UPTEC F 23054

Examensarbete 30 hp

Augusti 2023

Machine learning for detecting financial crime from transactional behaviour

Markus Englund

Civilingenjörsprogrammet i teknisk fysik

UPPSALA
UNIVERSITET

Machine learning for detecting financial crime from transactional behaviour

Markus Englund

Abstract

Banks and other financial institutions are to a certain extent obligated to ensure that their services are not utilized for any type of financial crime. This thesis investigates the possibility of analyzing bank customers' transactional behaviour with machine learning to detect if they are involved in financial crime. The purpose of this is to see if a new approach to processing and analyzing transaction data could make financial crime detection more accurate and efficient. Transactions of a customer over a time period are processed to form multivariate time series. These time series are then used as input to different machine learning models for time series classification. The best method involves a transform called Random Convolutional Kernel Transform that extracts features from the time series. These features are then used as input to a logistic regression model that generates probabilities of the different class labels. This method achieves a ROC AUC-score of 0.856 when classifying customers as being involved in financial crime or not. The results indicate that the time series models detect patterns in transaction data that connect customers to financial crime which previously investigated methods have not been able to find.

Teknisk-naturvetenskapliga fakulteten

Uppsala universitet, Utgivningsort Uppsala/Visby

Handledare: Johan Abrahamsson Ämnesgranskare: Isaac Skog

Examinator: Tomas Nyberg

Populärvetenskaplig sammanfattning

Banker och andra finansiella institutioner har ett ansvar att se till att deras tjänster inte utnyttjas för finansiell brottslighet. Några exempel på olika typer av finansiella brott är penningtvätt, mutbrott och förskingring. För banker som hanterar flera miljoner transaktioner per dag kan det vara ett omfattande arbete att upptäcka kunder som är involverade i finansiell brottslighet. Den konventionella metoden för att göra detta är att använda regler som är kopplade till faktorer som ofta är förknippade med finansiell brottslighet. När en regel bryts så skapas ett larm och en utredning påbörjas för den berörda kunden. Om utredningen visar att kunden är inblandad i finansiell brottslighet, överlämnas ärendet till myndigheter.

För att göra processen för att upptäcka finansiell brottslighet mer effektiv har under de senaste åren olika implementeringar som utnyttjar maskininlärning undersökts. Resultat från studier på området indikerar att de bästa maskininlärningsmodellerna för att detektera finansiell brottslighet är så kallade trädbaserade modeller. En nackdel med dessa modeller är att mycket information om transaktionsdata går förlorad på grund av formatet som de tar emot indata. Denna uppsats undersöker därför ett nytt sätt att formattera och analysera data från transaktioner som utförts av kunder för att prediktera finansiell brottslighet. Transaktioner av en specifik kund under en tidsperiod behandlas så att de utgör en tidsserie, där varje parameter motsvarar en viss typ av transaktion. Maskininlärningsmodeller för tidsserieklassificering används sedan för att prediktera vilka kunder som är involverade i finansiell brottslighet.

Detta projekt har genomförts i samarbete med banken Nordea, och en av deras kunddatabaser som innehåller personer som har utretts för finansiell brottslighet har använts för att träna och testa olika modeller. Denna databas innehåller dels information om transaktioner som kunder har utfört, men även annan övrig information såsom saldon för olika konton, ålder etc. Förmågan av olika metoder att skilja mellan kunder som är involverade i finansiellt brottslighet bedöms framförallt utifrån ett mått som kallas ROC AUC-värde. Den metod som generellt sett presterade bäst för tidsserieklassificering involverade användningen av en transform kallad Random Convolutional Kernel Transform. Denna transform extraherar parametrar från tidsserien som sedan används i en Logistic Regression-modell. ROC AUC-värdet för denna metod uppmättes till 0.856. För att jämföra metoderna som involverade tidsserieklassificering med trädbaserade modeller så implementeras även en trädbaserad modell som kallas Extreme Gradient Boosting. Denna modell uppnådde ett ROC AUC-värde på 0.878. Den trädbaserade modellen presterar alltså bättre än metoderna som involverar tidsserieklassificering. Men två av modellerna för tidsserieklassificering presterar fortfarande förhållandevis bra då de bara utnyttjar transaktionsdata, i jämförelse med den trädbaserade modellen som utnyttjar all tillgänglig data. Resultaten indikerar även att de två olika typerna av metoder hittar olika mönster som kopplar kunder till finansiell brottslighet. En slutsats är att tidsseriemodellerna presterar bra i att prediktera finansiell brottslighet utifrån transaktionsdata, men en metod behövs för att kombinera dessa modeller med en annan modell som utnyttjar all den tillgängliga datan.

Acknowledgements

I want to thank my supervisor Johan Abrahamsson for the opportunity to write this thesis and his support throughout the project. I would also like to thank my subject reader Isaac Skog for his input on the scientific aspects of the project and for his helpful feedback on the report.

Table of content

List of acronyms and abbreviations

1-NN	One Nearest Neighbour
AML	Anti Money Laundering
AUC	Area Under the Curve
DTW	Dynamic Time Warping
FCN	Fully Convolutional Neural Network
GRU	Gated Recurrent Units
KYC	Know Your Customer
LSTM	Long Short-Term Memory
PPV	Proportion of Positive Values
RAM	Random-Access Memory
RNN	Recurrent Neural Network
ROC	Receiver Operating Characteristics
ROCKET	Random Convolutional Kernel Transform
TSC	Time Series Classification
XGBoost	Extreme Gradient Boosting

1 Introduction

Banks and other financial institutions are due to the nature of their services vulnerable to being exploited by criminal ventures. For this reason, they are obligated by law to take certain measures to prevent their products from being utilized for different types of financial crime. For banks carrying out millions of transactions per day, the process of finding and investigating customers who are potentially involved in some type of financial crime can be a challenging task. To make the process of detecting customers involved in illicit activities more effective and accurate, different implementations using machine learning have been investigated. This thesis explores the possibility of analyzing customers' transactional behavior with machine learning to detect if they are involved in financial crime. The project is made in collaboration with the Nordic bank Nordea. To investigate the performance of different machine learning models, data from Swedish customers of Nordea is used.

1.1 Background

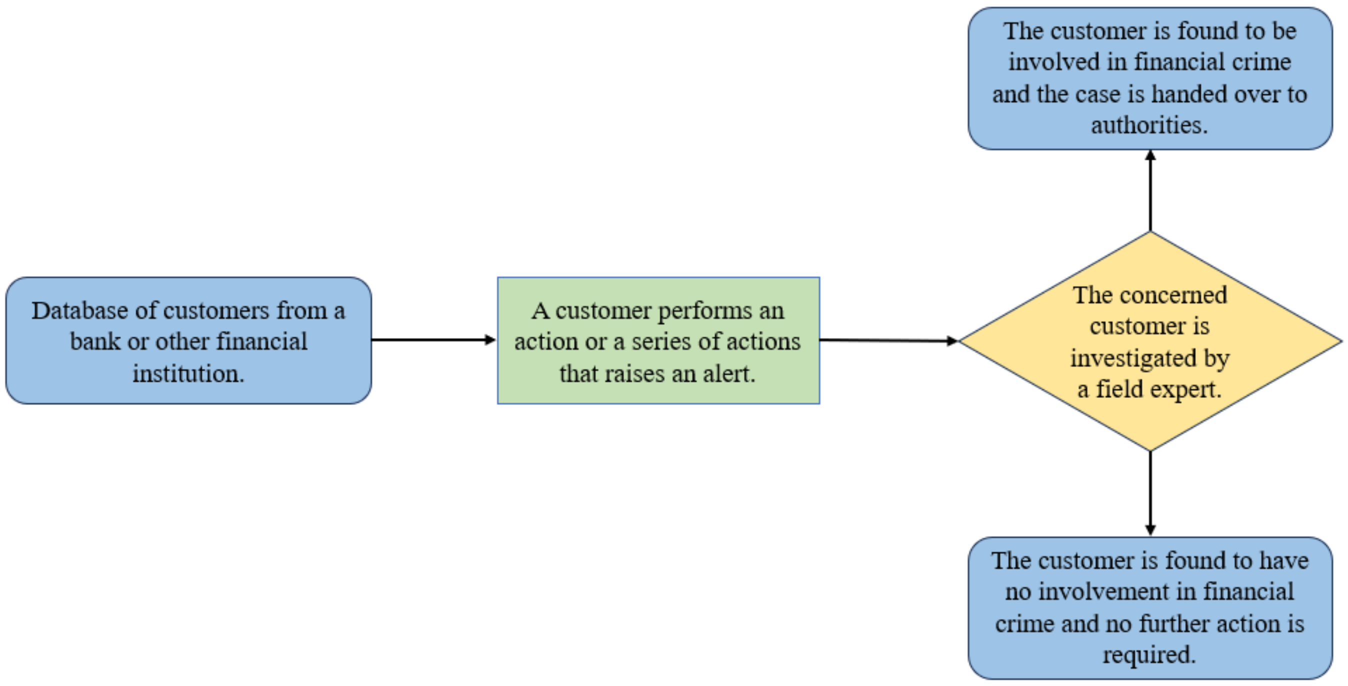
The term financial crime encompasses many types of illicit activities including money laundering, tax evasion, bribery, and terrorist financing [1]. Financial crime does not pose a direct danger to individuals but can have far-reaching consequences on societies and organizations in the long term. Money laundering in particular, which means the process of disguising an illegal source of money and replacing it with a legitimate one, enables the profitability of other violent crimes. When profits have been made from crimes such as illegal arms sales, drug trafficking, or prostitution, these earnings need to somehow be introduced into the financial system to be legitimized [2]. United Nations Office on Drugs and Crime estimated that the amount of money being laundered through the financial system in 2009 was equal to 1.6 trillion US dollars or 2.7 % of global GDP [3].

Banks and other financial institutions are to a certain extent obligated to ensure that their services are not used in any criminal activity. If banks do not make enough efforts to ensure that their services are not utilized for financial crimes, they can receive regulatory fines or even lose their banking license. Additionally, banks being involved in a financial crime can also lead to negative publicity causing reputational damage and customer loss [1]. In 2020 the Swedish Financial Supervisory Authority decided that Swedbank would have to pay 4 billion Swedish crowns in sanction fees for not following regulations regarding measures against money laundering [4]. Clifford Chance's report of investigation Swedbank shows that between 2015 and 2019 there had been an inflow of 75.4 billion Euros to accounts belonging to Anti Money Laundering (AML) Risk Identified Customers in subsidiary banks to Swedbank in the Baltic states [5]. There was also a consideration for removing the banking permit of Swedbank, but the bank was due to its efforts of improving its work against money laundering, in the end, let off with a warning [4].

Clearly, it's in the interest of banks to oversee the activity of their clients to ensure that their services are not being exploited by criminal ventures. The conventional method of finding clients who are involved in financial crime is to have a set of rules, which if broken, sends an alert indicating that the client should be investigated [6]. A rule could, for example, be that if a client withdraws a certain amount of cash during a time period, an alert should be generated. When a rule has been broken and an alarm has been raised, an investigation is initiated regarding the concerned client. If the client is found to be involved in criminal activity, the case is then handed over to authorities for further review and possible incarceration. Figure 1 shows a flowchart demonstrating the typical process of detecting financial crime in banks or other financial institutions. In recent times, approaches involving machine learning have been investigated to either improve the process of finding customers who should be investigated or to facilitate the investigation of customers found by the rule-based method.

Machine learning is the process of using a model to gain insight into the patterns of data and then use this information to make predictions for new, unseen data points. The data points that the model utilizes for learning the patterns of a dataset are often referred to as the training set. In supervised machine learning each data point has both an input and an output. The goal of the machine learning model is then to find the patterns in the input that defines the output. The output can either be a continuous numerical value or a set of discrete values. When the output is a continuous numerical value, the problem can be referred to as a regression problem. If the possible range of outputs is a set of discrete labels, the problem can be referred to as a classification problem. The goal of this project is to classify data examples into two different classes is thus a classification problem. To evaluate a model's ability to capture the patterns in the data a test set is used. This is a set of data points that are excluded from the training set. The input of the test set is fed to the model and the output of the model is then compared to the true labels of the test set [7].

The type of input that will be primarily used as input to the machine learning models in this project is time series. A time series refers to a sequence of data points collected or recorded over successive



Database of customers from a bank or other financial institution.
A customer performs an action or a series of actions that raises an alert.
The concerned customer is investigated by a field expert.
The customer is found to be involved in financial crime and the case is handed over to authorities.
The customer is found to have no involvement in financial crime and no further action is required.

Figure 1: A flowchart demonstrating the typical process of detecting financial crime in banks or other financial institutions.

time intervals. From a mathematical perspective, a time series can be seen as a stochastic process observed at discrete time points [8]. There are two different types of time series, univariate and multivariate. A univariate time series consists of a single value associated with each time step, whereas in a multivariate time series, multiple values correspond to each time step. All of the time series used in this project are multivariate. Since this project's objective is to classify different time series, a type of machine learning model called the time series classification (TSC) model will be utilized.

