infrastructure. To explore that idea, an experiment using two models was designed and tested on the TLS dataset. The traditional approach would train the model on the whole first week of data for proper generalisa-

tion. However, the behaviour of multiple recurring concepts was observed with weekend traffic differing from the one of a working week. Because of that, two models are trained on the subsets of the training dataset. One is trained on the days representing the working week, and the second is trained on weekend data. The experiment presumes that an advanced concept-matching framework would always choose the model trained on the closest concept for inference, resulting in a combined model with better performance. The results of the simulation of this method are showcased in Figure 4.4.

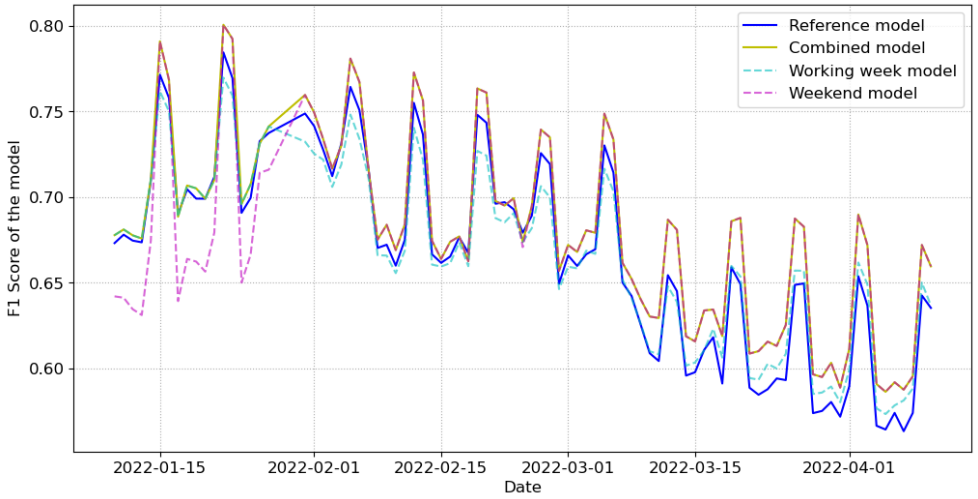

Figure 4.4: Experiment simulated theoretical concept matching frameworks. A combined model switching between two models trained on the subsets of the training dataset representing different concepts outperforms a referential model trained on the whole week.

The experiment simulated how the four different models would behave under long-term deployment. No dataset update was performed after training on the first week (or its subsets). Not only did this simulation serve as proof that one should experiment with methods of concept matching further, but it also provided unexpected observations of how drift affects the multiple concepts present. In the first couple of simulated deployment weeks, the behaviour was as expected. The working week model performed vastly better during the working week and the weekend model during the weekend. However, the effects of concept drift resulted in such degradation of the working week model that it underperformed and was beaten during the working week. The drift of the working week concept brought it closer to the concept of training weekend traffic than the one of training working week.

Outside of this observation, the results illustrate that this approach provides an increase in performance and should be further studied. For every day, choosing the model representing the closer concept for inference yielded

a better F1 score than using the reference model trained on the data from the whole week. The testing F1 scores can be observed in table 4.3. The combined model provided an increase of 0.015 of the average F1 score.

Table 4.3: Testing F1 scores of the models used for the concept matching simulation.

Reference	**Working week**	**Weekend**	**Combined**
0.667	0.666	0.675	0.682

While the previous concept matching simulation showcased that such methods would result in increased performance, the combined model still underperformed during the working week. The experiment presumed the existence of two recurring concepts – the working week and weekend traffic. However, as the features may develop with the different network load, there may be more concepts present or the working week off-peak hour traffic may be closer to the weekend one. The following experiment was designed to explore those ideas further. The created drift detector was used to analyse the development of traffic (using the Wasserstein distance test) through the day on two chosen days: D_0 – 1.1.2022, Saturday and national holiday; D_2 – 3.1.2022, Monday. Each time window was an hour long, with the first hour creating the historical time window. Every other hour represented the different current windows, and the drift severity between those samples was studied. As can be seen in the visualisation of the experiment results in Figure 4.5, the possible source of models generally underperforming on the working week traffic may be the existence of multiple concepts during a single day. This differs from the weekend sample, where every test lies under the drift detection threshold.

