UPPSALA UNIVERSITET

Examensarbete 30 hp
6 juni 2021

Interpretable serious event forecasting using machine learning and SHAP

Sebastian Gustafsson

Civilingenjörsprogrammet i informationsteknologi

Computer and Information Engineering Programme

UPPSALA
UNIVERSITET

Institutionen för informationsteknologi

Besöksadress:
ITC, Polacksbacken
Lägerhyddsvägen 2

Postadress:
Box 337
751 05 Uppsala

Hemsida:
http:/www.it.uu.se

Abstract

Interpretable serious event forecasting using machine learning and SHAP

Sebastian Gustafsson

Accurate forecasts are vital in multiple areas of economic, scientific, commercial, and industrial activity. There are few previous studies on using forecasting methods for predicting serious events. This thesis set out to investigate two things, firstly whether machine learning models could be applied to the objective of forecasting serious events. Secondly, if the models could be made interpretable. Given these objectives, the approach was to formulate two forecasting tasks for the models and then use the Python framework SHAP to make them interpretable. The first task was to predict if a serious event will happen in the coming eight hours. The second task was to forecast how many serious events that will happen in the coming six hours. GBDT and LSTM models were implemented, evaluated, and compared on both tasks. Given the problem complexity of forecasting, the results match those of previous related research. On the classification task, the best performing model achieved an accuracy of 71.6%, and on the regression task, it missed by less than 1 on average.

Extern handledare: Carl Barck-Holst
Ämnesgranskare: Salman Toor
Examinator: Lars-Åke Nordén

Sammanfattning

Exakta prognoser är viktiga inom flera områden av ekonomisk, vetenskaplig, kommersiell och industriell verksamhet. Det finns få tidigare studier där man använt prognosmetoder för att förutsäga allvarliga händelser. Denna avhandling syftar till att undersöka två saker, för det första om maskininlärningsmodeller kan användas för att förutse allvarliga händelser. För det andra, om modellerna kunde göras tolkbara. Med tanke på dessa mål var metoden att formulera två prognosuppgifter för modellerna och sedan använda Python-ramverket SHAP för att göra dem tolkbara. Den första uppgiften var att förutsäga om en allvarlig händelse kommer att ske under de kommande åtta timmarna. Den andra uppgiften var att förutse hur många allvarliga händelser som kommer att hända under de kommande sex timmarna. GBDT- och LSTM-modeller implementerades, utvärderades och jämfördes för båda uppgifterna. Med tanke på problemkomplexiteten i att förutspå framtiden matchar resultaten, de från tidigare relaterad forskning. På klassificeringsuppgiften uppnådde den bäst presterande modellen en träffsäkerhet på $71,6\%$, och på regressionsuppgiften missade den i genomsnitt med mindre än 1 i antal förutspådda allvarliga händelser.

“Don’t ever prophesy; for if you prophesy wrong, nobody will forget it; and if you prophesy right, nobody will remember it.”

— Josh Billings

Acknowledgment

I want to thank my supervisor Carl Barck-Holst and Daniel Bernhed at West code solutions for their encouraging support and assistance during this thesis, and for allowing me to pursue my master thesis together with them. I want to thank Norah Navér for proofreading and helping me with the report. I also want to thank my reviewer Salman Toor for his guidance and expertise that helped me during this project.

Contents

List of Figures

List of Tables

1 Introduction

Accurate forecasts are vital in multiple areas of economic, scientific, commercial, and industrial activity. Examples include sales for a specific product the coming week, electricity consumption during the coming month, and the coming day's temperature at a specific location. Knowing the future sales of a product can help in production planning, knowing electricity consumption can help in directing more power to the places where the strain will be higher, and knowing the temperature can help an indoor climate system maintain a good indoor climate. The use cases of having accurate forecasts are evident and one area where models, statistical, or machine learning are applied on these types of tasks is called time series forecasting, where predictions are made using data comprised of one or more time series[8].

The type of data required to do time series forecasting is time series data, a set of observations that are made sequentially through time and can be continuous or discrete. The essential characteristic of time series data is that there exists a time linked with each observation and that it is ordered by this[8]. With access to time series data it can be possible to predict the future, and for that, there are multiple different methods. Practitioners in the area of time series forecasting have historically used linear statistical models such as *Autoregressive Integrated Moving Average(ARIMA)*[8]. Still, machine learning models have established themselves as serious contenders to these classical statistical methods[3].

In 2018, professor Spyros Makridakis organized a competition called M4. The purpose of the competition was to identify the most accurate forecasting methods for different types of tasks. Three previous versions of this competition have been held, and for each iteration, the time series data are extended and additional forecasting methods are introduced, as well as new ways of evaluating the performance of these methods. The platform Kaggle[1], which is an online community of data scientists and machine learning practitioners, has hosted similar competitions. Article[4] summarizes and showcases what forecasting methods performed best for six different time series forecasting competitions on Kaggle. The result shows that four of the competitions were won by non-traditional forecasting methods, namely, two machine learning approaches called neural networks and gradient boosting decision trees. These competitions drive the time series forecasting area forward and, give thorough documentation of the performance of different methods on different types of tasks.

Instead of using the classical approach of statistical models for time series forecasting, it has been shown that machine learning is a great alternative. As for statistical models, the objective of machine learning is to discover knowledge from data to make intelligent decisions and, there are a variety of different types of models and algorithms[21]. Machine learning models, sometimes also called black-box models, are examples of non-parametric nonlinear models that use only historical data to learn the stochastic dependency between the past and the future[5]. One sub-domain of machine learning called supervised learning is comprised by algorithms and models that are well suited for time series forecasting. The term supervised refers to a set of training samples, also called input signals, where the desired output signals are already known, also called labels, which means that the correct answer is known. Time series forecasting tasks can rather straightforwardly be re-framed as supervised learning problems, this re-framing gives access to a suite of different supervised learning algorithms that can be used to tackle the problem.

Some machine learning approaches that have been successful at time series forecasting are neural networks and gradient boosting decision trees[4]. These models are black boxes, meaning that there are inputs and output and what happens in between is difficult to understand and derive, which makes them difficult to interpret. This results in models that may perform well on both classification and regression tasks can be hard to interpret or understand how or why they perform well. To make sense of these black boxes, a fairly new field called explainable artificial intelligence has gained recent popularity. Within this area, work is devoted to finding techniques for making these black boxes more interpretable. Today there are a various tools that can help make sense of these machine learning models. One field where machine learning has been applied and where interpretability is essential is healthcare. Article[2] highlights the importance of interpretable machine learning in healthcare with critical use cases involving clinical decision-making. There is some hesitation towards deploying machine learning models that are black boxes because of the potentially high cost of model misclassification. Examples of applications of machine learning in healthcare are predicting a patient's risk of sepsis (a potentially life-threatening response to infection), predicting a patient's likelihood of readmission to the hospital, and predicting the need for end-of-life care. Interpretable machine learning models allows the end-user to understand, debug, and even improve the machine learning system. The opportunity and demand for interpretable machine learning are high, provided that this would allow end-users such as doctors to evaluate the model, ideally before an action is taken. By explaining the reasoning behind predictions, interpretable machine learning systems give end-users reasons to accept or reject predictions and recommendations[2]. Many different approaches for explaining complex machine learning models have been proposed and also implemented. Two popular frameworks that both are model agnostic, which means they can be applied to any machine learning model, are the frameworks SHAP[16] and LIME[24].

There are many examples where time series forecasting quite successfully, has been applied within the commercial and financial field, and the advantages of having accurate predictions within these areas are many. There is, however, not a lot of research where time series forecasting has been applied to predicting serious events before they happen. Having access to accurate predictions whether a severe event will happen in the near future would help in taking precautions to prevent the event from happening. There are many scenarios where this would be of importance, examples include machine malfunction in industries, crime in cities, and traffic accidents. As long as there is historical data of serious events happening, time series forecasting can be applied and, possibly successful at anticipating these serious events. Similar to applying interpretability in the healthcare sector to understand the predictions of the models, having interpretability when receiving predictions of serious events happening is important. Understanding why the model predicts a certain outcome will help decide on whether to disregard it or take action. Can a machine learning model be used for time series forecasting to anticipate serious events, while also being interpretable? These theories are what this thesis will examine.