TSC models find relevance in a broad array of applications, for example in healthcare, where it helps in interpreting various types of medical data that are recorded over time, such as heart rate and brain wave patterns. It is also used in finance for detecting market trends, in environmental science for climate pattern recognition, and in many other fields [9]. Several different types of TSC models have been presented, including distance-based, kernel methods, and dictionary-based methods [10]. Recently, deep learning methodologies have shown promising results, offering advanced techniques for feature learning and sequence modeling. Despite the advances, the TSC field still facilitates an ongoing exploration of innovative methods and strategies to improve prediction performance across a wide range of application domains [9].

1.2 Related work

When examining the literature on the subject one finds that there have been several studies on implementing machine learning algorithms for detecting financial crime. Since the datasets needed for conducting such research contains sensitive information, the amount of public datasets is very limited, and different studies use different sources of information. Different countries also have different laws and procedures regarding financial crime, which also has implications for the methods for detecting them. For this reason, it's hard to compare the results of different studies and determine which ones have found the best methods. However when reading about different studies one gets an overview of what methods seems to perform the best in general and what performances have been achieved.

One study on the subject is done by Raiter (2021), who due to the lack of available data, utilizes a synthetic dataset of transactions. He uses this to compare different machine learning algorithms for detecting money laundering. The best-performing algorithm is found to be the Random Forest model which has an accuracy of 0.99 and an F1-score of 0.36 [11]. In the last few years, the use of Bitcoin and other cryptocurrencies has created new challenges in the field of AML. The anonymity and the new technologies involved in cryptocurrency trading have made traditional methods of detecting financial crime ineffective. Alotibi et. al (2022) investigates the possibility of using machine learning and deep learning to detect money laundering activities using data of cryptocurrency transactions [12]. Random Forest is once again found to have the best performance.

Another method for detecting clients involved in money laundering is proposed by Tertychnyi et. al (2020) who investigates a two-layered approach utilizing two different machine learning models. The aim of their implementation is to find clients involved in money laundering from a bank's database. A logistic regression model is first used to remove clearly non-illicit clients and then an extreme gradient boosting (XGBoost) model is used on the remaining samples to find customers with the highest risk. The purpose of the first layer is to reduce the size of the dataset while removing as few true positives as possible. This facilitates more complex methods in the second layer as well as reducing the issue of class imbalance [13].

Ingemann Tuffveson Jensen and Iosifidis (2023) investigate an implementation using deep learning to raise and qualify anti-money laundering alarms at the Danish bank Spar Nord Bank. The input data to the model consists of transactions in the form of a multivariate time series and client data in a tabular arrangement. The architecture of the models involves an embedding that maps categorical features to a latent representation in the form of vectors. Three different types of processing blocks, long short-term memory cells (LSTM), gated recurrent units (GRU), and Transformer encoder layers, are used to create a latent vector representation from the transaction sequences. This vector is then processed together with the client features in two feedforward layers and a sigmoid activation function to generate the output value. From alarms made by the rule-based system, the model was able to reduce the number of false positives by 33.3 % while retaining 98.8 % of true positives. The best-performing model was a two-layer bidirectional GRU with self-attention. All of the deep learning models utilizing time series data had better performance in terms of Receiver Operating Characteristics (ROC) Area Under the Curve (AUC) score than the baseline models, which included four different tree-based models [14].

1.3 Purpose

One of the drawbacks of the previously mentioned rule-based method is that it leads to a large proportion of false positives, meaning alerts that incorrectly classify a client as being involved in financial crime. The rate of false positives has been reported to be as high as 98 % in some cases. This leads to investigators spending much of their time reviewing clients who are not criminals. Not only is this an inefficient use of resources, but it can also have the effect of diverting attention from high-risk clients [6]. Furthermore, some criminal activities can involve a series of transactions over a longer period of time and be too complex for a rule-based method to detect it. Criminals continually update their strategies to remain undetected and the methods for finding them hence always need to improve to be effective [1]. To detect clients who exhibit more complex irregular behaviours different types of implementations using machine learning have been explored.

Machine learning could be used in multiple different ways to make the process of detecting financial crime. Firstly they could be used to create alerts instead of the rule-based method. This problem would include an unlabeled dataset since the investigations that decide if a client is involved in financial crime are only carried through for clients who have been alerted by the rule-based method. Since no labels would be provided for clients involved in financial crime in the dataset, this problem would require a solution involving an unsupervised machine learning implementation. This project will instead focus on a supervised machine learning problem. If one considers a dataset only including alerts generated by the rule-based method, it would be a suitable dataset for a supervised machine learning problem since all of the data examples are labeled. The goal of the machine learning model is then to qualify each alert, meaning to classify the concerned client of an alert as being involved in financial crime or not. If the performance of the machine learning model is good enough, one could consider the possibility of replacing investigators with the machine learning model. If the performance is not good enough, only alerts that are given the lowest score by the model could be disqualified. The rest would then be reviewed by investigators. This would lessen the work burden of investigators and save resources for the bank. A model trained on alerts could also be used to classify customers not detected by the rule-based method. However, due to limited resources, no new investigations will be made based on the findings of the models evaluated in this project. Therefore it will not be possible to know if the models would find clients involved in criminal activity who were not found by the rule-based method.

In general, the literature seems to suggest that tree-based models are the best for detecting financial crime. However, in tree-based models, each input is in the form of a one-dimensional array where each entry represents a feature. This means that they are not able to utilize time series data in an effective way to make predictions. A potential approach to consider would be implementing a one-hot encoding based on transaction type and day or time period, however, this would result in a very sparse representation and the model would likely not be able to capture any meaningful temporal patterns. Thus, tree-based methods are limited to using aggregate statistics related to transactions. To be able to utilize more granular transaction data, other types of models which are designed to use time series data as an input needs to be used. Currently, the only study which investigates models utilizing transaction data in a more granular arrangement is done by Ingemann Tuffveson Jensen and Iosifidis [14]. Their findings suggest that these models outperform tree-based models [14]. Clearly, this seems like an approach worth further investigation. Thus, the aim of this project will be to explore how time series models perform in classifying transaction data from alerts and how this can be used to improve financial crime detection.

1.4 Research questions

- How do machine learning models using time series data perform compared to tree-based models in detecting financial crime?
- What performance can be achieved with a machine learning model using time series data based on cases reported by rule-based methods and then identified by experts?
- Do tree-based models and time series models find different patterns in the data which connect customers to financial crime?
- Are deep learning algorithms suitable for classifying the time series data from transactions or do less complex machine learning models perform better?

2 Problem formulation

Two different types of models will be used in this project, TSC models and tree-based models. One difference between these two types of models is the format of the input data. Each input sample to the tree-based model is in the format of an array which can be written as $\boldsymbol{x_c} \in \mathbb{R}^d$ where d is the number of features. In this project, $\boldsymbol{x_c}$ contains both nominal and ratio data. The input to the TSC models is a multivariate time series. A time series can be seen as a stochastic process that is observed at discrete times t during a time interval $(0, T)$. Each observation of the time series can be written as $\boldsymbol{x_t}^{(\tau)} \in \mathbb{R}^q$ where q is the number of features of the time series and τ is a point in time. The time series used in this project has a fixed sampling frequency, meaning that the time between each observation is constant. The multivariate time series can be represented by a matrix $\boldsymbol{X_t} \in \mathbb{R}^{q \times l}$ where l is the number of observations or the length of the time series. The features in the considered time series only contain ratio data.

The following concepts will apply to both the TSC models and the tree-based model. The goal of the machine learning models in this project is to classify if the concerned client of an alert is involved in financial crime or not. This means learning a binary classification model given a training set $\mathcal{T} = \{\boldsymbol{x_i}, y_i\}_{i=1}^{n}$, where $\boldsymbol{x_i}$ can represent a multivariate time series or an array depending on the type of model. The class label y_i which is being predicted can be seen as a random variable conditioned on the input variable $\boldsymbol{x_i}$. In other words, the classifier should as accurately as possible, model the probability $p(y = Y|\boldsymbol{x_i})$ where $Y = \{-1, 1\}$, indicating the two different class labels. The two different classes labels represent the two outcomes of the classification problem as

$$Y = \begin{cases} 1: & \text{The concerned client of the alert is involved in financial crime} \\ -1: & \text{The concerned client of the alert does not exhibit any illicit behaviour.} \end{cases} \tag{1}$$

The two probabilities being modeled are $p(y = 1|\boldsymbol{x_i})$ indicating that a client is involved in financial crime and $p(y = -1|\boldsymbol{x_i})$ indicating that the client does not exhibit any illicit behaviour. If the probability $p(y = 1|\boldsymbol{x_i})$ is modelled by the function $g(\boldsymbol{x_i})$ the probability of the other label, $p(y = -1|\boldsymbol{x_i})$, is modelled by $1 - g(\boldsymbol{x_i})$ [7]. This can be deduced since the probabilities of the two class labels have to add up to one. However, so far only the probabilities of the different class labels have been derived. To actually classify data examples to one of the two labels, a decision needs to be made based on these probabilities. To classify a data example to one of the two classes one has to decide on a classification threshold r. The classification is then made according to

$$\hat{y}(\boldsymbol{x_i}) = \begin{cases} 1 & \text{if } g(\boldsymbol{x_i}) > r \\ -1 & \text{if } g(\boldsymbol{x_i}) \leq r \end{cases} \tag{2}$$

where $\hat{y}(\boldsymbol{x_i})$ is the predicted class label of a data example $\boldsymbol{x_i}$. The choice of the classification threshold depends on the circumstances of the problem at hand. If the goal is to achieve the highest possible accuracy, a classification threshold of $r = 0.5$ will generally achieve the best result [7]. However, in some problems, one class can be more important to correctly classify than the other. This can be referred to as an asymmetric problem. The problem can also be imbalanced, which means that there is a disparity between how often the different classes occur in a dataset. In the case of an asymmetric or imbalanced problem, the classification threshold r can be adjusted so that the results better reflect the aim of the model.

In this project, both parametric and non-parametric machine learning models will be used to classify data points. The method for finding the function g differs between these two different types of models. Contrary to what the name might suggest, non-parametric modeling does not mean that no parameters are learned from the data. A machine learning model being non-parametric instead means that the number of parameters for a model is not fixed, but depends on the training data set. This means that the same model can learn a varying number of parameters depending on which training set is used. This allows for great flexibility and enables non-parametric models to capture complex patterns in the data. This high flexibility aspect of non-parametric models means that this type of model is a good choice for problems where little is known about the input-output relationship of the data. In non-parametric models the function g which models the probability of a class label is unknown. This lack of insight into the relationship between the output and the input variables of the model leads to a generally low interpretability of results from non-parametric models [15].

The very flexible aspect of non-parametric models makes it hard to make any general outline of how a model is composed. Parametric models follow a more general framework which will be presented in the following paragraphs. In parametric modeling, the goal is to find a function f_θ which describes the input-output relationship of the problem as accurately as possible. To map this nonlinear function f_θ to a probability between 0 and 1 a Sigmoid function is used in the following way

$$g(\boldsymbol{x_i}) = \frac{e^{f_\theta(\boldsymbol{x_i})}}{1 + e^{f_\theta(\boldsymbol{x_i})}}. \tag{3}$$

The function f_θ can be constructed in many different ways and also contains hyperparameters that also need to be tuned in order to obtain the best results. Throughout this project, different functions f_θ will be tested and evaluated on their performance in classifying customers as being involved in financial crime or not. But in order to make a good classifier, the parameters $\boldsymbol{\theta}$ in f_θ also need to be determined. So the objective is to find both the best type of function for f_θ as well as the best parameters $\boldsymbol{\theta}$ for that function. In other words, for each model which is investigated for describing the input-output relationship, there is a parametric family of functions,

$$\{f_\theta(\boldsymbol{x_i}) : \boldsymbol{\theta} \in \boldsymbol{\Theta}\}, \tag{4}$$

where $\boldsymbol{\Theta}$ is the space of all possible values of the parameters [7]. These parameters are to be learned from the training set. More specifically, the parameters are learned from solving a minimization problem. The minimization problem can be stated as

$$\hat{\boldsymbol{\theta}} = arg \ \underset{\boldsymbol{\theta}}{min} \ J(\boldsymbol{\theta}). \tag{5}$$

where the objective function, $J(\boldsymbol{\theta})$, is called a cost function. To measure the performance of the function f_θ in approximating the output, the cost functions contain a loss function. The loss function means a formula that quantifies the similarity of the model's predictions compared to the true output. The best loss function for a specific problem depends on several different factors. This could depend on which parametric family is used, if it's a regression or classification problem, or if regularisation should be used or not. During this project, the choice of loss function is considered separately for each implemented model. The cost function can be defined as

$$J(\boldsymbol{\theta}) = \frac{1}{n} \sum_{n=0}^{n-1} L(y_i, f_\theta(\boldsymbol{x_i})) \tag{6}$$

where y_i is the true output label and L is the loss function. Most often there is no analytical solution to the minimization problem, so the problem is instead solved numerically for an approximate solution. There are many different algorithms to use for solving the minimization problem and which one of them is most appropriate to use depends on the specific problem. Therefore a loss function will also be chosen specifically for each implementation. However, the real goal of a model is not to be able to classify the training data, but to predict labels of new, unseen data. Therefore, to evaluate the performance of the model on unseen data, parts of the available data are not included in the training set. These data points are instead included in a test set. The problem of finding the best parameters with regard to the test set can be stated as

$$\hat{\boldsymbol{\theta}} = arg \ \underset{\boldsymbol{\theta}}{min} \ E_{new}(\boldsymbol{\theta}). \tag{7}$$

Here E_{new} represents some evaluation metric that describes the model's performance in predicting the labels of the data points included in the test set [7]. But as previously stated, the test set is not available during training when the parameters $\hat{\boldsymbol{\theta}}$ are determined. Thus, even though the objective presented in 7 is what is ultimately being solved, it can't be optimized explicitly. Instead, the training objective in 5 is minimized, generating a solution that is expected to also generalize well to the unseen data in the test set.