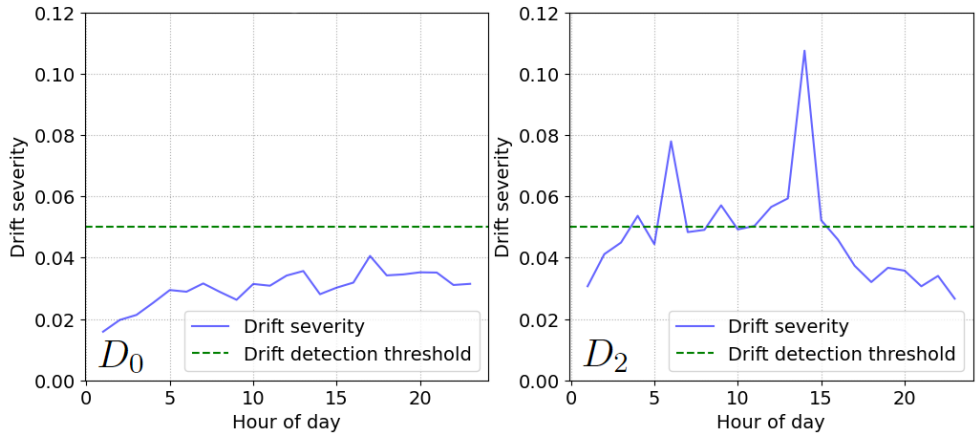

Figure 4.5: Comparison of distribution development through day on weekend and working week. Two days were chosen: D_0 – 1.1.2022, Saturday and national holiday; D_2 – 3.1.2022, Monday.

The methods of concept matching present a novel and promising approach that should be further studied as it may bring further increase in model performance. However, these experiments lie outside of the scope of this thesis and were done to demonstrate that these methods may deliver the most robust approach to network traffic classification. Further research into concept matching is advised as it was showcased how the traffic may not only drift because of a major process change, but also through the week and even through the day. Another possible solution is to research the use of other feature vectors than those currently used. The features are often correlated with throughput and are thus dependent on the state of the network. In an ideal world, the features would be stable and independent of the current load of the network. If such features exist, the distributions may develop differently than under the current paradigm of multiple recurring concepts, which would result in less complex classification tasks and more robust models.

Conclusion

The thesis delved into various aspects of creating more robust models for network traffic classification. Machine learning approaches are currently often utilised. However, there still exist the challenges of biases present in the model and concept drift. If these challenges are not addressed, the performance of the models may degrade over time. Biases should be handled by ensuring that no false data patterns are introduced during the process of data capturing and model creation. Concept drift makes the model underperform as the trained data patterns become obsolete because of the changes in the distribution. In those cases, models should be retrained, which can be done by utilising model maintenance frameworks, such as the Active Learning Framework. The created drift detector may then serve to enhance those frameworks by guiding the dataset update.

The main goal of this thesis is the creation of said concept drift detector. As it was tailor-made to the domain of network traffic, many experiments have been performed to analyse the domain behaviour first. It was discovered that distributions periodically develop, with working week traffic having a different distribution than the weekend one. The features used as a basis of ML models may correlate with the throughput and thus depend on the current load on the network. If drift is not addressed, the model may quickly degrade. An experiment of simulated model deployment showcased that a model trained only on weekend traffic from a single day may provide significantly worse predictions only a couple of days later (dropping from F1 score of 0.92 to 0.7). Other than this periodic behaviour, slow long-term decay and sudden severe change have been observed. If the models are not retrained after such severe process changes, they become ineffective, as the trained data patterns may not be useful anymore. It was showcased that network traffic indeed represents a complex domain, and such research was needed.