1.1 Motivation

West Code Solutions(WCS)[34] is a software development company that builds systems for multiple clients. One of their systems, ISAP[30], is an event reporting and handling system used by a multitude of companies and organizations. This system makes it a lot easier for the users to report, handle, follow up, and organize among the events that occur on their premises. Events are graded using a scale where for example, 'Light bulb broke in the cafeteria' would be graded as low seriousness and 'Payment system offline' as high.

Through the years, they have accumulated event data that has happened up until the present time. What WCS would like to see is if they can learn from that particular historical data and use time series forecasting to anticipate when more serious events will happen. The data are confidential and anonymous, access to this data has been granted by an unnamed client of the system to explore the possibility of anticipating more severe events beforehand. Suppose a machine learning model can be trained using this data and accurately predict serious events before they happen, coupled with explanations for each forecast. In that case, that could become one of the best unique selling points for WCS and their event reporting system. Having access to such a risk analysis tool would firstly give the end-user an early warning of a serious event happening, and secondly, indicate on the reasons why. This could help the end-user in taking precautions to prevent the serious event from happening, or reject the prediction.

1.2 Objective and Research Question

The objective of this thesis is to investigate whether it is possible to, through a machine learning implementation of time series forecasting, try and anticipate serious events before they happen. Two different machine learning methods, long short-term memory which is a neural network setup, and gradient boosting decision trees, which is a variation of decision trees and powerful on classification tasks, will be compared on two different tasks of anticipating serious events. Additionally, a model agnostic approach will be taken to explain the predictions made by the models.

Will a machine learning model trained on confidential and anonymous data be able to anticipate serious events accurately? On top of making the predictions, could the system further explain its predictions? Given these problems, the following research question is posed.

Can one successfully predict serious events before they happen and have the system explain those predictions?

1.3 Delimitation

This thesis will compare the performance of two machine learning methods, gradient boosting decision trees and long short-term memory, on the time series forecasting task of anticipating serious events. Statistical methods such as ARIMA will not be tested because of the statement in the paper[7] that a machine learning approach performs better when the amount of data is larger and, because of the review[4] of the Kaggle competitions, where machine learning approaches in the majority outperformed statistical ones. During pre-processing, both one hot encoding and count encoding will be used. An alternative could be to use auto-encoders, but the decision was that this would be unnecessarily time-consuming. To make the predictions interpretable, the model agnostic approach SHAP will be utilised for each model. LIME which is an alternative model agnostic approach could be an option, but because of time constraints the decision is to pick one of them. SHAP is in some sense, such as consistency and accuracy, a similar but improved technique. Lastly, the focus of this project is to implement local explanations for specific predictions, as opposed to global explanations which explain a model in general.

2 Related Work

There are several articles where machine learning models have been used for time series forecasting on different types of tasks. There are also some articles where explainable machine learning has been implemented, although it is a rather new area where a lot is happening. This section will describe the works detailed in a couple of articles related to the thesis and at the end compare and describe how this thesis differs from those related works.

Article[31] investigates whether it could be possible to predict how many crimes will happen in the next day for a certain region in the cities of Chicago and Portland in the United States of America. They used historical crime data augmented with additional datasets containing information on weather, census, and public transportation. During the data preparation, they broke down crime counts into crime count ranges, and then the model predicts the crime count range and for which region in the cities. The model that performed the best had 75.6% accuracy on correctly predicting the overall crime count for Chicago and 65.3% for Portland. In total, they developed four different types of neural networks to try to tackle the task. They tried one simple feed-forward network, one convolutional network which is usually used to handle image data, one recurrent network using long short-term memory, and lastly a combined convolutional and recurrent network. During training, walk forward validation was used, which is the process of selecting a range of values in the time series data as inputs and a subsequent range as output. During the evaluation, they tested the models on the complete dataset and subsets where the data was partitioned by crime type. The best performing model was the combined recurrent and convolutional network which had the best accuracy on both the full dataset and the subset for both cities. Lastly, they tried excluding some of the augmented datasets to see which of the datasets had the most influence on the accuracy of the models. Although all the augmented datasets had a positive influence, it turned out that the census data was the most influential with an accuracy drop of 4.1% when excluded.

Article[26] examined the possibility of using machine learning to predict the accident severity from traffic accidents on the North-East expressway in Malaysia. In this study, they used data containing 1130 accident records that all had occurred on the North-East Expressway in Malaysia over a six-year period, between 2009 and 2015. The data were provided in Excel spreadsheets where one file contained the accident frequency and one contained accident severity. The accident frequency file contained the positional and descriptive accident location and the number of accidents in each road segment of 100m. The accident severity file contained general accident characteristics such as accident time, road surface and light conditions, collision type, and the reported cause of the accident. The two datasets were merged using a unique accident number. With this data, the mission was to train a model to predict injury severity for coming accidents. The model with the best performance was the one that utilised recurrent neural networks. It achieved a validation accuracy of 71.77%. They also examined what factors contributed the most to severe injuries using a profile-based method. This technique examines the evolution of each factor on a scale of values, while the remaining factors keep their values fixed. Using this, they could explain that the model predicted a certain outcome because of the features being of a certain value. The recurrent neural network was comprised of five layers, namely, an input layer, a long short-term memory layer, two dense layers, and a softmax layer. They did a thorough analysis to find the best setup, optimization method, and hyperparameters. They compared the recurrent neural network to a multilayer perceptron and a Bayesian logistic regression model, but in the end, the recurrent neural network had the best performance.

Article[36] examined the possibility of predicting passenger flow on subways using smart card data in Jinan, China. They implemented multiple machine learning models for investigating short-term subway ridership prediction considering bus transfer activities and temporal features. They picked three subway stations in Beijing, and short-term subway ridership, as well as the number of alighting passengers from its nearby bus stops, were obtained based on transit smart card data. They used a method proposed in the paper[12] to estimate the number of passengers transferring from busses to the subway. The subway ridership was calculated using the smart card data of passengers entering the station and, both bus and subway ridership was aggregated into 15-minute intervals. They used the past three time steps as input to the models because the current subway ridership has strong correlations with the past ridership within one hour. Temporal factors such as time of day, day, week, and month were also incorporated into the models. They implemented three different models, an ARIMA, a random forest, and a gradient boosting decision trees approach. As it turned out, the gradient boosting decision trees model outperformed all other models in predicting the number of subway passengers. According to the authors, this was because of the advantage of gradient boosting decision trees in modelling complex relations. To determine which factors influenced the predictions of the model the most, they calculated the relative contributions for all three subway stations. What unsurprisingly was the most influential factor in predicting the ridership for the next 15 minutes was the number of passengers in the subway in the previous time step. This was by far the most influential factor, followed by the two factors, the number of passengers alighting at nearby bus stations during the previous time step and time of day.

Article[18] investigated whether SHAP can be used to explain why particular reviews, predicted by a variety of machine learning models, change a customer's purchase decisions. They used real review data on headsets and facial cleanser products that were retrieved from the web-based shopping site Amazon.cn. After some feature engineering on the texts, they had features such as structure, sentiment, writing style, etc. coupled with metadata such as review ratings. The data was labelled as either helpful or not by looking at the number of votes deeming it as helpful by Amazon users. They compared multiple different types of ordinary models, baseline ensemble models, and gradient boosting decision trees models. The implementation that performed best on both the headphone and facial cleanser data was a framework for gradient boosting decision trees called XGBoost which had an accuracy of 77.25% while classifying a review as helpful or not for the headphones and 70.17% for the facial cleanser. The global explanations provided by the framework SHAP, for the best performing model, showed that for the headset classifier the feature that mattered the most was the number of words in a review, and for the facial cleanser data it was the number of feature-opinion pairs in a review. This essentially meant that on the whole, headset consumers were more concerned with the content elaborateness, and the facial cleanser consumers cared more about the evaluation of some attributes of the product. They also examined more specific feature importances, looking at the dependencies between certain features and the classification.

These articles are all related to this thesis in one way or another. In this thesis, the implementation of gradient boosting decision trees and recurrent neural networks with long short-term memory will be tested on a time series forecasting problem. If the model accuracy is high, the explanations become even more important, and for that, the framework SHAP will be used for explaining the predictions of the models.

Individually, none of the related work tries the same thing as this thesis but combined they have inspired certain choices. Long short-term memory recurrent neural networks will be used for time series forecasting which is done in the papers about crime and traffic accident severity forecasting[26, 31]. XGBoost, the gradient boosting decision trees framework will be used on the time series forecasting problem similar to what was done in the paper about subway ridership[36] as a contender to the deep learning method in getting the best performance. Lastly using the SHAP framework for explaining the models, similarly to what was done in the paper about helpful reviews[18].