The fact that it is not possible to explicitly minimize the error in (7) and the data which the model, in the end, should classify is unknown during training has several implications on the problem. If one builds a very complex model which is able to predict the training data extremely accurately or even without any errors, there is a risk that the model is overfitted to the training data. One could describe this phenomenon as the model learning to classify the specific data points in the training set rather than the trends or patterns in the data. The consequence of overfitting is the model performs significantly worse on the test set compared to the training set [7]. One method to reduce overfitting is to apply some type of regularisation when solving the minimization problem in (6).

One method of regularisation is to modify the cost function in (6) by adding a regularisation term. The desired effect of this addition is to penalize large parameter values. This method is called explicit regularisation and the general formula can be defined as

$$\hat{\boldsymbol{\theta}} = arg \ \underset{\boldsymbol{\theta}}{min} \ J(\boldsymbol{\theta}) + \lambda \ R(\boldsymbol{\theta}) \tag{8}$$

where $R(\boldsymbol{\theta})$ represents the regularisation term [7]. λ is a parameter that determines the weight of the regularisation term compared to the cost function. However, there are also several techniques of regularisation that don't involve any change to the loss function. These methods are referred to as implicit regularisation. One such technique is called early stopping, which is applicable to any method which involves iterative numerical optimization. As the name suggests, the technique means that the training stops before a minimum of the cost function is reached [7]. Since different types of regularisation are appropriate to use for different loss functions, the choice of regularisation will be made separately for each model that is implemented.

After training a parametric machine learning model, both the structure of the function f_θ and the parameters $\boldsymbol{\theta}$ are known. Thus, the mathematical reasoning of how the model classifies data points is entirely available to the researcher. This leads to a much higher level of interpretability of the model compared to non-parametric models. Since the structure of parametric models are generally more rigid means that they are often less computationally demanding than non-parametric models. The more rigid nature of parametric models also leads to a lower risk of overfitting data compared to non-parametric models that are generally more flexible [15].

3 Method

The following section describes the methodology of the project and an insight into why different choices were made regarding the approach to the problem. The aim of the project is both to explore the amount of information that can be extracted from the transaction data in regard to financial crime as well as if this can be used to improve the performance of a model using all the available data. Different time series models will be investigated on their ability to classify customers based on their transaction data. The best-performing model will then be combined with another model utilizing the rest of the customer data. One tree model will be used as a benchmark, which utilizes all of the customer data and the sums of different transaction types over the time period. The scoring of different customers by the time series models will then be added as input to the tree-based model. The performance change when adding this variable will give an indication regarding if the use of machine learning models utilizing time series data can provide a meaningful improvement in financial crime detection. The lists of the highest-scored customers from the baseline tree model and the best-performing time series models will also be compared. If they are similar it indicates that the time series models were not able to find any meaningful patterns which the tree-based model was not able to capture. A large deviance in the customers which are scored the highest by the two models indicates that the models are finding different patterns which characterize financial crime. The dataset used in this project is from the Nordic bank Nordea. All details about the dataset and its features are not disclosed in this report due to the sensitive nature of the information.

3.1 Dataset

The dataset for this project contains alerts from Swedish customers from Nordea's database. The exact number of alerts is not disclosed due to confidentiality reasons, but it can be considered a relatively large dataset. These alerts from the rule-based method have been triggered at different time points during a 3.5-year period. If there have been several alerts involving the same customer these will show up several times in the dataset. This data is split up into two different datasets representing different types of information linked to the alerts. Firstly, there is a dataset that contains information about the customer associated with each alert. When a customer opens an account at a bank they need to fill in certain information about themselves. Banks are legally obliged to collect this information in order to prevent financial crime and it is often referred to as "Know Your Client" (KYC) - data. Examples of entries for KYC data are the types of transactions that the customer expects to utilize, if they are somehow associated with any other countries than Sweden, and other general information about the customers, such as age. Some information about the customer is also obtained at the time of the alert. This information includes for example the number of accounts and account balances at the time of the alert. This dataset also contains the class label of each alert, meaning if the customer related to the alert was involved in financial crime or not. Class labels have been set by domain experts who investigate clients who have been found to exhibit potentially illicit behaviour by the rule-based method. In total, the dataset contains 45 features which are nominal and ratio variables. A demonstration of the structure of this dataset can be seen in Figure 1

Alert ID	Expected cash usage	Associated with country	Age	Number of accounts	Class label	...
1	True	Norway	35	5	1	
2	False	Null	40	10	-1	
3	True	Germany	21	2	-1	
...						
130	False	Null	66	4	1	

Table 1: The data structure of the customer information related to each alert. All of the data entries in the table are examples and are not taken from a real alert. This table only serves as a demonstration of the structure of alert data and does not display all of the features.

The other category of data associated with the alerts is transaction data. This is found in a separate dataset which contains information about each transaction that has been performed by each customer related to an alert. The transaction dataset contains information about transactions associated with each alert from 90 days before the alert was triggered until the alert date. Each row in the dataset represents a transaction and has a unique transaction ID. It also has a column with alert IDs representing which alert each transaction is connected to. The temporal information of each transaction is limited to date and does not contain any more granular data about the time of day. The first column represents the transaction amount. The rest of the columns represent different types of transactions that have been found to be risk factors for financial crime. When a particular transaction reflects one or more of these types, these will also contain the transaction amount or otherwise be zero. The transaction data set contains 28 different types of transactions. A demonstration of the structure of the transaction dataset can be seen in Table 2.

Transaction ID	Alert ID	Date	Total transaction amount	Gambling	Crypto	ATM deposit	...
1	1	05-10-2021	465.2	0	0	0	
2	1	06-10-2021	1200.5	0	1200.5	0	
3	2	10-10-2021	15040.2	0	0	0	
...							
1520	130	23-07-2022	16.0	0	0	16.0	

Table 2: The arrangement of the transaction data before any preprocessing.

3.2 Preprocessing the data

To classify each alert based on transaction data, first all of the transactions from Table 2 have to be sorted by which alert they are related to. The transaction data related to each alert then need to be arranged in a way that is appropriate to use as input to a TSC model. Most TSC models are only able to use input where all of the time series in the training and testing sets are of equal length. However, in the original transaction dataset, different customers have varying numbers of transactions, and these transactions are unevenly distributed over time. In order to expand the range of applicable models, the transaction data is rearranged into a structure where each time series has equal length. The chosen method to do this is to divide the total time period T into shorter time periods Ω_n. Every transaction executed during a time period is then summed. This means that each row in Table 2 which corresponds to the same Alert ID and the same time period is summed. The transactions associated with an alert can be represented as an irregularly sampled time series $\{z_m\}_{m=1}^{M}$, where the corresponding sample times are denoted by $t_m \in (0, T)$ and M is the number total number of transactions performed during the time period $(0, T)$. This original time series is resampled into a new time series $\{x_n\}_{n=1}^{N}$ with a fixed sample period as follows

$$x_n = \sum_{m \in \Omega_n} z_m \quad n = 1, \dots, N. \tag{9}$$

Here $\Omega_n = \{\forall t_k \; s.t.(n-1)T_s \leq t_k < nT_s\}$ and T_s denotes the sampling period of the new time series. The result of this process is that the transaction data of each alert is sorted into an individual table where each column represents a transaction type and each row corresponds to a time period. Each table represents a multivariate time series where each column represents a feature and each row represents a time step. The length N of the new time series found by

$$N = \frac{T}{T_s}. \tag{10}$$

The arrangement of the transaction data for an example alert is demonstrated in Table 3, where the new sampling period is one day. In this project, the time period T which transaction data is collected for each alert is 90 days. When this is plugged into (10) together with sampling period T_s from the example, we obtain a maximum time index of 90 which can be seen in Table 3.

Time index	Total transaction amount	Gambling	Crypto	ATM withdrawal or deposit	...
1	10200.4	0	6500.1	3700.3	
2	750.1	750.1	0	0	
3	0	0	0	0	
...					
90	50.0	25.2	0	0	

Table 3: The arrangement of the transaction data that is used as input to the time series models.

When transactions are summed over time periods information is lost about the number of transactions that were executed, when they were executed, and the amounts of the individual transactions. The more transactions that are summed over a time period, the more information is lost. A higher frequency of transactions leads to more transactions being summed over the same period which in turn leads to more information loss. This process resembles that of low-pass filtering, where the information from signals of higher frequencies is reduced more compared to the lower frequencies.

Using a longer sampling period T_s when creating the new time series z_n increases the effect of the low pass filtering. This means that more information about the temporal patterns in the transaction data is lost, primarily in the parts of the time series where the frequency of transaction are higher. When larger values of T_s are plugged into (10) one observes that the time series length N becomes shorter. This can be a desirable effect since longer time series cause higher memory usage when processing them in TSC models. The time complexity of many TSC models is also influenced by the length of the time series. This means that as the length of the time series grows, the computational requirements of many TSC models also increase, potentially resulting in longer processing times. Compressing the time series therefore also widens the scope of models which can process the transaction data set within a feasible computing time.

Longer sampling periods also lead to the number of transactions executed during each period becoming higher on average. From (10) it can be deduced that this also leads to more transactions being summed in each time step of the new time series z_n. The original time series x_m contains a large proportion of zeros since each transaction only matches one or a few of the transaction types. Most customers also don't utilize every type of transaction and might not perform any transactions at all during time periods. This leads to the time series representation of most alerts becoming relatively sparse. However, as longer sampling periods are used and more transactions are summed this effect is reduced. Jiang et al. explain that sparsity in data causes deep learning algorithms to become inaccurate [16]. It, therefore, seems worthwhile to evaluate how the length of sampling periods affects the performance of different time series models. These sampling periods were chosen to be 1, 2, and 3 days, which results in time series lengths of 90, 45, and 30 respectively.

As mentioned previously, some transaction types do not occur frequently in the dataset. When features are too sparse they are not meaningful to use in time series analysis. More features lead to more extensive memory usage and often longer computing times when they are used as input to TSC models. For this reason, some of the features from the original transaction data set are removed. Some transaction types in the original dataset are also split up into debit and credit. To further decrease the size of the dataset, these features are added together into one. The final representation of the transaction data contained 18 features in total.

The features which contain transaction amounts span multiple magnitudes. To be able to capture the pattern in these variables effectively a log transformation is used. Since many of the transaction sums are zero, one is also added before the log transformation. The formula of the transformation is given by

$$\tilde{T} = log(\hat{T} + 1) \tag{11}$$

where $\hat{T}$ and $\tilde{T}$, represent the transaction sum before and after the transformation respectively. Finally, to ensure that there is no bias and that all variables contribute equally to the model fitting a min-max scaling is also performed before being used in a model. The formula for min-max is given by

$$T = \frac{\tilde{T} - \tilde{T}_{min}}{\tilde{T}_{max} - \tilde{T}_{min}} \tag{12}$$

where $\tilde{T}_{min}$ and $\tilde{T}_{max}$ represent the smallest and biggest value in the set of that feature.

The structure of the input data that was used for the tree-based model is presented in Table 4. Not much preprocessing was made to the data due to the chosen model being very flexible in handling data of different scales and of different types. Transaction features were created by summing all of the transactions related to an alert over the entire time span where information was available. The purpose of this model is to be used as a baseline to compare with the performance of the time series models.

Alert ID	Expected cash usage	Number of accounts	Age	Sum of crypto debit transactions in the last 90 days	Sum of gambling debit transactions in the last 90 days	...
1	False	5	35	7800.5	0	
2	True	9	21	0	250.7	
3	False	1	60	0	0	
4	True	11	27	1090.5	5689.2	

Table 4: The arrangement of the data was used as input to the tree-based models.

The ROC AUC-score of the different time series models with different sampling periods is then evaluated. The models are in this case evaluated in the ROC AUC-score metric since it is a good measurement of how accurate a model is in distinguishing data points between two different classes, even though the data set is unbalanced [7]. The best method for time series classification in terms ROC AUC-score is chosen. The output from this method is added as a feature to the input data used in the baseline tree-based model. This new feature can be referred to as a "Transaction risk score". The resulting data structure of the input to this model is presented in Table 5. After this new feature has been added to the input the performance of the tree-based model is then evaluated again. The purpose of this experiment is to evaluate if a time series model can be used together with a tree-based model to improve performance.

Alert ID	Expected cash usage	Number of accounts	Age	Sum of crypto debit transactions in the last 90 days	Sum of gambling debit transactions in the last 90 days	Transaction risk score	...
1	False	5	35	7800.5	0	0.231	
2	True	9	21	0	250.7	0.567	
3	False	1	60	0	0	0.122	
4	True	11	27	1090.5	5689.2	0.912	

Table 5: The arrangement of the data was used as input to the tree-based models.