The following research experimented with existing concept drift detectors to study if the currently available methods were able to detect concept drift in samples associated with model underperforming. This would be the case

for the TLS dataset but not for the QUIC dataset. It was discovered that a couple of severely drifted features could be the main source of model degradation. This urged further research into improving the detection to handle such scenarios. It was discovered that classes may drift differently. The idea of detecting concept drift was then generalised for complex analysis of the change of the distribution, where various choices of time windows enabled different observations of how the distributions develop. This idea is addressed in the final drift detector by allowing the user to supply the time windows in a way suitable for the current experiment.

The created novel concept drift detector is available as a basis for a modular drift detection framework. The primary use case of the drift detector is incorporating it into the existing infrastructure of the Active Learning Framework but it may also be used for offline analysis of existing datasets. To tailor the detector to this use case, various analysis modules have been implemented. Detection is done on the whole sample, but then independently for each class to discover which classes are the most drifted. On top of that, a novel approach to drift understanding of classifying drift types and further explaining the change in data was implemented. To solve the issue of several drifted features being the cause of model degradation, a novel logic for drift detection was invented. Tests are run independently for each feature and the final drift severity is calculated as a weighted mean, using feature importances supplied by the ML model.

This approach was showcased to be highly effective in simulated model deployment and retraining guided by the detector. With this restraining, the model was prevented from degrading and in one case, the performance even improved compared to the validation dataset. Usage of such techniques is thus advised and the detector is prepared to be incorporated into model maintenance frameworks, such as ALF. There still exist areas of possible future research. As network traffic can be characterised by the paradigm of multiple recurring concepts, the model performance could be further increased by creating a complex concept-matching framework which would retrain multiple models and choose the one most suitable for the current state of the network. The search for the most stable and robust feature vectors is also advised.

Bibliography

1. PEŠEK, Jaroslav; SOUKUP, Dominik; ČEJKA, Tomáš. Active Learning Framework For Long-term Network Traffic Classification. In: *2023 IEEE 13th Annual Computing and Communication Workshop and Conference (CCWC)*. 2023, pp. 0893–0899. Available from DOI: 10.1109/CCWC57344.2023.10099065.
2. AZAB, Ahmad; KHASAWNEH, Mahmoud; ALRABAEE, Saed; CHOO, Kim-Kwang Raymond; SARSOUR, Maysa. Network traffic classification: Techniques, datasets, and challenges. *Digital Communications and Networks*. 2022. ISSN 2352-8648. Available from DOI: https://doi.org/10.1016/j.dcan.2022.09.009.
3. BAKNI, Michel. *Wikimedia - Data packet of IPv4*. [online]. [visited on 2023-12-18]. Available from: https://commons.wikimedia.org/wiki/File:IPv4_Packet-en.svg.
4. Internet Security Research Group. *2023 Annual report* [online]. [visited on 2024-01-08]. Available from: https://www.abetterinternet.org/documents/2023-ISRG-Annual-Report.pdf.
5. FUCHS, Christian. SOCIETAL AND IDEOLOGICAL IMPACTS OF DEEP PACKET INSPECTION INTERNET SURVEILLANCE. *Information, Communication & Society*. 2013, vol. 16, no. 8, pp. 1328–1359. Available from DOI: 10.1080/1369118X.2013.770544.
6. GitHub – ntop. *nDPI library* [online]. [visited on 2023-12-19]. Available from: https://github.com/ntop/nDPI/.
7. QUITTEK, Juergen; ZSEBY, Tanja; CLAISE, Benoit; ZANDER, Sebastian. *Rfc 3917: Requirements for ip flow information export (ipfix)*. RFC Editor, 2004.
8. TRAMMELL, B; BOSCHI, E. *RFC 5103: Bidirectional flow export using IP flow information export (IPFIX)*. RFC Editor, 2008.

9. AITKEN, P. *RFC 7011: Specification of the IP Flow Information Export (IPFIX) Protocol for the Exchange of Flow Information*. RFC Editor, 2013.