Essentially, what this thesis is trying to do is to take a fairly complex and unique task of predicting serious events. This by using two machine learning methods that have proven to be successful at time series forecasting, see if it is possible to get good accuracy, and then explain both individual predictions and to a certain extent, the model in general.

3 Background

This section introduces the background relevant to the methods used during the implementation described in Section 4.

3.1 Machine Learning

Machine Learning(ML) has become one of the most exciting technologies of our time. Technology giants such as *Google*, *Facebook*, *Amazon* and *IBM* heavily invest in research conducted in the field of machine learning, and for good reasons. This field has given us applications that are frequently used in our daily life, such as the voice assistant in our phones, product recommendations in webshops and smart spam filters for our emails[28].

Machine Learning is divided into three subcategories and those are *Supervised Learning*, *Unsupervised Learning* and *Reinforcement Learning*. Supervised learning is when one has labelled data which is passed to the ML algorithm for fitting the model. When the model fitting is done, one can make predictions on new unlabelled data. Unsupervised learning, in contrast to supervised learning does not have labelled data, which means that it does not know the right answer beforehand. This subcategory is used for finding hidden structures in order to extract some type of valuable information. One common unsupervised learning method is clustering, which organizes an arbitrary amount of data into clusters(subgroups) where the data points share a certain degree of similarity. Reinforcement learning is a set of machine learning algorithms where the goal is to improve the behaviour of the system through interaction with the environment. This is implemented using a reward signal so that the system knows when it chooses the correct action. This results in the system wanting to learn a sequence of actions that maximizes the reward via an exploratory trial-and-error approach.

As hardware capacity- and speed continuously improves, Deep Learning is a field that has been further researched in recent years. Deep learning can be understood as a subfield of ML that implies training an artificial brain using artificial neurons in layers efficiently. The possibilities of this subfield are many and some applications include speech recognition, self-driving cars and fraud detection.

3.2 Artificial Neural Networks

An Artificial Neural Network (ANN) is a machine learning model that is supposed to mimic the human brain. It is made up of artificial neurons that loosely model the neurons in a biological brain. The artificial neuron represented in Figure 1 takes three inputs, x_1, x_2, x_3, and produces output **z**. The neuron represented by a circle in the figure multiplies each input with a corresponding weight, sums the results and then activates the output with a chosen activation function. This is mathematically represented in Equation 1 where w represents the weights, b the bias, which can be used to shift the output by a constant value, and ϕ the activation function.

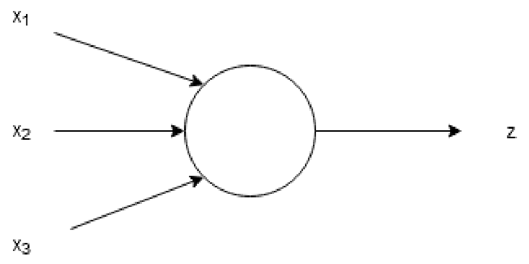


Figure 1 An artificial neuron with several inputs and one output.

$$z = \phi((\sum_{j=0}^{m} w_j x_j) + b) \tag{1}$$

The activation function defines the output given a set of inputs and there are a couple of different activation functions one can choose from depending on the task. Some of the most common ones are Step function(step), Sigmoid(σ), Rectified Linear Units (ReLU), and tanh. These functions are defined in Equations 2, 3, 4 and 5.

$$\text{step}(z) = \begin{cases} 1 & \text{if } z \geq \theta \\ 0 & \text{if } z \leq \theta \end{cases} \tag{2}$$

$$\sigma(z) = \frac{1}{1 + e^{(-z)}} \tag{3}$$

$$\text{ReLu}(z) = max(0, z) \tag{4}$$

$$tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}} \tag{5}$$

Multilayer neural networks, represented in Figure 2, are when there is more than one layer of neurons. In this figure, there are three layers consisting of multiple neurons with interconnections that form layers of computational units. Each neuron is connected to each neuron in the next layer and each neuron in the previous layer using a weighted edge. The first layer is called the *input layer* and the last layer is called the *output layer*, between these layers one can have multiple hidden layers.

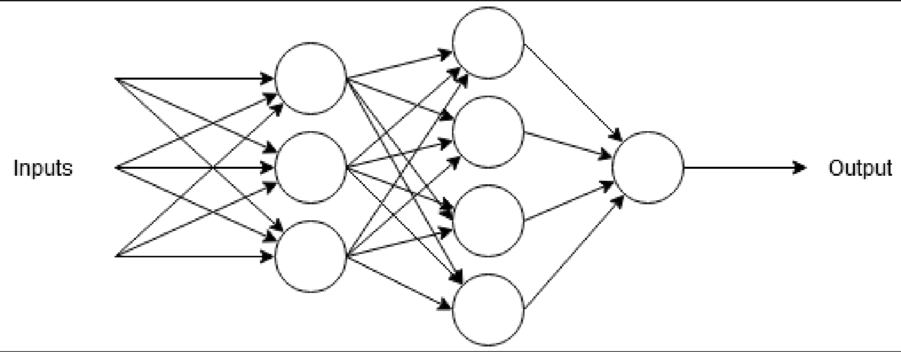


Figure 2 A multilayered artificial neural network consisting of three layers.

One common strategy for determining the difference in predicted output to the actual output is by using the *Mean Squared Error(MSE)* loss function. It calculates the averaged squared distance between predicted- and actual output. How this works mathematically is presented in Equation 6. The $\hat{y}$ is the predicted output and the *y* is the actual output. The *n* is how many samples we will calculate the mean squared error for.

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 \tag{6}$$

When MSE is introduced it proceeds from being a learning problem to an optimization problem since we want to minimize the loss given by MSE. One common optimization technique is *Gradient Descent* and the aim of this algorithm is to update the weights of the network in order to minimize the loss given by MSE. The gradient is determined by calculating the partial derivative of the cost function with respect to each weight. All the weights are then updated simultaneously using the equation listed in Equation 7 where W is the weights, η is the learning rate, which dictates how big of steps the algorithm should take in pursuit of the optimal solution, and L is the loss function.

$$W = W - \eta\frac{\delta L}{\delta W} \tag{7}$$

Since calculating the weight change for each training sample is inefficient, there is a different approach called *Stochastic Gradient Descent* which randomly picks a small batch of training samples that are used to update the weights.

3.3 Recurrent Neural Networks

3.3.1 Standard RNN

Similar to *Artificial Neural Networks(ANNs)*, *Recurrent Neural Networks(RNN)* uses neurons to, given an input, predict an output. ANNs expects the data to be *Independent and Identically Distributed(IID)* which means that the sequence that data are inputted into the network does not matter[28]. One example of an IID dataset is the Iris dataset[35], which contains measurements of iris flowers. These flowers have been measured independently and the measurements of one flower do not influence the measurements of another flower.

RNNs are used when dealing with sequential data, meaning that the order data is fed during training matters. One example could be predicting the market value of a particular stock. If the objective is to predict the stock value for the next three time steps, it would make sense to consider the previous values in a date-sorted order to derive trends rather than feeding the model the training data in a randomized order. RNNs, in contrast to ANNs, are capable of remembering past information and able to take that into account when processing new inputs, which is a clear advantage when dealing with sequential data.

In Figure 3 a comparison between the architectures of ANN and RNN is presented. The difference between them is that with RNN, input and output are bound to a moment in time and that the hidden layer in RNN has a loop. This loop represents the hidden layer receiving input both from the current time step and the output from the previous time step. Using an RNN, this flow of information from adjacent time steps in the hidden layer allows the network to memorize past events. The loop is also known as a recurrent edge in graph notation, which is essentially how the architecture got the name Recurrent Neural Network.