3.3 Choice of algorithms

The first step in choosing models to evaluate was to thoroughly research the field of TSC. Subsequently, the task at hand is to determine which algorithms are worth exploring for their ability to classify the time series data in this project. The choice of algorithms to evaluate will be made based on two criteria. Firstly, there should be research that acknowledges the performance of the algorithm in classifying time series data. Secondly, domain knowledge also plays a certain part in choosing algorithms. Knowledge about what kind of transactional behaviour is generally connected with financial crime gives some insight into which types of algorithms could be effective in making such predictions.

In general, the most common starting point when comparing TSC algorithms is the One Nearest Neighbour (1-NN) algorithm with Euclidean distance. However, this has been proven to be a very low benchmark. Bagnall et. al (2017) instead suggest that 1-NN with Dynamic Time Warping is a more competitive and useful benchmark [17]. Therefore, this is chosen as the first algorithm to implement and serves as a benchmark for comparing the other algorithms. Another linear classifier that seems worthwhile investigating is the Random Convolutional Kernel Transform (ROCKET). Dempster, Petitjean, and Webb (2020) show that the ROCKET together with a ridge regression classifier outperforms several state-of-the-art algorithms for TSC using an archive of 85 different datasets [18].

MLSTM-FCN is a deep learning model for time series classification proposed by Karim et al [19]. The model employs the ability to capture relationships between the different variables. This can be an important property since there most likely are inter-dimensional dependencies in the transaction dataset. Financial crime most often involves a series of different types of transactions over a period of time. Detecting these kinds of patterns could give an edge to this model compared to the linear classifiers. Karim et al. show that the MLSTM-FCN classifier reaches state-of-the-art performance on 28 out of the 35 evaluated datasets [19].

To evaluate the customer data and the aggregated transaction statistics the XGBoost algorithm was chosen. Out of the available tree methods, XGBoost was chosen due to its efficiency when working with large datasets. The XGBoost algorithm employs a certain block structure when storing data, which enables the algorithm to work more efficiently in a parallel environment and utilize multiple CPUs. It is also able to process the categorical features which are present in the customer dataset [20].

3.4 Evaluation metrics

The majority of customers in the dataset are not involved in financial crime, meaning that the dataset is imbalanced between the two different classes. An implication of this is that accuracy alone fails to provide a comprehensive view of the performance of a model. This is because a model that simply predicts the most common label can still attain a favorable score. Hence, two additional metrics will be employed to assess the models' performance, specifically chosen to address the challenges posed by imbalanced datasets.

One way of measuring a model's ability to distinguish between two different classes is to use a ROC curve. In the ROC curve, the rate of true positives is plotted against the rate of false positives for different values of the classification threshold between 0 and 1. A demonstration of a ROC curve can be seen in Figure 2. It can be seen that the ROC curve of a perfect classifier goes along the top left border of the plot, while a curve of a classifier making random guesses, goes on a diagonal line across the plot. This means that the AUC of the ROC curve of a perfect classifier and a classifier making random guesses will be 1 and 0.5 respectively. A typical model will have a ROC AUC somewhere between 0.5 and 1. Models with better performance will always lead to a larger value of the ROC AUC. This means that the ROC AUC becomes a good metric for summarizing the result of a ROC curve. This can be referred to as a ROC AUC-score.

The ROC AUC-score is a good performance measure when the output of the model for a data point is a probability between 0 and 1 since it considers different thresholds for classification. However, if the output of a model is only a score of 0 or 1 instead of a probability, the ROC AUC-score is not

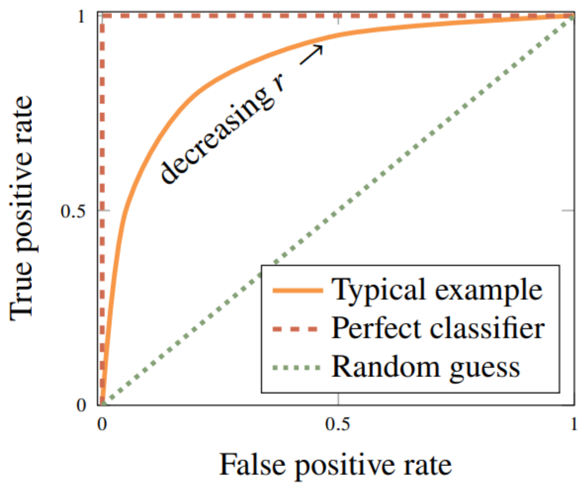


Figure 2: A ROC curve. r refers to the classification threshold [7].

relevant. Since all of the results for different thresholds between 0 and 1 which are computed in the ROC AUC, will generate the same result. A better measurement for methods that only has a binary output while also having an imbalanced dataset is the F1-score. The F1-score can be described as a balanced measurement between the precision and recall of a classifier. The precision of a model means calculating the accuracy only among the samples that were classified with a positive label. The formula for the precision metric is given by

$$precision = \frac{TP}{TP + FP} \tag{13}$$

where TP is the number of true positives and FP is the number of false positives made by the classifier on a dataset. If you instead calculate the accuracy among only the samples that were true positives, you obtain the recall metric. The formula for the recall metric is given by

$$recall = \frac{TP}{TP + FN} \tag{14}$$

where FN is the number of false negatives made by the classifier on a dataset. The formula of the F1-score is then given by

$$F_1 = \frac{2 * precision * recall}{precision + recall}. \tag{15}$$

Thus, three different performance measures will be used to evaluate models: accuracy, ROC AUC-score, and F1-score. For the F1-score, the classification threshold is set to the default value of 0.5.

3.5 Walk-forward validation

The objective of this project is to train a model during a certain time span and then be able to make accurate predictions about future alerts. To test each model's ability to do this, the models will be trained on data from alerts triggered between two different dates. The trained model will then be tested on alerts triggered in a time period directly after the training data was retrieved. This method for testing and evaluating a model is often referred to as *walk-forward validation.* The walk-forward validation method is chosen for testing the models since it most accurately reflects how the model would perform in production. The dataset is divided into seven distinct training and test periods. Each time period from when the training set is collected is 1 year and each time period from when the test sets are collected is 3 months long. There is a 9-month overlap between consecutive time periods where training data is collected. Figure 3 demonstrates how the different training and test periods are ordered on a timeline. In total, there are 7 training periods and 7 respective test periods. This means that there are in total 7 training sets and 7 respective test sets. Each model's performance is then computed by taking the average of the seven scores obtained on the respective test sets.

3.6 Limitations

The models in this project are only trained on data from alerted customers. This is only a subset of the entire set of customers. The rule-based method which generates alerts is not perfect and does not detect every case of financial crime in the bank. However, the focus when creating these rules is to have a small proportion of false negatives rather than a small proportion of false positives. A small proportion of false negatives means that the rules can detect a large proportion of the financial crime that is performed by customers of the bank. However, the high number of false positives means that after investigation most of the alerts are disqualified as false alarms, meaning that the customer associated with the alert is found to be innocent. Nevertheless, the dataset of alerts will contain most of the true cases of financial crime committed by customers of the bank which is important for training accurate machine learning models in detecting financial crime. But the fact that the models are trained on a subset of the entire set of customers still creates a bias in the models. However, as long as the rules don't change, this will not worsen the performance of the models in classifying alerts. Nonetheless, it would reduce the performance if the model would be tasked with classifying customer which had not been alerted.

3.7 System specifications

The tests are run on a system with an IntelCore i7 3.00 GHz processor with 32 GB of random-access memory (RAM). All of the models are implemented in Python version 3.10.9.

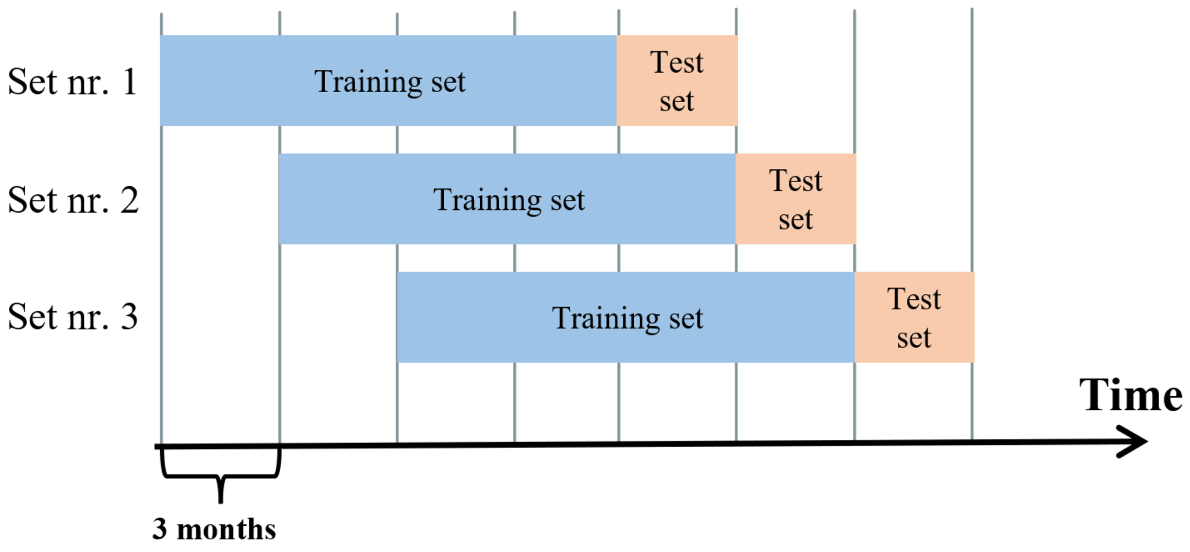


Figure 3: A timeline demonstrating how different training and test sets are ordered in walk-forward validation. The number of splits in the figure is only to make the illustration clearer. 7 different training- and respective test sets are used in this project.

4 Implementation

The following sections will describe the different implementations which were used to predict customers' involvement in financial crime. The first three sections describe the different time series models which are used to make predictions only based on transaction data. These models will be evaluated on their performance to determine which type of model is the best for predicting financial crime from transaction data. In the fourth and last section, the XGBoost model is described, which makes predictions based on both transaction- and customer data.

4.1 One nearest neighbour with dynamic time warping

The 1-NN model is a non-parametric machine learning model. An intuitive explanation of how the model classifies a new data point is by finding the most similar data point in the training set and then setting the label to the same as that point [7]. One of the simplest ways of measuring the similarity between two time series is to calculate the Euclidean distance. This would mean computing the distance between each data point with the same index in the two time series and then aggregating these distances for the entire time series. To account for displacement on the time axis the dynamic time warping (DTW) similarity measurement can be used instead. The DTW algorithm achieves this by computing a matrix of distances between each point in the two time series and then finding the optimal path through this matrix [21]. Consider the two vectors $\mathbf{a} = (a_1, a_2, ..., a_m)$ and $\mathbf{b} = (b_1, b_2, ..., b_m)$ of the same the length m. Let $\mathbf{M}$ represent a matrix that represents the absolute difference between every value of the two vectors. This matrix $\mathbf{M}$ containing distances between points can be referred to as a local cost matrix. The local cost matrix is thus computed by calculating each entry as $M_{i,j} = (a_i - b_j)$. The DTW algorithm equates to finding a path through the local cost matrix which minimizes the sum of all matrix entries or distances in the path. This optimal path can be referred to as the "warping path". A path through the local cost matrix $\mathbf{M}$ is given by

$$\mathbf{P} = (w_1, w_2, ..., w_N) \tag{16}$$

where w represents pairs of indices from the matrix $\mathbf{M}$ and $m \leq N \leq 2m - 2$. Thus, the nth element of $\mathbf{P}$ is given by $w_n = (i, j)$. Let d_n represent the distance found at the coordinate w_n of the local cost matrix $\mathbf{M}$. The total distance of a path $\mathbf{P}$ is found by computing the sum

$$D_{\mathbf{P}} = \sum_{n=0}^{N} d_n. \tag{17}$$

To find the warping path $\hat{\boldsymbol{P}}$ one solves the minimization problem

$$\hat{\boldsymbol{P}} = arg \underset{\boldsymbol{P} \in \boldsymbol{\mathcal{P}}}{min} D_{\mathbf{P}} \tag{18}$$

where $\mathcal{P}$ represents the space of all possible paths. The warping path must adhere to two different conditions. Firstly, the path must begin at $w_1 = (1, 1)$ and end at $w_N = (m, m)$. The path can also only move to adjacent cells (including diagonally) in a positive semi-definite direction with regard to both indices. In other words, if $w_k = (i, j)$ and $w_{k-1} = (i', j')$, a valid path must abide according to $0 \leq i - i' \leq 1$ and $0 \leq j - j' \leq 1$. The computed sum over the warping path $D_{\hat{\mathbf{P}}}$ is the DTW distance of the two vectors [21]. When computing the DTW distance between two multivariate time series, the DTW distance is computed for all the different features and then summed. Algorithm 1 shows the different steps classifying instances of a test set with the 1-NN DTW model.