10. GitHub – Cisco. *Cisco Joy* [online]. [visited on 2023-12-19]. Available from: `https://github.com/cisco/joy`.

11. Progress. *Flowmon Probe* [online]. [visited on 2023-12-19]. Available from: `https://www.flowmon.com/en/products/appliances/probe`.

12. GitHub – Canadian Institute for Cybersecurity. *CICFlowMeter* [online]. [visited on 2023-12-19]. Available from: `https://github.com/ahlashkari/CICFlowMeter`.

13. GitHub – CESNET. *ipfixprobe - IPFIX flow exporter* [online]. [visited on 2023-12-18]. Available from: `https://github.com/CESNET/ipfixprobe`.

14. LUXEMBURK, Jan; HYNEK, Karel. DataZoo: Streamlining Traffic Classification Experiments. In: *Proceedings of the 2023 on Explainable and Safety Bounded, Fidelitous, Machine Learning for Networking*. 2023, pp. 3–7.

15. GitHub – CESNET. *CESNET DataZoo Features* [online]. [visited on 2023-12-19]. Available from: `https://cesnet.github.io/cesnet-datazoo/features/`.

16. LUXEMBURK, Jan; HYNEK, Karel; ČEJKA, Tomáš. Encrypted traffic classification: the QUIC case. In: *2023 7th Network Traffic Measurement and Analysis Conference (TMA)*. 2023, pp. 1–10. Available from DOI: `10.23919/TMA58422.2023.10199052`.

17. SHEARER, Colin. The CRISP-DM model: the new blueprint for data mining. *Journal of data warehousing*. 2000, vol. 5, no. 4, pp. 13–22.

18. DIETTERICH, Thomas G; KONG, Eun Bae. Machine learning bias, statistical bias, and statistical variance of decision tree algorithms. 1995.

19. FERNANDES DE MELLO, Rodrigo; ANTONELLI PONTI, Moacir. A Brief Review on Machine Learning. In: *Machine Learning: A Practical Approach on the Statistical Learning Theory*. Springer International Publishing, 2018, pp. 1–74. ISBN 978-3-319-94989-5. Available from DOI: `10.1007/978-3-319-94989-5_1`.

20. HELLSTRÖM, Thomas; DIGNUM, Virginia; BENSCH, Suna. Bias in Machine Learning–What is it Good for? *arXiv preprint arXiv:2004.00686*. 2020.

21. MEHRABI, Ninareh; MORSTATTER, Fred; SAXENA, Nripsuta; LERMAN, Kristina; GALSTYAN, Aram. A survey on bias and fairness in machine learning. *ACM computing surveys (CSUR)*. 2021, vol. 54, no. 6, pp. 1–35.

22. SCHLIMMER, Jeffrey C; GRANGER, Richard H. Incremental learning from noisy data. *Machine learning*. 1986, vol. 1, pp. 317–354.

23. LOSING, Viktor; HAMMER, Barbara; WERSING, Heiko. KNN Classifier with Self Adjusting Memory for Heterogeneous Concept Drift. In: *2016 IEEE 16th International Conference on Data Mining (ICDM)*. 2016, pp. 291–300. Available from DOI: 10.1109/ICDM.2016.0040.

24. GAMA, João; ŽLIOBAIT, Indr; BIFET, Albert; PECHENIZKIY, Mykola; BOUCHACHIA, Abdelhamid. A Survey on Concept Drift Adaptation. 2014, vol. 46, no. 4. ISSN 0360-0300. Available from DOI: 10.1145/2523813.

25. LU, Jie; LIU, Anjin; DONG, Fan; GU, Feng; GAMA, João; ZHANG, Guangquan. Learning under Concept Drift: A Review. *IEEE Transactions on Knowledge and Data Engineering*. 2019, vol. 31, no. 12, pp. 2346–2363. Available from DOI: 10.1109/TKDE.2018.2876857.

26. WEBB, Geoffrey I; HYDE, Roy; CAO, Hong; NGUYEN, Hai Long; PETITJEAN, Francois. Characterizing concept drift. *Data Mining and Knowledge Discovery*. 2016, vol. 30, no. 4, pp. 964–994.