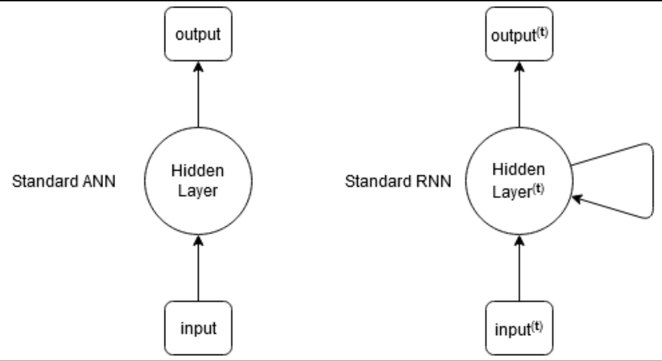


Figure 3 The difference between a neuron in an artificial neural network and a neuron in a recurrent neural network. The difference being the recurring edge.

RNNs are just as ANNs capable of having multiple hidden layers. Figure 4 shows a single layer RNN unfolded. On the left we have the same architecture as in Figure 3, and on the right there is a view of it unfolded where one can see the architecture at different time steps. As previously shown, each hidden unit in the ANN receives only one input, the one from the input layer. In an RNN each hidden unit receives two distinct inputs, the one from the input layer, and the activation from the same hidden layer from the previous time step.

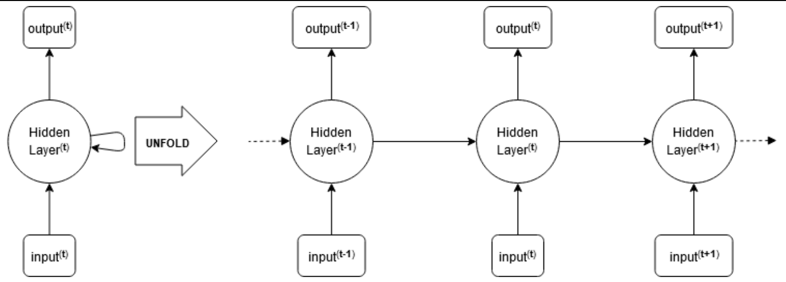


Figure 4 A neuron in a single-layer recurrent neural network unfolded. On the left is the single-layer recurrent network folded and on the right unfolded.

Figure 5 is similar to Figure 4 apart from that it illustrates an RNN with multiple hidden layers. This architecture similarly has the following information flow.

- *Layer 1:* The hidden layer represented as **Hidden Layer**$_1^{(t)}$ receives input from the data point at the current time step **input**$^{(t)}$ and the hidden values from the previous time step **Hidden Layer**$_1^{(t-1)}$
- *Layer 2:* The hidden layer represented as **Hidden Layer**$_2^{(t)}$ receives as input the output of the layer below at the current time step represented as **output**$_1^{(t)}$ and the hidden values from the previous time step **Hidden Layer**$_2^{(t-1)}$

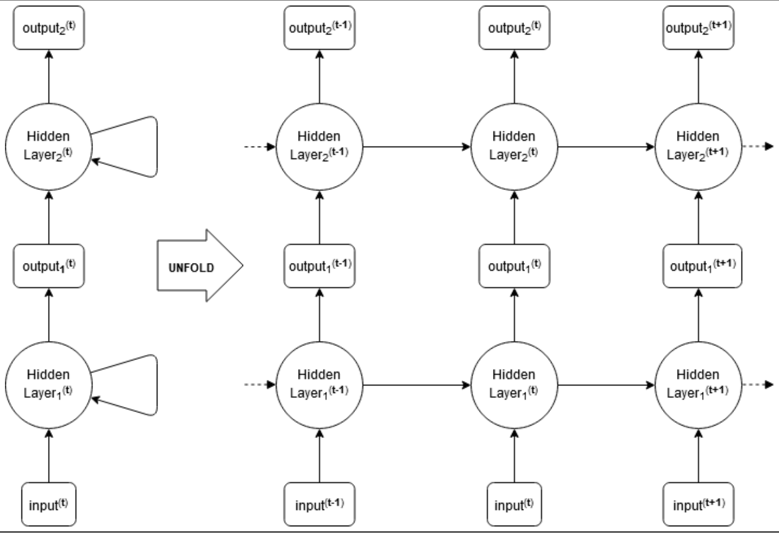


Figure 5 A neuron in a multi-layer recurrent neural network unfolded. On the left is the multi-layer recurrent network folded and on the right unfolded.

Sequence Modelling Types

There are a couple of different sequence modelling types within RNN. Figure 6 illustrates the most popular ones where the *One-To-One* is the most common formation of the regular ANN.

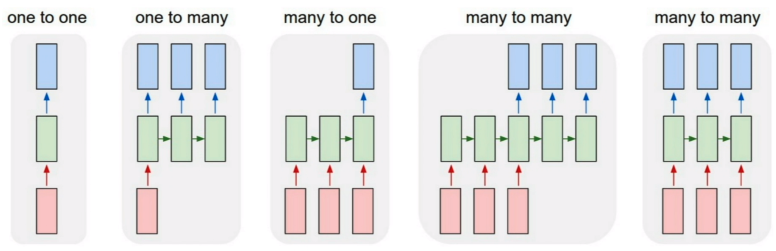


Figure 6 An illustration of different sequence modelling types[6].

One-to-many The input data are singular and the output is a sequence. An example could be having the current temperature as input and predicting the temperature for future three time steps.

Many-to-one The input data are a sequence, and the output is singular. An example could be having the stock value for three time steps back and predicting the stock value for the next time step.

Many-to-many(First variation) The first many-to-many model, takes a sequenced input and outputs a sequenced output. An example could be having the temperature for the past three time steps and predicting the temperature for the future three time steps.

Many-to-many(Second variation) The second many-to-many model similar to the previous model, takes the input as a sequence and outputs a sequence. However, there is output after each time step, one example is models that translate sentences from one language to another in real-time.

Activations

Calculating activations, described in Equation 8, for RNNs is similar to the calculations done for an ANN in that it is a weighted sum of inputs. W_{xh} is the weight matrix between the input $x^{(t)}$ and the hidden layer h. ϕ_h is the chosen activation function, W_{hh} is the weight matrix associated with the recurrent edge and b_h is the bias.

$$h^t = \phi_h(W_{xh}x^{(t)} + W_{hh}h^{t-1} + b_h) \tag{8}$$

Backpropagation Through Time

The learning algorithm *Backpropagation Through Time(BPTT)* for RNNs was introduced in 1990[33]. The basic idea behind calculating the loss is presented in Equation 9 where L is the sum of all the loss functions at times $t = 1$ to $t = T$.

$$L = \sum_{t=1}^{T} L^{(t)} \tag{9}$$

Similar to how the losses calculated by the loss function are summed, the gradients at each time step are summed with Equation 10.

$$W = \frac{\delta L^{(t)}}{\delta W_{hh}} \tag{10}$$

Learning long-range interactions

The learning algorithm BPTT introduces some new challenges. Because of the multiplicative factor in Equation 11, which is used for calculating the gradients of the loss function, the problem called vanishing and exploding gradient arises. t is the current time and k is the starting time step $t - k$ time steps back which is incremented until it reaches t.

$$\frac{\delta h^{(t)}}{\delta h^{(k)}} \tag{11}$$

The problem with BPTT is described in Figure 7 which shows a single-layered RNN for simplicity. In essence, Equation 11 has $t - k$ multiplications, therefore multiplying the weight by itself $t - k$ times results in a factor w^{t-k}. w becomes very small if $t - k$ is large and $w < 1$, which is called *Vanishing*

Gradient, and w becomes very big if $t-k$ is large and $w > 1$, which is called *Exploding Gradient*. Notice that $t-k$ refers to long-range dependencies. The naive solution to this problem would be to make sure that $w = 1$ but there are multiple solutions to this problem, one being a technique called *Long Short-Term Memory*.

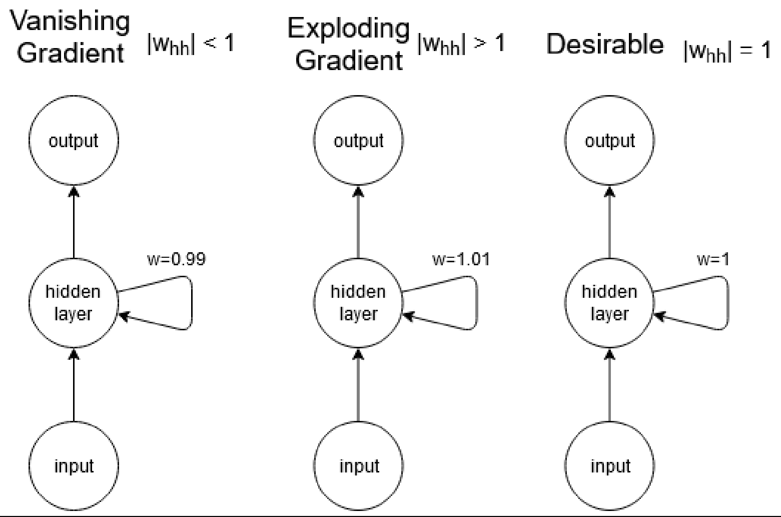


Figure 7 An illustration of the problems with long range learning gradients exploding or vanishing.