The time complexity of the DTW algorithm, when used on a multivariate time series, is $\mathcal{O}(k * m^2)$, where k is the number of features and m is the length of the time series [10]. This quadratic scaling with regard to the time series length may restrict the range of time series lengths suitable for the algorithm's usage. When classifying a new data point with the 1-NN model, a comparison needs to be made with every time series in the training data set. This means that the time complexity of the 1-NN model becomes $\mathcal{O}(l)$, where l is the number of data points in the training set. When running the 1-NN model, both the training- and test sets need to be stored in RAM to be able to be directly accessible by the processor to make computations. When using large training or test sets along with long time series that have a high number of features, the model demands a significant amount of RAM. The implementation of 1-NN with DTW in this project is implemented in this project with the KNeighborsTimeSeriesClassifier class from the Sktime library with version 0.18.0 [22] [23].

4.2 ROCKET

Another method for classifying time series is called the Random Convolutional Kernel Transform (ROCKET). This method uses convolutional kernels to capture relevant features in the time series. This can be seen as a feature extraction stage where the structure of the input data is transformed from a time series to a more typical format for machine learning models. The extracted features are then used as input to a machine learning model, which was chosen to be logistic regression in this implementation. A flowchart explaining how the ROCKET was used together with logistic

Algorithm 1 One nearest neighbour with dynamic time warping algorithm

```
1: for every time series in the test set do
2:     Let $\boldsymbol{a}$ represent the time series from the training set of the current iteration.
3:     for every time series in the training set do
4:         Let $\boldsymbol{b}$ represent the time series from the training set of the current iteration.
5:         for every feature $i$ in time series do
6:             Find the warping path $\hat{\boldsymbol{P}}_i$ with DTW distance $D_{\hat{\mathbf{P}},i}$ of the current feature $i$ between $\boldsymbol{a}$ and $\boldsymbol{b}$.
7:         Sum the DTW distance $D_{\hat{\mathbf{P}},i}$ of every feature to obtain the total DTW distance $D_{\hat{\mathbf{P}}}$ of the two time current series.
8:     Choose the time series from the training set which resulted in the smallest DTW distance $D_{\hat{\mathbf{P}}}$ and check the label of that instance. Classify $\boldsymbol{a}$ as that label.
```

regression is shown in Figure 4.

The most common use of convolutional kernels is in convolutional neural networks where the weights in the kernels are learned through backpropagation. However, this process can be computationally demanding and might not always prove effective in learning patterns in the data. In the ROCKET method, the weight, length, bias, dilation, and padding of each kernel are instead set at random. k number of random kernels are then applied to the time series. Applying a one-dimensional kernel to a time series means computing a sliding dot product between the kernel and the time series. If $\boldsymbol{X}$ represents a part of a time series, the process of applying a kernel $\boldsymbol{\omega}$ to this part of the time series can be described as

$$\boldsymbol{X} * \boldsymbol{\omega} = \sum_{j=0}^{l_{kernel}-1} \boldsymbol{X}_{i+(j \times d)} \times \omega_j. \tag{19}$$

Here i represents the starting index of $\boldsymbol{X}$. d denotes the dilation of the kernel, l_{kernel} represents the kernel length and $\boldsymbol{\omega}$ represents the weights of the kernel. The kernel is moved along the time series so that a dot product is computed for each index of the time series. When a kernel has been applied to the entire time series a new vector has been obtained with the resulting dot products. Two values are extracted from this vector. Firstly, the maximum value from the vector is extracted. Secondly, the proportion of positive values (PPV) in the result vector is measured. The purpose of the maximum value is to capture the strongest match in the time series to the pattern in the kernel and the PPV represents how frequently the pattern appears in the time series. When every kernel has been applied to the time series, $2k$ features have been extracted. In the implementation in this project, k was lowered from 10000, which is proposed by Dempster, Petitjean, and Webb, to 3000 to reduce computation time. The ROCKET is applied with the Rocket class from the Sktime library with version 0.18.0 [22] [24].

The extracted features can then be used to train a classifier. Dempster, Petitjean, and Webb propose to use logistic regression as a classifier when the number of features is larger than the number of instances in the training set. Since the number of data points in the smallest training set is larger than $2k$, logistic regression was chosen for classification. In logistic regression, the function f_θ in (3) is given by

$$f_\theta(\boldsymbol{x}) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + ... + \theta_p x_p = \boldsymbol{\theta}^\top \boldsymbol{x}. \tag{20}$$

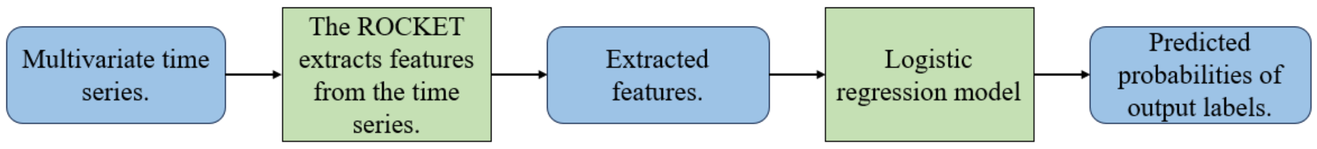

Multivariate time series.
The ROCKET extracts features from the time series.
Extracted features.
Logistic regression model
Predicted probabilities of output labels.

Figure 4: A flowchart showing the different steps of classifying a time series with the ROCKET method.

The most common loss function to use in logistic regression is binary cross entropy which was therefore chosen to be used in this implementation. Thus, the loss function in (6) for this implementation is given by

$$L(y, f_\theta(\boldsymbol{x})) = ln(1 + e^{-y\boldsymbol{\theta}^\top \boldsymbol{x}}). \tag{21}$$

Since the ROCKET model applies random kernels to the time series, some kernels will be useful since the pattern doesn't appear anywhere in the data set. This means some of the extracted features will not be useful for predicting the labels and adds noise to the model. For this reason, the L^1 or LASSO regularisation was chosen. L^1 regularisation works as a feature selector where it shrinks the less useful features of the input to zero [7]. Thus, the regularisation term from (8) is given by the formula for L^1 regularisation which is

$$R(\boldsymbol{\theta}) = ||\boldsymbol{\theta}||_1. \tag{22}$$

To solve the minimization problem, the 'Saga' solver from the Scikit-learn library of version 1.2.2 is used [25]. The choice of the solver was due to its fast convergence when applied to problems involving large training sets [26].

4.3 MLSTM-FCN

Karim et al. propose a method for TSC involving both an LSTM- and a fully convolutional neural (FCN) network [19]. The following section will provide a summary of the model's architecture and explain the choices made for the implementation in this project. In this model, the multivariate time series is used as input to both an LSTM- and an FCN network. The FCN network works as a feature extractor that captures temporal patterns which are related to the output. The following passage provides an explanation of the general framework of an FCN network. The values of the nodes before the first layer in the FCN network will be given by the input time series $\boldsymbol{x}$ as

$$\boldsymbol{h}^{(0)} = \boldsymbol{x}. \tag{23}$$

These values are then propagated forward through a series of convolutional layers and activation functions. The result obtained by applying a convolutional layer can be expressed as

$$\boldsymbol{a}^{(l+1)} = (\boldsymbol{h}^{(l)})^\intercal \boldsymbol{W}^{(l)} \tag{24}$$

Here $\boldsymbol{W}^{(l)}$ represents the weights and biases of the lth layer, and $\boldsymbol{a}^{(l+1)}$ represents the output of the lth layer. The output is then passed to an activation function. The process of applying an activation function to the output is given by

$$\boldsymbol{h}^{(l+1)} = \sigma(\boldsymbol{a}^{(l+1)}) \tag{25}$$

Here the symbol σ denotes a given activation function. If the network contains L layers, the computations outlined in equations 24 and 25 are repeated for each respective layer until the final output $h^{(L)}$ is obtained. The function mapping the input-output relationship of the model is given by the final nodes of the network as

$$f_{\boldsymbol{\theta}}(\boldsymbol{x}) = h^{(L)} \tag{26}$$

where $\boldsymbol{\theta}$ represents the weights and biases found in $\boldsymbol{W}$.

An LSTM network is a type of recurrent neural network (RNN). RNNs are a type of neural network algorithm designed to capture patterns in sequential data. These networks are characterized by their ability to maintain information from previous time steps, which is made possible by the presence of feedback loops within the network [27]. The RNN has a hidden state for each time step of the input time series. The general framework of an RNN network is similar to that of the FCN network but contains a few differences. In the case of an RNN, the values of initial nodes $\boldsymbol{h}^{(0)}$ presented in 23 only represent data from the first time step of the time series, instead of the whole time series which is the case for the FCN network. Thus, the initial nodes of the RNN are given by

$$h^{(0)} = x^{(0)} \tag{27}$$

where $x^{(0)}$ denotes the first value from the time series. The next step of the network can then be represented as

$$a^{(t)} = b + Wh^{(t)} + Ux^{(t-1)} \tag{28}$$

where b is a bias, W and U are trainable weights and $h^{(t)}$ represents the memory from previous states in the tth iteration. During the first iteration of the network when t is equal to 0, the variable $x^{(t-1)}$ is set to zero. To create the memory input for the next hidden state, a^t is then processed by an activation function as presented in 25 for the FCN network. The computations presented in 28 and 25 are then repeated recursively for each time step of the time series. Thus, if T represents the length of the time series, the computations are repeated T times. The function $f_{\boldsymbol{\theta}}$ mapping the input-output relationship of the model is just as for the FCN model given by the nodes following the last layer of the network. It is thus given by

$$f_{\boldsymbol{\theta}}(\boldsymbol{x}) = h^{(T)} \tag{29}$$

where $\boldsymbol{\theta}$ is given by the weights W and U and bias b.

But as the feedback loops become longer to capture more long-term dependencies RNNs become less and less effective. To be able to capture dependencies over longer sequences LSTM networks have been developed. To choose the number of cells in the LSTM model, Karim et al. propose to do a grid search for three different values, 8, 64, or 128. Due to time constraints for this project and a large dataset resulting in long training times, the number of LSTM cells was chosen to the intermediate value of 64. To prevent the LSTM network from becoming overfit to the training data, a dropout layer is also added. This means that for each iteration of the network, random entries of the output vector from the LSTM are set to zero which works as a form of regularisation [28].

Each time series is also processed by a convolutional block containing three temporal convolutional blocks. All three temporal convolutional blocks include a convolutional layer which is followed by a batch normalization layer and a rectified linear unit activation function. As proposed by Karim et al., the convolutional layers inside the three temporal convolutional blocks contain 128, 256, and 128 filters, ordered from the input to the output of the convolutional block. Each of the two first temporal convolution blocks is followed by *squeeze and excite blocks*. The purpose of the squeeze and excite block is essentially to amplify useful feature mappings and reduce the impact of features from channels that aren't proving effective in classifying the output. The first step of the squeeze and excite block involves performing a global average pooling operation on each channel of the input. This operation is followed by two fully connected layers where the first one reduces the dimensionality of the input and the second one restores it to its original format. The dimensionality reduction of the bottleneck layer is set by a reduction ratio parameter, which in this project will be set to 16 as proposed by Karim et al. and Hu, Shen, and Sun [19] [29]. The purpose of these dense layers is to model the interdependencies between different channels in the network by generating a set of weights indicating which channels contain the most useful filter kernels. These weights are then multiplied with the original input to the squeeze and excite block. These recalibrated kernel weights from the squeeze and excite block are then passed to the next layer of the network [19]. The third and last temporal convolutional block is instead of a squeeze and excite block, followed by a global average pooling layer. The output from this layer is then concatenated with the parallel LSTM- block and then passed to a softmax activation function. Two probabilities are returned from the softmax function indicating the probabilities of the different labels [19]. In Figure 5 one can see an overview of the models' architecture.

Binary cross entropy was chosen as the loss function for the implementation. No regularisation term was added to the loss function. Most of the variables of the network and training were set to the values proposed by Karim et al. with a few exemptions. When choosing an algorithm to solve the minimization problem posed by the network the stochastic gradient descent algorithm was chosen. This choice was made due to the limited computing resources assigned to this project compared to the relatively large size of the network. The model has over 320 000 trainable parameters and the training time with the computing resources allotted to the project is considerable. Regarding the training times of neural networks, Bottou writes "Use stochastic gradient descent when training time is the bottleneck." [30]. For this reason, stochastic gradient descent was chosen as the optimization algorithm for the implementation. When running the model on the dataset it was noticed that there were large oscillations of the loss over different epochs. This led to a large variance in the accuracy of the results of the model, which is undesirable from a model in any realistic scenario. Therefore the learning rate was decreased to 5e-5 from 1e-3, which was proposed by Karim et al. To compensate for the lower learning rate, momentum was added to the stochastic gradient descent algorithm to speed up convergence [31]. Karim et al. propose to train the network for 250 epochs. It was noticed that when training the network for that number of epochs, the model's performance in classifying the validation set was deteriorating towards the end of the training. This means that the model was becoming overfit to the training data. To combat this effect of overfitting, the number of epochs was reduced to 180, which is a form of implicit regularisation. The MLSTM-FCN model is implemented using the Tensorflow library of version 2.12.0 [32].