27. GOLDENBERG, Igor; WEBB, Geoffrey I. Survey of distance measures for quantifying concept drift and shift in numeric data. *Knowledge and Information Systems*. 2019, vol. 60, no. 2, pp. 591–615.

28. MORENO-TORRES, Jose G; RAEDER, Troy; ALAIZ-RODRÍGUEZ, Rocío; CHAWLA, Nitesh V; HERRERA, Francisco. A unifying view on dataset shift in classification. *Pattern recognition*. 2012, vol. 45, no. 1, pp. 521–530.

29. STORKEY, Amos et al. When training and test sets are different: characterizing learning transfer. *Dataset shift in machine learning*. 2009, vol. 30, no. 3-28, p. 6.

30. GAMA, Joao; MEDAS, Pedro; CASTILLO, Gladys; RODRIGUES, Pedro. Learning with drift detection. In: *Advances in Artificial Intelligence–SBIA 2004: 17th Brazilian Symposium on Artificial Intelligence, Sao Luis, Maranhao, Brazil, September 29-Ocotber 1, 2004. Proceedings 17*. Springer, 2004, pp. 286–295.

31. DOS REIS, Denis Moreira; FLACH, Peter; MATWIN, Stan; BATISTA, Gustavo. Fast unsupervised online drift detection using incremental kolmogorov-smirnov test. In: *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*. 2016, pp. 1545–1554.

32. CONNOR, Richard; CARDILLO, Franco Alberto; MOSS, Robert; RABITTI, Fausto. Evaluation of Jensen-Shannon distance over sparse data. In: *Similarity Search and Applications: 6th International Conference, SISAP 2013, A Coruña, Spain, October 2-4, 2013, Proceedings 6*. Springer, 2013, pp. 163–168.

33. FILIPPOVA, Olga. *Which test is the best? we compared 5 methods to detect data drift on large datasets* [online]. [visited on 2023-12-04]. Available from: `https://www.evidentlyai.com/blog/data-drift-detection-large-datasets`.

34. SETTLES, Burr. *Active Learning Literature Survey*. 2009. Computer Sciences Technical Report, 1648. University of Wisconsin–Madison.

35. EL-HASNONY, Ibrahim M; ELZEKI, Omar M; ALSHEHRI, Ali; SALEM, Hanaa. Multi-label active learning-based machine learning model for heart disease prediction. *Sensors*. 2022, vol. 22, no. 3, p. 1184.

36. KENJIRO CHO, Koushirou Mitsuya; KATO, Akira. *Traffic Data Repository at the WIDE Project* [online]. [visited on 2023-12-20]. Available from: `https://mawi.wide.ad.jp/mawi/`.

37. LUXEMBURK, Jan; ČEJKA, Tomáš. Fine-grained TLS services classification with reject option. *Computer Networks*. 2023, vol. 220, p. 109467.

38. UHŘÍČEK., Daniel. *nemea-fet - Feature Exploration Toolkit* [online]. [visited on 2023-12-20]. Available from: `https://docs.danieluhricek.cz/fet/`.

39. GitHub – Evidently AI. *Evidently* [online]. [visited on 2023-12-26]. Available from: `https://github.com/evidentlyai/evidently`.

40. LUXEMBURK, Jan; HYNEK, Karel; ČEJKA, Tomáš; LUKAČOVIČ, Andrej; ŠIŠKA, Pavel. CESNET-QUIC22: A large one-month QUIC network traffic dataset from backbone lines. *Data in Brief*. 2023, vol. 46, p. 108888.