3.3.2 Long Short-Term Memory

Long Short-term memory(LSTM) was first introduced in 1997 to overcome the gradient vanishing problem[27]. The building block of an LSTM is a memory cell, which replaces the hidden layer of a standard RNN. The unfolded structure of a memory cell can be viewed in Figure 8 where c_{t-1} is the cell state from the previous time step, h_{t-1} is the hidden units at the previous time step, and x_t is the input at the current time step. In this figure c_{t-1}, the cell state from previous time step, is modified to get the cell state for the current time step c_t without being multiplied directly with a weight factor, which is an attempt at overcoming the exploding and vanishing gradient problem. The circle with an *X* corresponds to element-wise multiplication and the circle with a *plus-sign* refers to element-wise addition. The σ sign indicates the sigmoid activation function, formulated in Equation 3, and $tanh$ is the tanh activation function, formulated in Equation 5.

There are three regions within the memory cell, namely;

- *Forget Gate:* This gate allows the memory cell to reset its state without growing indefinitely. This gate decides what information is allowed to pass through to the current state, and what information to forget.
- *Input Gate:* This gate's main purpose is to update the cell state.
- *Output Gate:* This gate decides how the hidden units are updated.

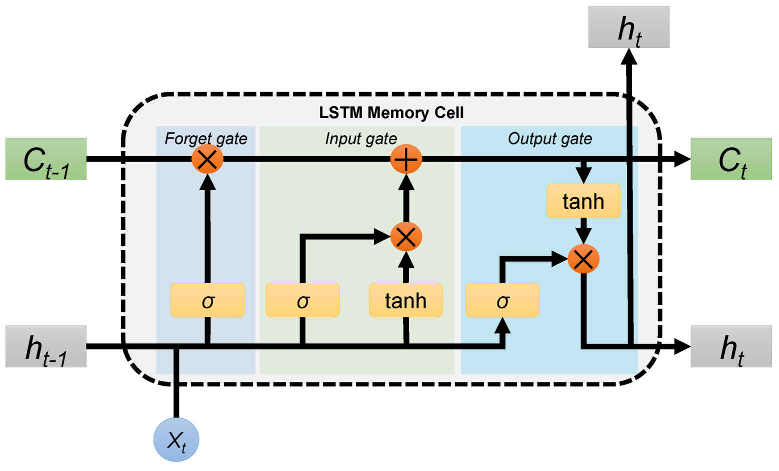


Figure 8 An illustration of a long short-term memory cell and its different gates[11].

There is a more recent alternative strategy to LSTM called *Gated Recurrent Unit(GRU)* which has a simpler architecture and therefore is more efficient.

3.3.3 Gated Recurrent Unit

Similar to LSTM, *Gated Recurrent Unit(GRU)* is a technique for avoiding the vanishing or exploding gradient problem, utilizing gates to decide what information that is passed to the next step. An illustration of the GRU can be seen in Figure 9 and opposed to the LSTM, GRU does not pass on a cell state to the next step, it uses the hidden state to transfer information. This makes GRU faster than LSTM and it allocates less memory. In contrast to LSTM which have three gates, *Forget, Input* and *Output*, GRU has two gates, namely, *Reset(R_t)* and *Update(Z_t)*. On the other hand, LSTM is more accurate on datasets with longer sequences where the additional cell state can be of assistance.

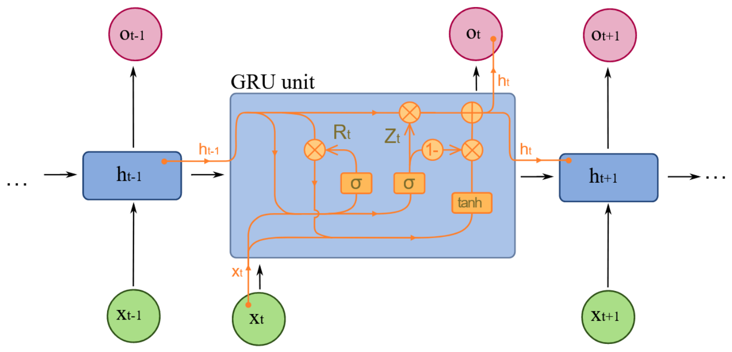


Figure 9 An illustration of a gated recurrent unit for three time steps[10].

3.4 Gradient Boosting Decision Trees

3.4.1 Decision Trees

Decision trees are a flexible type of ML model that can be implemented both for classification and regression. A decision tree uses the typical tree structure of a root node, internal nodes, edges, and leaf nodes. Figure 10 illustrates an example of an already constructed decision tree. In this example, the root node connects to one internal node *Weather?* and one leaf node *Stay in* by the edges with values *yes* and *no*. To classify what to do given this example of a constructed decision tree with data having the features *Work to do?* being *no*, *Weather?* being *rainy*, and *Friends busy?* being *yes* the algorithm would traverse the tree and end up at the classification *Stay in*.

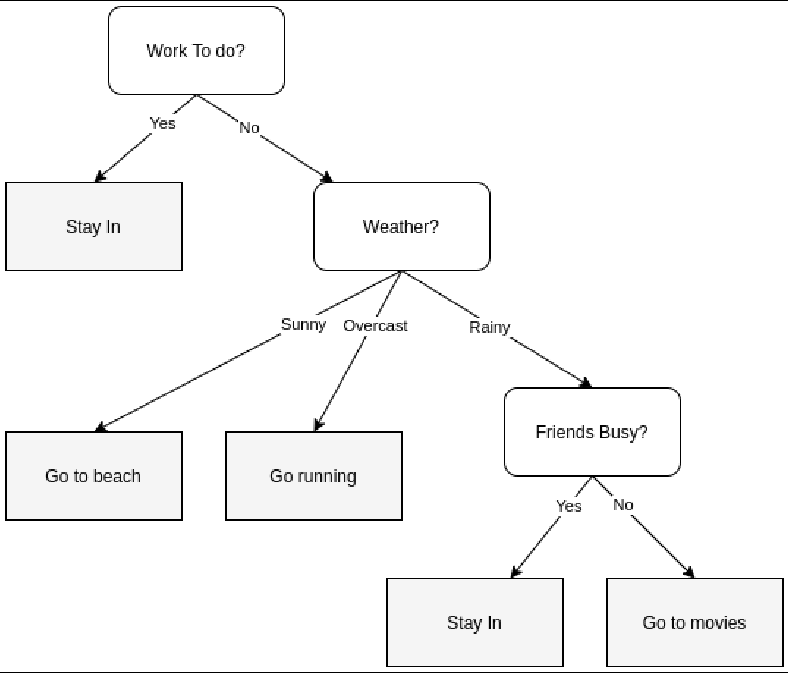


Figure 10 Illustration showing an example of a decision tree.

The example in Figure 10, only uses categorical values when classifying, but it could just as well be numerical values. For example, the *Weather* node could be a binary node with the text *Temperature over 20 degrees celsius* and the outgoing edges *True* or *False*.

Creating decision trees is an iterative process where you split the data on the feature that results in the highest *information gain (IG)*. To split the nodes with the most informative features, the use of an objective function that the tree learning algorithm can optimize is necessary. An objective function for maximizing IG can be seen in Equation 12. In this equation, f is the feature to perform the split, D_p is the dataset of the parent node, D_j is the dataset of the child node, I is the impurity measure, N_p is the number of training examples at the parent node, and N_j at the child node. This means that the information gain is the difference between the impurity of the parent node and the sum of impurities of the child nodes, which means the lower the impurities of the child nodes, the bigger the information gain. For simplicity and to reduce the search space, common libraries implement binary search trees, which means that one node can only have two child nodes, therefore the equation can be simplified into Equation 13.

$$IG(D_p, f) = I(D_p) - \sum_{j=1}^{m} \frac{N_j}{N_p} I(D_j) \tag{12}$$

$$IG(D_p, f) = I(D_p) - \frac{N_{left}}{N_p} I(D_{left}) - \frac{N_{right}}{N_p} I(D_{right}) \tag{13}$$

Impurity is a measure of how pure a node is, which essentially is a measure of how many of the training examples at each node belong to the same class. Three common impurity measures are *Gini impurity(I_G)*, *entropy(I_H)*, and the *classification error(I_E)*.