4.4 XGBoost

Boosting is a type of algorithm where a set of weak learners are used together to create a strong model [7]. XGBoost is a type of boosting algorithm created by Chen and Guestrin (2016) [20]. The following section provides a summary of the XGBoost algorithm. In XGBoost the weak learners are generally smaller decision trees that are added to the algorithm in an iterative manner. The first step of the algorithm is to make an initial prediction of the output.

$$\hat{y}_i^{(0)} = 0 \tag{30}$$

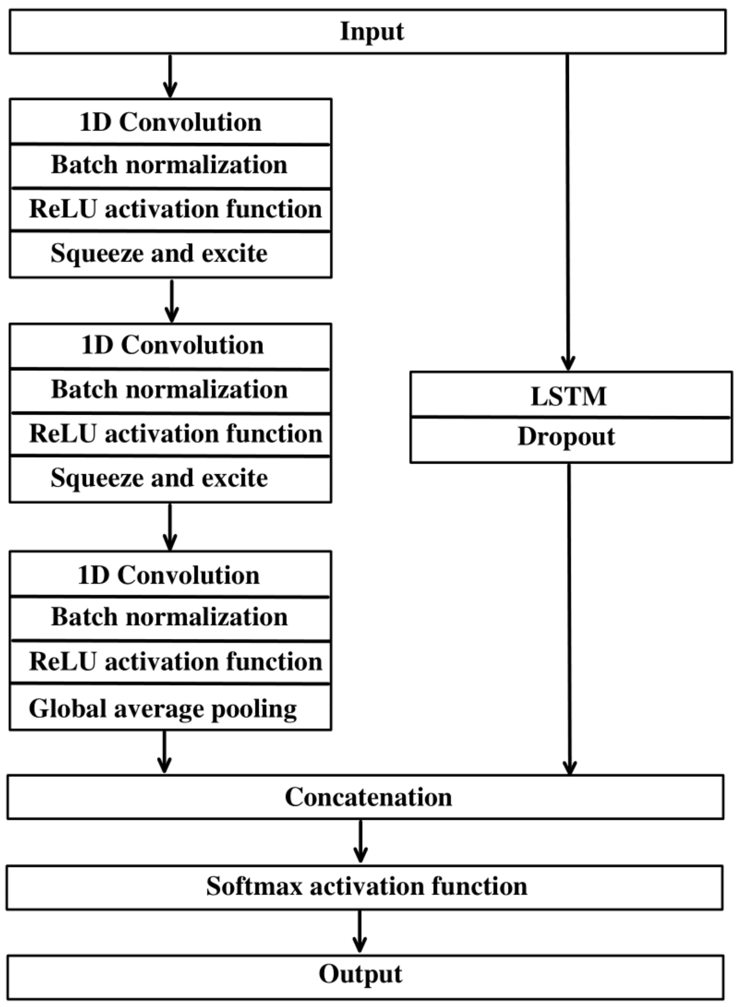


Figure 5: Overview of the building blocks of the MLSTM-FCN model architecture.

Here $\hat{y}_i^{(k)}$ is the algorithm's prediction at the tth iteration of the label of a datapoint $\boldsymbol{x}_i$ from the training set. In each iteration of the training, a new decision tree model is added.

$$\hat{y}_i^{(1)} = f_1(\boldsymbol{x}_i) = \hat{y}_i^{(0)} + f_1(\boldsymbol{x}_i) \tag{31}$$

$$\hat{y}_i^{(2)} = f_1(\boldsymbol{x}_i) + f_2(\boldsymbol{x}_i) = \hat{y}_i^{(1)} + f_2(\boldsymbol{x}_i) \tag{32}$$

Here $f_k(\boldsymbol{x}_i)$ represents the decision tree generated in the kth iteration of the training. New decision trees are added until the maximum iteration K is reached.

$$\hat{y}_i^{(K)} = \sum_{k=1}^{K} f_k(\boldsymbol{x}_i) = \hat{y}_i^{(K-1)} + f_K(\boldsymbol{x}_i) \tag{33}$$

Each new decision tree is created by generating a set of splits in the data. One such split could for example be if a customer has more or less 10 000 Swedish crowns in their bank account. The training set is then split up into two new nodes based on if they fulfill that criterion or not. When all the splits have been generated all of the training data has been split up into a set of $leaf$ nodes. Each leaf is associated with a weight w, which is the value that should be added to the prediction of all instances of the data which are classified to that node. So, to create a new decision tree f_t at an iteration t a tree structure needs to be formed and weights need to be assigned to each leaf node. Finding the best f_t means solving the minimization problem

$$\mathcal{L}^{(t)} = \sum_{i=1}^{n} l(y_i, \hat{y}_i^{(t-1)} + f_t(\boldsymbol{x}_i)) + \Omega(f_t) \tag{34}$$

where l represents a loss function and Ω represents a function that measures the complexity of the tree structure in f_t. A requirement for a loss function l to be able to be used in the XGBoost algorithm is that it needs to be second-order differentiable. The chosen loss function for the implementation in this project was binary cross-entropy, which is the default option for the XGBClassifier class in the XGBoost library [33]. We denote the first-order derivative of the loss function with regard to the predicted label as

$$g_i = \partial_{\hat{y}^{(t-1)}} l(y_i, \hat{y}^{(t-1)}) \tag{35}$$

and the second order as

$$h_i = \partial^2_{\hat{y}^{(t-1)}} l(y_i, \hat{y}^{(t-1)}). \tag{36}$$

A second-order Taylor expansion can then be applied to the loss function l in 34 and we obtain

$$\mathcal{L}^{(t)} \simeq \sum_{i=1}^{n}(l(y_i, \hat{y}_i^{(t-1)}) + g_i f_t(\boldsymbol{x}_i)) + \frac{1}{2}h_i f_t^2(\boldsymbol{x}_i) + \Omega(f_t). \tag{37}$$

The constant terms can be removed since they are not relevant to the minimization problem and the following is obtained

$$\mathcal{L}^{(t)} \simeq \sum_{i=1}^{n} g_i f_t(\boldsymbol{x}_i)) + \frac{1}{2}h_i f_t^2(\boldsymbol{x}_i) + \Omega(f_t). \tag{38}$$

The set of all instances of training set which fulfills all of the criteria for leaf node j can be denoted as

$$I_j = \{i|q(\boldsymbol{x}_i) = j\} \tag{39}$$

where q is the function that maps each instance in the training set to a leaf node. If the T represents the set of all the leaf nodes in a tree, the function $\Omega(f_t)$ can then be written as

$$\Omega(f_t) = \gamma T + \frac{1}{2}\lambda\sum_{j=1}^{T} w_j^2 \tag{40}$$

where γ and λ are two different regularisation parameters. The parameter γ makes the algorithm more conservative by penalizing more complex tree structures with more leaf nodes. To limit the impact of an individual tree on the prediction of the entire model, the parameter λ penalizes higher values of leaf weights w_i. The objective function of the minimization problem from 38 can then be rewritten as

$$\mathcal{L}^{(t)} = \sum_{i=1}^{n}(g_i f_t(\boldsymbol{x}_i)) + \frac{1}{2}h_i f_t^2(\boldsymbol{x}_i)) + \gamma T + \frac{1}{2}\lambda\sum_{j=1}^{T} w_j^2 = \sum_{j=1}^{T}((\sum_{i\in I_L} g_i)w_j + \frac{1}{2}(\sum_{i\in I_j} h_i + \lambda)w_j^2) + \gamma T \tag{41}$$

The optimal value for a weight for a leaf j can then be computed by

$$w_j^* = -\frac{\sum_{i\in I_J} g_i}{\sum_{i\in I_J} h_i + \lambda}. \tag{42}$$

and all of the optimal weights for a fixed tree structure q can be calculated by

$$\mathcal{L}^{(t)}(q) = -\frac{1}{2}\sum_{j=1}^{T}\frac{(\sum_{i\in I_J} g_i)^2}{\sum_{i\in I_J} h_i + \lambda} + \gamma T. \tag{43}$$

So to learn the weight parameters in the XGBoost method the minimization problem can be solved analytically, instead of numerically as described in Section 2. But 43 can only be used to calculate optimal leaf weights of an already set tree structure. In every iteration of the XGBoost algorithm, a tree structure needs to be generated before the leaf weights can be calculated. However, the number of possible tree structures is often too high to be feasible to calculate. For this reason, XGBoost employs a greedy algorithm to generate tree structures where each split is considered individually. To decide each split when generating a tree, every possible split of every variable in the training set is considered. The split that generates the highest *gain* out of all possible splits is then chosen. The formula for the gain of a split is given by

$$gain = \frac{1}{2}\left(\frac{(\sum_{i\in I_L} g_i)^2}{\sum_{i\in I_L} h_i + \lambda} + \frac{(\sum_{i\in I_R} g_i)^2}{\sum_{i\in I_R} h_i + \lambda} - \frac{(\sum_{i\in I} g_i)^2}{\sum_{i\in I} h_i + \lambda}\right) - \gamma \tag{44}$$

where I represents the set of all instances in the current node and I_L and I_R are the instances in the left and right node after the split. Splits are generated until a split doesn't generate enough gain or a set maximum tree depth has been reached [20]. The function mapping the input-output relationship of the algorithm is given by

$$f_{\boldsymbol{\theta}}(x) = f_1(\boldsymbol{x}) + f_2(\boldsymbol{x}) + ... + f_K(\boldsymbol{x}). \tag{45}$$

Here the parameter $\boldsymbol{\theta}$ contains the weights $\boldsymbol{w}$ and the parameters from the function q representing the splits of the different trees. Algorithm 2 shows the different steps of training the XGBoost algorithm.

The relatively large training set available in this project generally decreases the risk of overfitting. Therefore the regularization parameters λ and γ were set to the relatively low values of 1 and 0 respectively. The extensive training set also enables more complex models without the risk of overfitting. Therefore the maximum tree depth M was set to the relatively high value of 6. The total number of trees K generated was set to the default value of 100. The implementation of XGBoost in this project is made with the XGBClassifier class in the XGBoost library of version 1.7.5. All of the parameters not mentioned previously in this paragraph were set to their default values [33].

5 Results

In Table 6 one can see the performance of the different time series models presented in Sections 4.1 to 4.3 in terms of the ROC AUC-score metric. It also displays how the performance varied for

Algorithm 2 Training process of the XGBoost algorithm

```
1: for $k = 1; k \leq K; k++$ do
2:     Let $k$ represent the current iteration of the training and $K$ denotes the total number of trees generated by algorithm.
3:     $m \leftarrow 0$
4:     while $m < M$ do
5:         Let $m$ represent the current tree depth and $M$ the maximum tree depth.
6:         for Each leaf node of the current decision tree do
7:             for Each feature of the dataset do
8:                 for Each possible split of the current variable do
9:                     Compute the gain generated by the split with the formula given by 44.
10:            Select the split which generated the highest gain.
11:            if $gain > 0$ then
12:                Generate the selected split.
13:        $m++$
14:    Compute the weights of the generated decision tree with the formula given by 43.
```

sampling periods of different lengths. Table 7 displays the performance in terms of accuracy for the different time series models with different sampling periods. Table 8 displays the performance in terms of F1-score for the different time series models with different sampling periods. Table 9 displays the combined time it took to train each time series model on all of the 7 training sets. It also shows how the training times varied over different time series lengths.

The computation time for the 1-NN DTW model used on any time series longer than 30 time steps was too long to be completed within the timeframe of the project. Therefore these results are absent in the tables below. 3 of the training sets also contained too much data to be used in the 1-NN DTW model on the system which was used for testing. When these sets were used as input to the 1-NN DTW model, the algorithm required more allocated memory than was available for use. The different training and test sets contain a varying number of data points. This is possible since the data from alerts is collected from different time periods for each set. The number of alerts triggered varies over time, which leads to a different number of alerts ending up in each set. For this reason, the 1-NN DTW model was only tested on 4 out of the 7 test sets.

	1-NN DTW	ROCKET	MLSTM-FCN
1 day sampling period	-	0.856	0.813
2 days sampling period	-	0.841	0.822
3 days sampling period	0.572*	0.832	0.817

Table 6: Average ROC AUC-scores on the test sets. The 1-NN DTW model's performance was only evaluated on four out of the seven test sets.

	1-NN DTW	ROCKET	MLSTM-FCN
1 day sampling period	-	0.854	0.760
2 days sampling period	-	0.843	0.776
3 days sampling period	0.814*	0.841	0.779

Table 7: Average accuracy on the test sets. The 1-NN DTW model's performance was only evaluated on four out of the seven test sets.

	1-NN DTW	ROCKET	MLSTM-FCN
1 day sampling period	-	0.418	0.421
2 days sampling period	-	0.405	0.414
3 days sampling period	0.252*	0.402	0.443

Table 8: Average F1-scores on the test sets. The 1-NN DTW model's performance was only evaluated on four out of the seven test sets.

	1-NN DTW	ROCKET	MLSTM-FCN
1 day aggregated	-	9.5	39.3
2 days aggregated	-	9.3	25.6
3 days aggregated	65.3*	9.0	15.3

Table 9: Total training times in hours for all the models on the different sets. The 1-NN DTW model's performance was only evaluated on four out of the seven test sets.

Figure 6 presents the ROC curves of the ROCKET and MLSTM-FCN models with different sampling periods. Since the 1-NN model outputs labels rather than probabilities, the ROC curve is not a relevant metric for evaluating this model. Thus, no ROC curves are displayed for the methods involving the 1-NN model.