Appendix A

Acronyms

AL Active learning

DPI Deep packet inspection

HTTP Hypertext Transfer Protocol

HTTPS Hypertext Transfer Protocol Secure

IANA Internet Assigned Numbers Authority

IETF Internet Engineering Task Force

IP Internet Protocol

IPFIX Internet Protocol Flow Information Export

ISRG Internet Security Research Group

KS Kolmogorov–Smirnov

ML Machine learning

TCP Transmission Control Protocol

SMTP Simple Mail Transfer Protocol

PCA Principal Component Analysis

PPI Per-packet information

Appendix B

Included contents

- `README.md` ..... Description of the included contents
- `analysis` ..... Jupyter notebooks used during the analysis
- `detector` ..... Implementation of the created concept drift detector
  - `analyser.py` ..... Implementation of the `Analyser` module
  - `detector.py` ..... Implementation of the main detection class
  - `logger.py` ..... Implementation of the `Logger` module
  - `test.py` ..... Implementation of the various drift detection tests
- `testing` ..... Jupyter notebooks used for testing the detector
  - `ALF_simulation_QUIC.ipynb` . Test of detector guided dataset update
  - `ALF_simulation_TLS.ipynb` .. Test of detector guided dataset update
- `thesis` ..... Thesis text directory
  - `thesis_Jancicka.pdf` ..... PDF version of the thesis
  - `thesis_source` ..... LaTeX source codes of the thesis
- `usage_example` ..... Jupyter notebooks showcasing the detector usage
  - `showcase_quic_analysis.ipynb` ..... Example of the detector usage
  - `showcase_tls_long-term.ipynb` ..... Example of the detector usage

Appendix C

Usage example

The following example describes how to use the drift detector to perform one round of detection to compare two time windows of one dataset to analyse how it evolves. Cesnet-QUIC dataset was used as an example, available through their DataZoo platform, which is used as follows:

```
from cesnet_datazoo.datasets import CESNET_QUIC22
from cesnet_datazoo.config import DatasetConfig

data = CESNET_QUIC22("./datasets/QUIC/", size="XS")

#Get the first week of data
dataset_config = DatasetConfig(dataset=data,
                  train_period="W-2022-44")
data.set_dataset_config_and_initialize(dataset_config)
data_ref = data.get_train_df()

#Get the first week of data
dataset_config = DatasetConfig(dataset=data,
                  train_period="W-2022-45")
data.set_dataset_config_and_initialize(dataset_config)
data_curr = data.get_train_df()
```

The process of initialising drift detector is done by supplying it with configuration for the global detection test. Additionaly, one may supply the configuration for the class-based test or use the `Logger` or `Analyser` modules. Only class-based test is showcased in this example, where one may use different tests for the different detection modules.

```
from detector.detector import DriftDetector, Config
from detector.test import KSTest, WassersteinTest

#Specify which features to run the test on
#Chosen test aren't suitable to work with categorical features
feat_names = ['BYTES','BYTES_REV','PACKETS',...,'IPT_BIN8_REV']

#Define the configuration of the global drift detection
```

```
global_config = Config(
    chosen_features = feat_names,
    drift_test=WassersteinTest()
)

#Define the configuration of the class-based drift detection
class_config = Config(
    chosen_features = feat_names,
    drift_test=KSTest(),
    class_name="APP"
)
#Initialise the detector
detector = DriftDetector(global_config, class_config)
```

The single round of drift detection is run as follows:

```
detector.detect(data_ref,data_curr)
```

No drift was discovered between those two chosen samples. Some of the features could still be detected as drifted. They may be returned by calling:

```
detector.get_drifted_features()
```

Two features were detected as drifted and their severities may be observed:

Feature	Drift severity
PSIZE_BIN5_REV	0.248
PSIZE_BIN4_REV	0.127

Each global statistic measured for the current round of detection can be returned in the following way. If one used the `Analyser` module, the inferred drift type would also be returned. This represents the same statistics that would be stored by the `Logger` module.

```
detector.get_drift_statistics()
```

Drift detection	Drift severity	Drifted feature share
False	0.03	0.05

The class-based tests are run separately and their result may be obtained in this way. KS test was used for the class test, so severity represents the p-values of test based around distributions being the same.

```
detector.get_class_drift()
```

Class	Drift detection	Drift severity	Drifted feature share
99	True	0.068	0.875
100	True	0.072	0.875
97	True	0.089	0.825
...	...	...	...