I_H, entropy impurity is defined by the function in Equation 14 where $p(i|t)$ is the proportion of examples at node t that belong to class i and c is the number of classes. Therefore, the entropy impurity is 0 if the examples at node t belong to the same class and maximal if there is a uniform class distribution. Entropy impurity attempts to maximize the mutual information in the tree.

$$I_H = -\sum_{i=1}^{c} p(i|t) \log_2 p(i|t) \tag{14}$$

Gini impurity can be understood as minimizing the probability of misclassification and is calculated by the function in Equation 15. Similarly to entropy, the Gini impurity is maximal if the classes are perfectly mixed.

$$I_G(t) = \sum_{i=1}^{c} p(i|t)(1 - p(i|t)) = 1 - \sum_{i=1}^{c} p(i|t)^2 \tag{15}$$

In practice, not much can be gained when evaluating trees using different impurity criteria rather than experimenting with different pruning cut-offs. That is where classification error impurity is useful. Classification error which can be calculated by Equation 16, is a convenient criterion for pruning rather than growing a tree since it is less sensitive to class probability changes.

$$I_E(t) = 1 - max(p(i|t)) \tag{16}$$

The impurity measure is central during construction of decision trees, or the training process, meaning it decides where to make splits, and as a result, highly impact the depth of the tree. Keep in mind that the deeper the tree, the more complex the decision boundary becomes, this can easily result in overfitting, which means that the model is too closely fitted to a specific set of data and does not generalize well. This can result in the model being unreliable for future observations.

3.4.2 Gradient Boosting

Gradient Boosting is a supervised learning method for classification and regression based on the idea of taking a weak learner, running it sequentially, and combining the learners for better performance. A weak learner is a model that performs slightly better than random chance, and one weak learner that is popular to use together with gradient boosting is decision trees.

The gradient boosting algorithm consists of three elements, a loss function to be optimized, a weak learner, and an additive model to add weak learners in pursuit of minimizing the loss function and achieving a strong learner.

- *Loss Function:* Depends on the problem, however, gradient boosting is generic enough so that any differentiable loss function can be used.
- *Weak learner:* Decision trees are commonly used as the weak learner, specifically regression trees that output real values which can be added together, allowing the subsequent model's outputs to be added so that shortcomings in the predictions can be corrected, in the next iteration.
- *Additive Model:* Trees are added one every iteration, and the existing trees are immutable. Gradient descent is the optimization method used to minimize the loss function when adding new trees.

A simplistic illustration of the general idea of how *Gradient Boosting Decision Trees (GBDT)* algorithm that adds trees and tries to minimize the error can be seen in Figure 11.

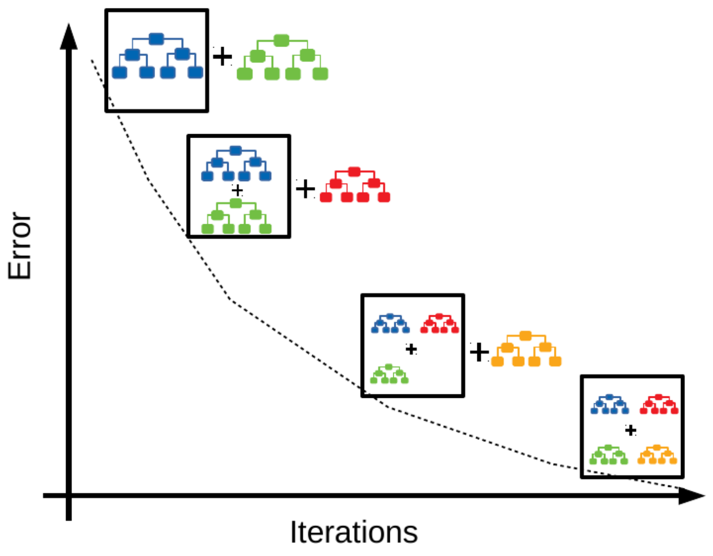


Figure 11 An illustration describing the gradient boosting decision trees algorithm on a high level[32]. For each iteration a new decision tree is added, that aims to rectify the errors of previous trees.

The algorithm for Gradient Boosting Decision Trees is presented in Algorithm 1. It starts by making an initial model $f_0(x)$, this model is what is used in the first round of calculating the gradients, which are then used to fit the regression tree, computing the gradient descent step length, and when updating the model.

Algorithm 1: Gradient Boosting Decision Trees

```
Result: Final Model $f_M(x)$
Initialize $f_0(x) = argmin_p \sum_{i=1}^{N} L(y_i, p)$
for m=1 to M do
    for i=1 to n do
        Compute negative gradients
        $z_{im} = -(\frac{\delta L(y_i, f(x_i))}{\delta f(x_i)})$ where $f = f_{m-1}$
    end
    Fit regression tree $g_m(x)$ to predict the targets $z_{im}$ from covariates $x_i$ for all training data.
    Compute gradient descent step length
    $p_m = argmin_p \sum_{i=1}^{n} L(y_i, f_{m-1}(x_i) + pg_m(x_i))$
    Update model
    $f_m(x) = f_{m-1}(x) + p_m g_m(x)$
end
Return final model $f_M(x)$
```

Gradient Boosting Decision Trees are highly efficient on classification and regression tasks, which can handle mixed types of features, for example, both categorical and numerical without pre-processing. However, careful tuning of hyperparameters is essential and it can be sensitive to outliers.

Two crucial hyperparameters are *learning rate*, which is how fast the model learns, and *number of estimators* which is how many trees or rounds the algorithm is supposed to run for, denoted as M in Algorithm 1. If the *number of estimators* is large, the risk of overfitting the model becomes higher.

3.5 Explainable Machine Learning

Explainable machine learning is about making a machine learning model interpretable. Methods for explaining a model can be grouped into either local or global explanations. Global explanations aid in interpreting the overall model. Meaning which features are more influential for predictions in general. Local explanations are for interpreting individual predictions. There are a couple of different methods of achieving both local and global explanations, and in this chapter two popular ones are presented, namely, *Shapeley Values(SHAP)* and *Local interpretable model-agnostic explanations(LIME).*

3.5.1 Local Interpretable Model-Agnostic Explanations

Local Interpretable Model-Agnostic Explanations(LIME) implements local surrogate models, which are models that are used to explain individual predictions made by a black box machine learning model. LIME tries to explain the predictions by perturbing the input values of one instance and observes how the predictions are affected. The local in the name means that it can be utilised for individual predictions, and model-agnostic means that it can be used for any model. LIME has been used largely in image classification tasks and has helped model creators to get a deeper understanding of why the model made certain predictions.

What LIME does, on a high level, is;

1. Takes the instance of interest that the user wants to have an explanation of, and its black box prediction

2. Perturbs that instance using a distribution and gets the predictions of these samples from the black box
3. Weight the new samples according to their proximity to the instance of interest
4. Trains a weighted, interpretable model on the dataset with perturbed samples
5. Explains the prediction by interpreting the local model

This way you have an interpretable model that explains that local prediction. The authors of *Why should I trust you?*[24] pointed out that local fidelity does not mean global fidelity, which means that important features in one local explanation might not be at all important in the global sense. It is not possible to derive a global explanation from this, and that is one of the things SHAP has tried to remedy.

3.5.2 Shapley Additive Explanation

SHapley Additive exPlanation(SHAP) values are inspired by LIME, and it is an attempt at making consistent interpretations of machine learning predictions. It is based on a unification of ideas from the field of game theory called Shapley Values[29] and local explanations[19]. The general idea behind applying Shapley values to explainable Machine Learning is by assuming each feature value is a ”player” in a game where the prediction is the payout. The formula for calculating Shapley values consists of two parts, calculating *Marginal Contribution(MC)* and then calculating the weight for each marginal contribution.