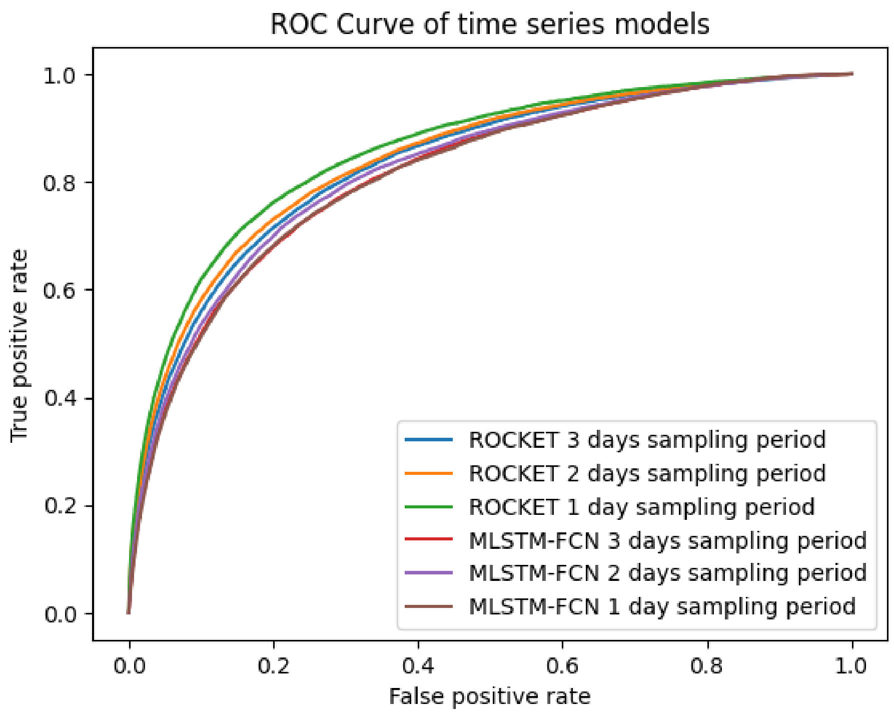


Figure 6: ROC curve for the different time series models.

In Table 10 the accuracy, ROC AUC-score, and training times of the two different methods involving the XGBoost algorithm presented in Section 4.4 are shown. The first method, which can be referred to as "XGBoost baseline", employs the XGBoost algorithm with input data with the structure described in Table 4. The second method, which can be referred to as "XGBoost with ROCKET transaction risk score", uses the same XGBoost algorithm but incorporates an additional element—the output from the ROCKET model with a one-day sampling period. The output from this particular method was chosen because it had the best ROC AUC-score out of all the time series models and sampling periods. The structure of the input data used in this second method, XGBoost with ROCKET transaction risk score, can be seen in Table 5. The XGBoost algorithms are not evaluated in the F1-score metric because the ROC AUC-score is judged to be a more relevant metric for comparing methods that output probabilities of labels rather than just the labels.

	XGBoost baseline	XGBoost with ROCKET transaction risk score
Accuracy	0.891	0.896
ROC AUC-score	0.878	0.885
Training time (minutes)	31.1	30.8

Table 10: Performance of the baseline XGBoost method and the same XGBoost method with the added risk score variable from the best performing time series model.

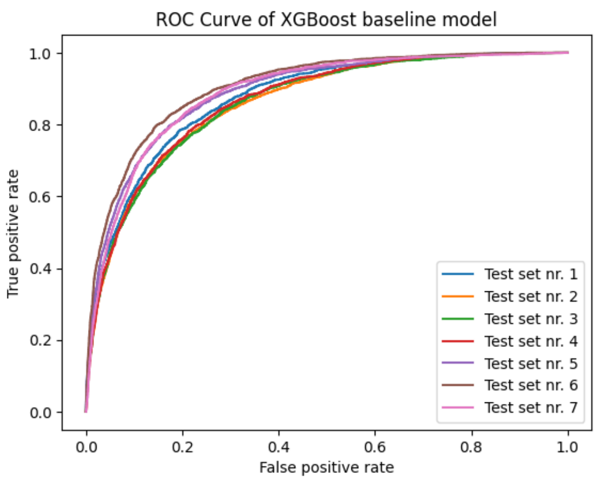
The ROC curves for both the XGBoost baseline method and the XGBoost with ROCKET transaction risk score method are displayed in Figures 7a and 7b, respectively.

Out of the 100 alerts in each test data set that were given the highest risk scores by the XGBoost baseline method, on average 96.3 of them were of the positive class and thus correctly classified. When the same measurement was done with the best-performing ROCKET method, 89.9 alerts were of the positive label. When comparing alerts that were in the 100 highest scored by each of the two models, on average 25.6 of the customers appeared in both sets.

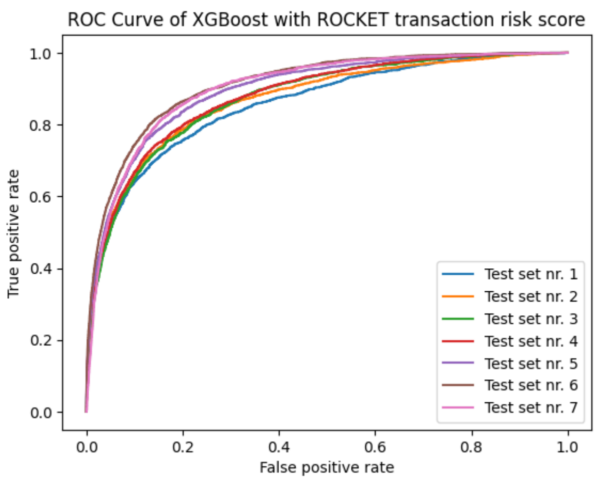
6 Discussion

The ROCKET model was the fastest out of the three algorithms and got only marginally slower with longer time series. In the implementation used in this project, only 3000 kernels were used to limit the computation time. More kernels would likely lead to better performance, but also longer computation times. The MLSTM-FCN approach demonstrated longer computing times, with a noticeable increase in processing time for longer time series. However, the model was implemented using Tensorflow, which is a framework that is possible to run on GPU. Utilizing GPU to run the model would lead to significantly shorter computation times. Even on the shortest time series, the 1-NN DTW model had the longest computation time out of all the tests. As mentioned in the description of the model, the time complexity of the 1-NN model has quadratic scaling in regard to the time series length. This means that none of the longer time series would be possible to run in a feasible time frame. Furthermore, the memory requirements of the algorithm were too extensive to run three of the test sets on the system which was available for testing the models.

Overall, it can be seen that out of the time series models, the ROCKET model had the best performance. It outperformed both of the other time series models in every time series length for every metric but the F1-score, where the MLSTM-FCN model was slightly better. On the time



(a) ROC curve for the XGBoost baseline method.



(b) ROC curve for the XGBoost with ROCKET transaction risk score method.

Figure 7: ROC curves of the two methods involving the XGBoost algorithm.

series length in which 1-NN DTW was able to be used, it outperformed MLSTM-FCN in terms of accuracy. However, it performs significantly worse in the F1-score metric. The 1-NN DTW has a remarkably low F1-score compared to accuracy. This indicates that the model performs well in classifying the majority class, which is the negative label, but worse on the minority class, which is the positive label. The most important aspect of a model in this application is to accurately predict customers who are involved in financial crime. The low F1-score of the 1-NN DTW model therefore strongly indicates that the model is not well suited for the problem. In general, all of the time series models performed poorly in the F1-score metric compared to the ROC AUC-score and accuracy. One reason for this could be that the classification threshold was set to 0.5 for the MLSTM-FCN and ROCKET models, which might not be an appropriate value. The relatively high ROC AUC-scores indicate that the models are able to accurately distinguish between positive and negative instances, which should also lead to a high F1-score with an appropriate classification threshold. So if another threshold was used, the F1-score likely be higher for the MLSTM-FCN model and the ROCKET model.

It can be seen in the results that the ROCKET model performs best on the time series aggregated over fewer days and the MLSTM-FCN model performs better on the more aggregated time series. When the time series is aggregated over fewer days, the time series retains more information, which the ROCKET model seems to be able to utilize. In terms of ROC AUC-score, the MLSTM-FCN model performs the best on the time series which were aggregated over 2 days. It is hard to draw any certain conclusions about the reason for the MLSTM-FCN model performing better on the shorter time series, even though it contains less information than the one which was only aggregated over 1 day. Jiang et al. argue that sparse data representations deteriorate the performance of deep learning methods [16]. The results of this project also indicate that deep learning methods seem to struggle with sparse input data.

In all of the ROC curves for the different methods, a relatively large deviation can be seen between the ROC curves of the different test sets. The most likely reason for this is that there was a considerable size difference between the different training sets. The training sets with higher numbering, containing alerts from later time periods, generally contained more data points. It can be seen in the ROC curves that the performance seems to be generally better on these sets. The biggest training set was double the size of the smallest, which should likely lead to a significant performance difference. The size difference between the different training sets can be considered one of the downsides of walk-forward validation. When there is a large performance difference between different test sets, it becomes hard to judge what performance can be expected from a method in a realistic setting. However, it also highlights a real problem with using machine learning to detect financial crime. Significant changes in the number of alerts reported over different time periods could lead to variance in the performance of machine learning models detecting financial crime, which would be an undesired effect.

The best overall performance out of all the methods was the ones using the XGBoost method, which had both higher accuracy and ROC AUC-score than the best time series method. When evaluating the 100 alerts that had the highest risks of being involved in financial crime according to the methods, the XGBoost was also more accurate than the time series model. However, the ROCKET model that was trained on the longest time series, which was the best time series method, was still fairly accurate when considering the customers with the highest risk scores. The resulting two lists of the 100 alerts with the highest risk scores from the two models were also relatively different, only containing 25.6 matches on average between the different test sets. This indicates that the two models are able to find relatively different patterns in the data to make classifications. The XGBoost with ROCKET transaction risk score method also performed slightly better than the XGBoost baseline method. This means there is information in the transaction risk score variable that the XGBoost uses to make better predictions. This indicates that the ROCKET model is able to detect useful patterns which the XGBoost is not able to detect.

One possible addition to the features of the dataset for the time series methods could be the number of transactions performed by a customer during each sampling period. So for example, if the sampling period is 1 day, the new feature would represent the number of transactions performed by the concerned customer during each day. This would decrease the amount of information being lost in the sampling process that creates the time series. A high frequency of transactions can be an indicator of some types of financial crime, so this feature could provide valuable information. For further studies on the subject, this could be an interesting addition to evaluate.

One thing that makes predicting financial crime with machine learning difficult is the ever-changing circumstances on the subject. The guidelines around what qualifies as financial crime are always changing. This has implications for the rules which trigger alarms and thus end up in the dataset which has been used in this project. It also affects how investigators qualify alarms, meaning that an alert can be classified differently depending on when the alarm was triggered. Behaviours of customers also change over time which affects the dataset. For example, the Covid-19 pandemic certainly changed the transaction patterns of many individuals, which could deteriorate the performance of machine learning models. If a bank or another financial institution is reliant on machine learning for financial crime detection these types of effects need to be taken into account. Considering the large fines and other detrimental consequences that may arise from negligence toward financial crime, it is crucial to carefully evaluate these risks. Since walk-forward validation was used in this project, these effects are accounted for and the result should still give an accurate representation of how the models would perform in production.

7 Conclusion

The results indicate that out of the three time series models, the ROCKET model is the best for classifying financial crime. The ROCKET model was relatively close to the XGBoost model when comparing their performance in ROC AUC-score while only using transaction data. Furthermore, more kernels can be added to the ROCKET model which would likely enhance performance. The MLSTM-FCN method performed slightly worse, but in the deep learning model, there is a great number of hyperparameters to tune, which could not all be considered when testing the model in this project. More hyperparameter tuning would likely improve the performance of this model. The 1-NN DTW did not have good results, too long computation times, and too much memory requirement to be feasible to use on any similar dataset. In summary, the ROCKET and the MLSTM-FCN model are worth investigating further in their ability to detect financial crime and the 1-NN DTW model is not.

Adding the "transactions risk score"-variable from the ROCKET method to the input of the XG-Boost model leads to a small performance improvement. The relatively high ROC AUC-scores indicate that the ROCKET and MLSTM-FCN models are able to rather accurately distinguish instances between the two classes. It seems worthwhile to further explore methods of combining these models with other models which can utilize the customer dataset. The ROCKET model was relatively accurate when considering alerts that were scored with the highest risk of being involved in financial crime. The resulting customers are also somewhat different than the ones found by the XGBoost model. In a scenario where models would be used to find new customers who are involved in financial crime, the ROCKET could be useful since it seems to find other patterns than the XGBoost model.

Machine learning models can be used for multiple different purposes in financial crime detection. In this project, the models were evaluated on their performance of doing the same job as an investigator, i.e. to qualify alarms created from a rule-based method. However, this might not be the most realistic use case. An accuracy of around 90 % is likely not good enough to make any financial institution consider replacing their financial crime investigators. A more realistic use case could be to disqualify alarms that are lowest scored by the machine learning model. This would lessen the burden of investigators and reduce costs for the bank. The models could also be used to find customers in a database that has not been alerted by the rule-based method. Since the machine learning models look for much more complex patterns, they are likely to find other customers than the rule-based methods. The models in this project were not evaluated on their potential performance in the two latter use cases, but the results still give a picture of how effective the models are at detecting the kind of patterns that characterizes financial crime. A model which performs well on the datasets in this study would likely also be useful in the other scenarios.

References

[1] Y. A. Ünvan, "Financial crime: A review of literature," in *Contemporary Issues in Audit Management and Forensic Accounting*, ser. Contemporary Studies in Economic and Financial Analysis, vol. 102, Feb. 2020, pp. 265–272.

[2] Financial Action Task Force OECD, "What is money laundering," Jul. 1999.

[3] T. Pietschmann *et al.*, "Estimating illicit financial flow resulting from drug trafficking and other transnational organized crimes," *United Nations Office on Drugs and Crime*, Oct. 2011.

[4] Finansinspektionen, "Swedbank får varning och fyra miljarder kronor i sanktionsavgift," Accessed: Feb. 21, 2023. [Online]. Available: https://www.fi.se/sv/publicerat/sanktioner/finansiella-foretag/2020/swedbank-far-varning-och-fyra-miljarder-kronor-i-sanktionsavgift/

[5] Clifford Chance, "Report of investigation on swedbank ab (publ)," Accessed: Feb. 21, 2023. [Online]. Available: https://internetbank.swedbank.se/ConditionsEarchive/download?bankid=1111&id=WEBDOC-PRODE57526786

[6] B. Richardson, D. Williams, and D. Mikkelsen, "Network analytics and the fight against money laundering," *McKinsey & Company*, Aug. 2019.

[7] A. Lindholm *et al.*, *Machine learning, A First Course for Engineers and Scientists*. Cambridge University Press, 2022.

[8] W. Palma, *Time Series Analysis*, 1st ed., ser. New York Academy of Sciences. John Wiley Sons Incorporated, 2016.

[9] I. Fawaz *et al.*, "Deep learning for time series classification: a review," *Data mining and knowledge discovery*, vol. 33, no. 4, pp. 917–963, Mar. 2019.

[10] J. Faouzi, "Time series classification: A review of algorithms and implementations," *Machine Learning (Emerging Trends and Applications)*, Feb. 2022.

[11] O. Raiter, "Applying supervised machine learning algorithms for fraud detection in anti-money laundering," *Journal of Modern Issues in Business Research*, vol. 1, no. 10, pp. 14–26, 2021.

[12] J. Alotibi *et al.*, "Money laundering detection using machine learning and deep learning," *International Journal of Advanced Computer Science and Applications*, vol. 13, no. 10, pp. 732–738, 2022.

[13] P. Tertychnyi *et al.*, "Scalable and imbalance-resistant machine learning models for anti-money laundering: A two-layered approach," in *Enterprise Applications, Markets and Services in the Finance Industry. Lecture Notes in Business Information Processing*, vol. 401, Aug. 2020, pp. 43–58.

[14] R. Ingemann Tuffveson Jensen and A. Iosifidis, "Qualifying and raising anti-money laundering alarms with deep learning," *Expert Systems with Applications*, vol. 214, Mar. 2023.

[15] H. F. Mahmoud, "Parametric versus semi and nonparametric regression models," *International Journal of Statistics and Probability*, vol. 10, no. 2, pp. 90–109, Mar. 2021.

[16] B. Jiang *et al.*, "Xdl: an industrial deep learning framework for high-dimensional sparse data," in *Proceedings of the 1st International Workshop on Deep Learning Practice for High-Dimensional Sparse Data*, Aug. 2019, pp. 1–9.

[17] A. Bagnall *et al.*, "The great time series classification bake off: a review and experimental evaluation of recent algorithmic advances," *Data Mining and Knowledge Discovery*, vol. 31, p. 606–660, Nov. 2017.

[18] A. Dempster, F. Petitjean, and G. I. Webb, "Rocket: exceptionally fast and accurate time series classification using random convolutional kernels," *Data mining and knowledge discovery*, vol. 31, no. 5, pp. 1454–1495, Nov. 2020.

[19] F. Karim *et al.*, "Multivariate lstm-fcns for time series classification," *Neural networks*, vol. 116, pp. 237–245, Aug. 2019.

[20] T. Chen and C. Guestrin, "Xgboost: A scalable tree boosting system," in *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*, Aug, pp. 785–794.

[21] P. Senin, "Dynamic time warping algorithm review," *Information and Computer Science Department University of Hawaii at Manoa Honolulu, USA*, vol. 855, no. 40, pp. 1–23, Dec. 2008.

[22] M. Löning *et al.*, "sktime: A unified interface for machine learning with time series," *arXiv preprint arXiv:1909.07872*, Sep. 2019.

[23] "Kneighborstimeseriesclassifier," sktime.net, http://www.sktime.net/en/latest/api_reference/auto_generated/sktime.classification.distance_based.KNeighborsTimeSeriesClassifier.html#kneighborstimeseriesclassifier, (accessed May 29, 2023).

[24] "Rocket," sktime.net, https://www.sktime.net/en/stable/api_reference/auto_generated/sktime.transformations.panel.rocket.Rocket.html, (accessed May 29, 2023).

[25] "sklearn.linear_model.logisticregression," scikit-learn.org, https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html(accessed May 26, 2023).

[26] A. Defazio, F. Bach, and S. Lacoste-Julien, "Saga: A fast incremental gradient method with support for non-strongly convex composite objectives," *Advances in Neural Information Processing Systems*, vol. 27, 2014.

[27] L. R. Medsker and L. C. Jain, "Recurrent neural networks," *Design and Applications*, vol. 5, 2001.

[28] N. Srivastava *et al.*, "Dropout: a simple way to prevent neural networks from overfitting," *The journal of machine learning research*, vol. 15, no. 1, pp. 1929–1958, Jun. 2014.

[29] J. Hu, L. Shen, and G. Sun, "Squeeze-and-excitation networks," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, Jun. 2018, pp. 7132–7141.

[30] L. Bottou, "Stochastic gradient descent tricks," *Neural Networks: Tricks of the Trade: Second Edition*, pp. 421–436, 2012.

[31] X. Luo *et al.*, "Efficient and high-quality recommendations via momentum-incorporated parallel stochastic gradient descent-based learning," *IEEE/CAA Journal of Automatica Sinica*, vol. 8, no. 2, pp. 402–411, Feb. 2020.

[32] M. Abadi *et al.*, "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015, https://www.tensorflow.org/.

[33] DMLC, Distributed Machine Learning Community., "Xgboost documentation." 2022, accessed May 28, 2023. [Online]. Available: https://xgboost.readthedocs.io/en/stable/index.html

8 Appendix

The ROC curves for the methods involving the MLSTM-FCN model, corresponding to three sampling periods, 1 day, 2 days, and 3 days, are presented in Figures 8, 9, and 10, respectively. The ROC curves of the methods involving the ROCKET model can be seen in Figures 11, 12 and 13 for the three respective sampling periods of 1,2 and 3 days.

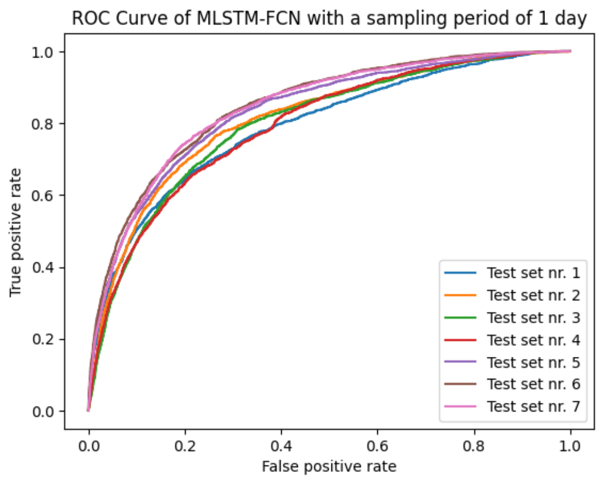


Figure 8: ROC curve for the MLSTM-FCN model with a 1-day sampling period.

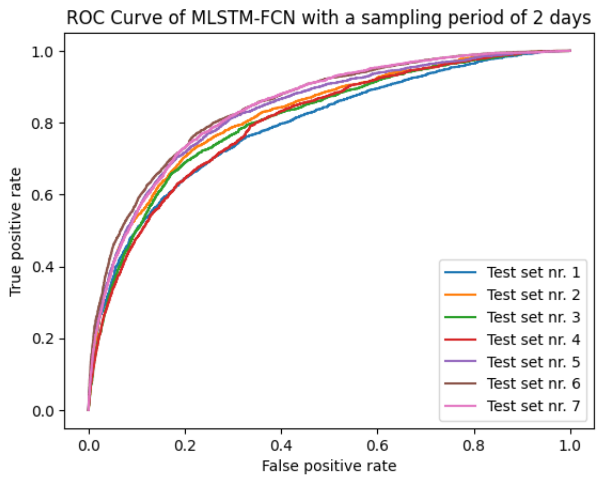


Figure 9: ROC curve for the MLSTM-FCN model with a 2-day sampling period.

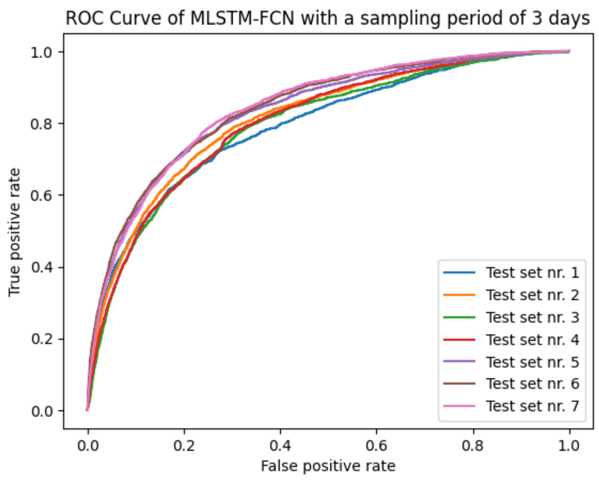


Figure 10: ROC curve for the MLSTM-FCN model with a 3-day sampling period.

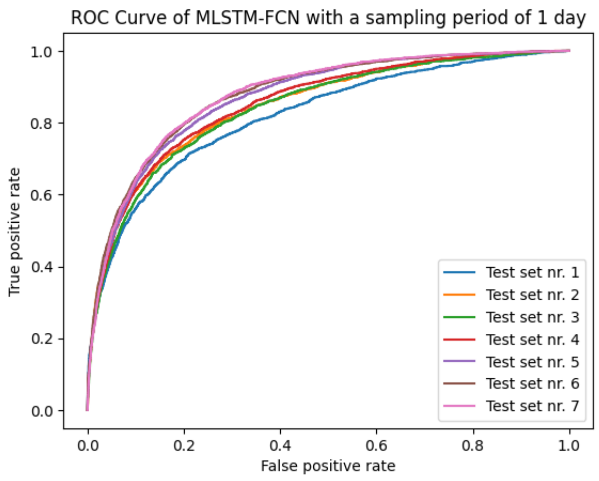


Figure 11: ROC curve for the ROCKET model with a 1-day sampling period.

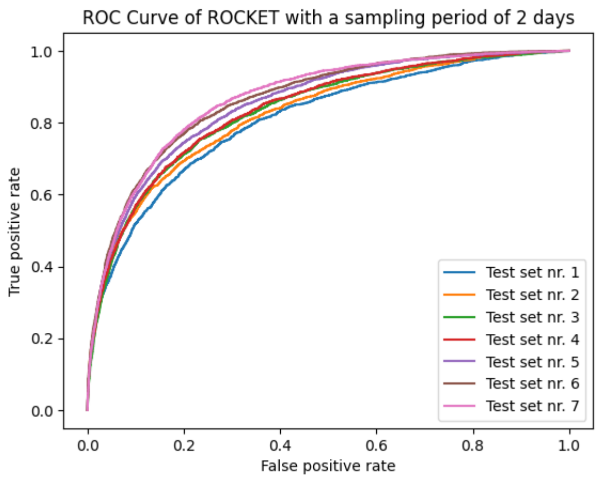


Figure 12: ROC curve for the ROCKET model with a 2-day sampling period.

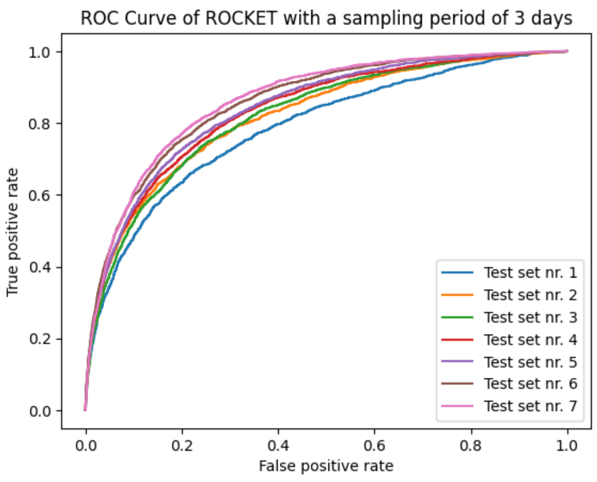


Figure 13: ROC curve for the ROCKET model with a 3-day sampling period.