If the task is to calculate Shapley values for the three features *Serious Event Yesterday(SEY)*, *Day Of Week(DOW)*, and *Power Outage(PO)* used in a model for predicting whether a serious event will happen today, the first thing to do is to create every coalition of features possible and predict the outcome for each one. A visualisation of this can be seen in Figure 12 where each node is a coalition of features. The top row in each node explains which coalition and the bottom row a value representing the prediction outcome of that coalition.

Calculating the Shapley value for *SEY* begins with calculating the MC for that feature which is represented in Equation 17 where the MC between node one and node two is calculated. This calculation is done for every transition to a node where *SEY* is part of the coalition and from a node where *SEY* is not part of the coalition. These transitions are marked by orange arrows in Figure 12.

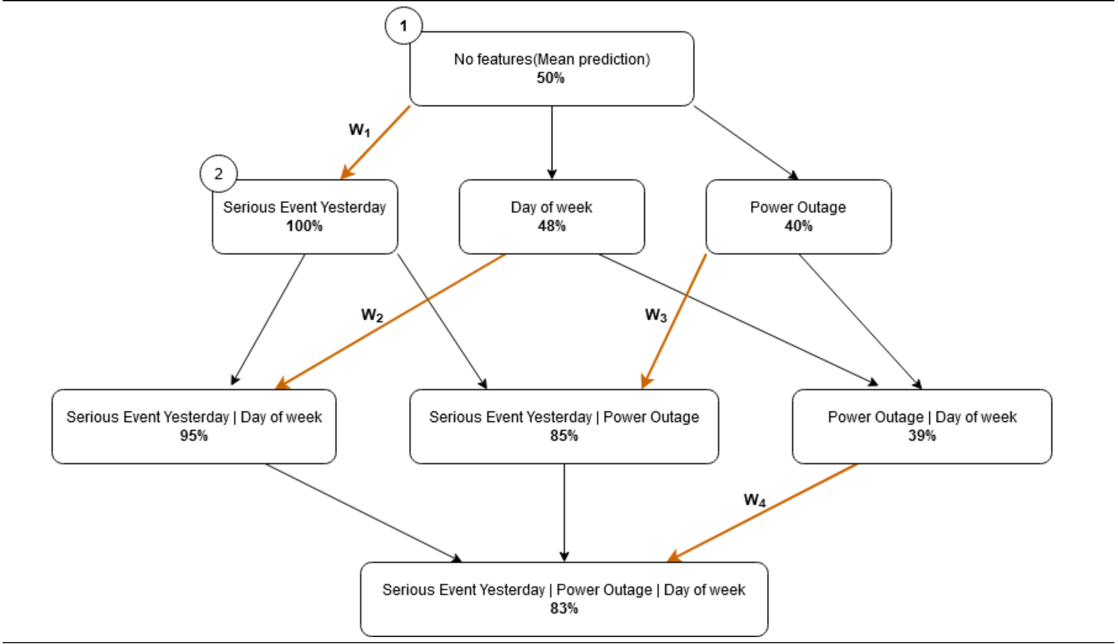

1
No features(Mean prediction)
50%
W1
2
Serious Event Yesterday
100%
Day of week
48%
Power Outage
40%
W2
W3
Serious Event Yesterday | Day of week
95%
Serious Event Yesterday | Power Outage
85%
Power Outage | Day of week
39%
W4
Serious Event Yesterday | Power Outage | Day of week
83%

Figure 12 An illustration showing all different coalitions of three example features in a tree structure.

$$\begin{gathered} MC_{SEY,\{SEY\}}(x_0) = \\ Prediction_{\{SEY\}}(x_0) - Prediction_{\{MeanPrediction\}}(x_0) = 100\% - 50\% = 50\% \end{gathered} \tag{17}$$

The MC value is also weighted by a weight W_x and that weight value is calculated using the formula in Equation 18 where f is the number of features in the coalition, and F is the total number of features.

$$\left[f \times \binom{F}{f} \right]^{-1} \tag{18}$$

Using the MC and weight values, one can calculate the Shapley value for feature *SEY*, as seen in Equation 19.

$$\begin{aligned} SHAP_{SEY}(x_0) = & \left[1 \times \binom{3}{1}\right]^{-1} \times MC_{SEY,\{SEY\}} + \\ & \left[2 \times \binom{3}{2}\right]^{-1} \times MC_{SEY,\{SEY,DOW\}} + \\ & \left[2 \times \binom{3}{2}\right]^{-1} \times MC_{SEY,\{SEY,PO\}} + \\ & \left[3 \times \binom{3}{3}\right]^{-1} \times MC_{SEY,\{SEY,DOW,PO\}} + \\ & = \frac{1}{3} \times (50) + \frac{1}{6} \times (47) + \frac{1}{6} \times (45) + \frac{1}{3} \times (44) \\ & = 46.66\% \end{aligned} \tag{19}$$

The Shapley values for *PO* is -11.33 and *DOW* is -2.33. The sum of all Shapley values gives the difference between the Mean Prediction and the prediction using all features. This is the total effect of each feature being used to predict the outcome.

$$SUM(SHAP) = 33\% = 83\% - 50\% \tag{20}$$

3.6 Data Processing

For any Machine Learning problem, the model will only perform as well as the quality and representation of the data. Data pre-processing are initial steps made to make the quality of the data better. Some common steps are handling of outliers, standardisation, normalisation, and categorical encoding.

3.6.1 Sliding Window method

One popular method used for turning time series data for forecasting into a supervised learning problem is by a method called Sliding Window Method or lag method. This method is simple. Take the data in Table 1, for example, where there are two columns, *Date* and *Temperature*. This data represents the measured maximum temperature for a couple of days in the beginning of the year 2020. To train a model to predict the maximum temperature tomorrow, the sliding window method could be applied. If the task is to given yesterdays temperature, predict tomorrows temperature, the lag factor would be one and Table 2 would be the resulting table. As seen in the table the new column *Temperature_Tomorrow* has the same values as *Temperature_Today* but shifted one row. The first and last row can be omitted since they do not provide any helpful information. Setting $T_{Tomorrow}$ as the output and T_{Today} as the input data one could train a model to predict future $T_{Tomorrow}$ values.

Date	Temperature
2020-01-01	20
2020-01-02	18
2020-01-03	17
2020-01-04	22

Table 1 A table of example temperature data.

Date	Temperature_Today	Temperature_Tomorrow
??	??	20
2020-01-01	20	18
2020-01-02	18	17
2020-01-03	17	22
2020-01-04	22	??

Table 2 Table of example temperature data after applying the sliding window method.

3.6.2 Categorical data

Categorical data are data that represents a class instead of a numerical value. In Table 3 the *Temperature* column is numerical, and the *Wind Direction* is categorical. To be able to train a model using categorical data, you would need to encode it, and two possible methods that can be used are *One Hot Encoding* and *Count Encoding*.

Temperature	Wind Direction
18	North East
19	East
22	South
15	North East

Table 3 Example table of both numerical and categorical data.

3.6.3 One Hot encoding

One hot encoding takes each class in categorical data and sets it as its own column/feature with binary values to indicate its presence. This is a straightforward way to encode categorical values, but the number of columns will grow fast if there are several different classes in the data. An example of One Hot encoding of the data seen in Table 3 can be seen in Table 4.

Temperature	Wind_Dir_North_East	Wind_Dir_East	Wind_Dir_South
18	1	0	0
19	0	1	0
22	0	0	1
15	1	0	0

Table 4 Table with one hot encoded categorical example data.

3.6.4 Count Encoding

Count encoding is, counting the occurrence of a class in the whole dataset and use that value as the encoding for each class. An example of this using the data in Table 3 can be seen in Table 5. This method can be problematic if there are multiple different classes with the same count, as seen in the Table 5, where classes *South* and *East* have the same count encoding.

Temperature	Wind_Dir_count_enc
18	2
19	1
22	1
15	2

Table 5 Table with count encoded categorical example data.

4 Method

The project consists of three main steps: firstly, pre-processing the data for training time series machine learning models; secondly, training and optimizing the models; finally, implementing prediction explainability using SHAP. The initial plan was to proceed in that particular order. However, since the data needed to be different for classification and regression, and to improve performance during the evaluation, revisits to the pre-processing steps were required.

4.1 System Overview

The flow of the system can be seen in Figure 13. The raw data are pre-processed to fit both a classification and a regression problem and is sent to *Pre-processing Classification* and *Pre-processing Regression* respectively. Most of the operations are the same for both datasets, but some operations are done differently. When the pre-processing is done, both datasets are fed into an LSTM and a GBDT model, respectively. When training and validation are done, the results are compared between the models, and some arbitrary predictions are explained using SHAP to get an idea of the reason behind each prediction made by the models.

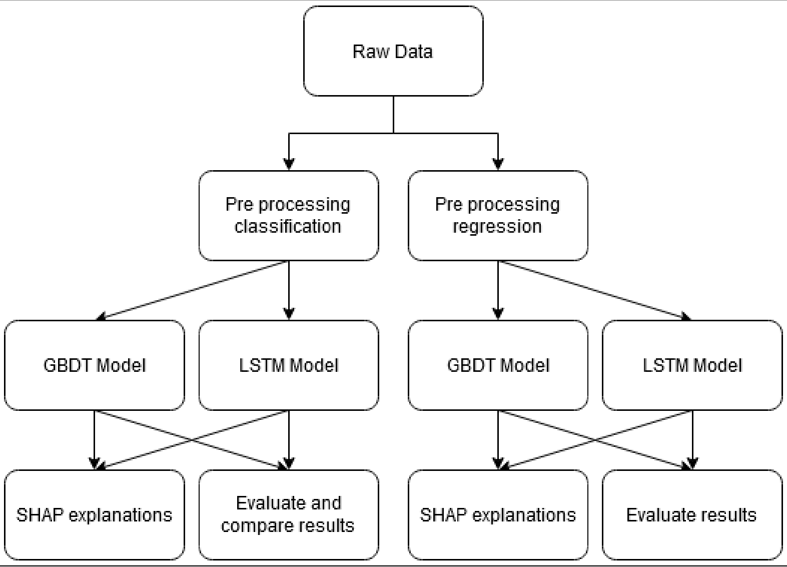


Figure 13 An illustration of the implementation process on a high level.

4.2 Preparing the data

The confidential raw data were provided by WCS which was the only data source used. It was provided in *Comma Separated Values (CSV)* format, see Listing 1 for an example of a CSV file. The top row defines column names for each data entry. The rows after that are data entries where each column is separated by a comma. This format is well established, small in size, and fast to process. However, it does not handle or read easily when there are thousands of rows and thousands of columns.

```
date, seriousness_grading, event_category
2020-11-11, high, breach
2020-11-15, low, malfunction
```

Listing 1: An example excerpt from a CSV file.

To make it easier to read and to do pre-processing operations on, Jupyter Notebook[15] was used. It is a web-based tool for segmenting each step of computing into cells and storing the intermediate results. This means, for example, that the data will not have to be read or imported from the CSV file each time to be able to do an operation. Some alternatives to Jupyter Notebook are the Integrated Development Environments(IDEs) RStudio and PyCharm. Jupyter Notebook is more lightweight and fulfils all the requirements for this project which is why it was chosen.

For common operations such as structuring the data, removing duplicates, and dealing with missing data, the library Pandas[17] was utilised. Pandas support loading data from CSV files into something called a DataFrame which is similar to a matrix or an excel document. This makes it easier to overview the data, compared to the CSV format. Pandas also support a range of different operations that can be executed on the said DataFrame, such as summing, aggregation, etc.

When the data were imported into a DataFrame it became obvious that the data needed pre-processing. There were columns having the same information in different representations and a lot of missing values that had to be either removed or dealt with appropriately. Provided that data from CSV files are missing type definitions, what Pandas do is that it tries to figure out the types with mixed success. Therefore, there was also a need to convert some columns to the correct type.

Having the data imported into a DataFrame in Jupyter Notebook it was easier, compared to CSV, to conduct the operations of pre-processing necessary. In general, the steps that were needed to be done can be seen in Figure 14.

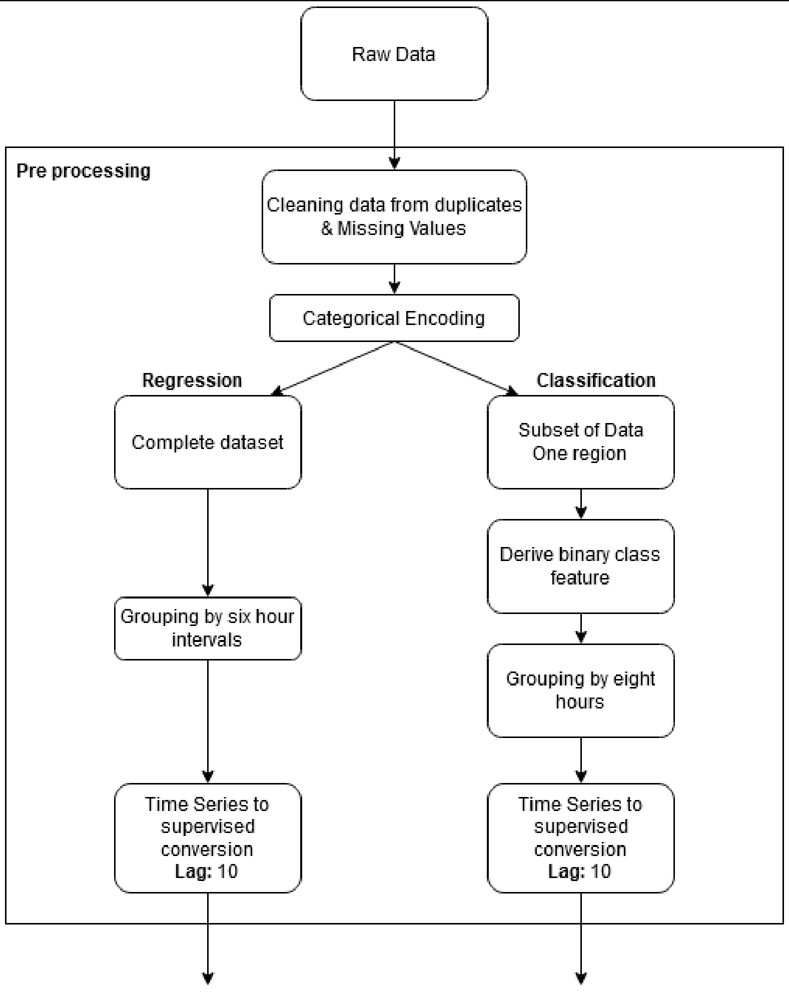

Raw Data
Pre processing
Cleaning data from duplicates & Missing Values
Categorical Encoding
Regression
Classification
Complete dataset
Subset of Data One region
Derive binary class feature
Grouping by six hour intervals
Grouping by eight hours
Time Series to supervised conversion Lag: 10
Time Series to supervised conversion Lag: 10

Figure 14 A high-level overview of the pre-processing steps taken. After the initial three steps the pre-processing splits into regression, and classification specific pathways.

The data were very sparse the first two years, probably because, during this period, the system was not the primary system for event reporting. Since the task is very dependant on finding patterns in sequence data connected to the date and time, these data entries were omitted.

Most of the columns were categorical rather than numerical. The columns having around five different categories or less were encoded using one hot encoding, and the remaining were encoded using count encoding, both described in Section 3.6.

For the LSTM models, both regression and classification, the data were normalized. This is because LSTM models can benefit from having each feature in the same range, otherwise neural networks can weight the features unequally. For GBDT models, which uses decision trees, normalization is not necessary.

The data were provided, each data entry being a reported event. This is a difficult format to train a model for forecasting, and therefore the data were grouped into intervals, for example, all events for every six hours, which is the interval used for the regression task and where an example of the data can be seen in Table 6.

While grouping the data into intervals the categorical data in each column were counted and added as new columns. Now the data had one row for each date and time where there had been reported events, and the columns are counts for each category for that specific day. The gap in the data, where there were no data for a given date and time, was remedied by inserting rows with counts being zero. Since the data are sequential and the project is about time series forecasting, the last step for preparing the data for the models was using the Sliding Window method discussed in Section 3.6.1. This is a method for converting the time series data into a supervised learning data set.

The data prepared for the regression models looks similar to the data in the example excerpt in Table 6 which is data with one hour lag using the Sliding Window Method, grouped by every six hours.

Date	Seriousness_ grading_ high_count(T-1)	Seriousness_ grading_ low_count(T-1)	Seriousness_ grading_ high_count(T)	Seriousness_ grading_ low_count(T)
2020-01-01 00:00	20	18	0	0
2020-01-01 00:06	0	0	15	2
2020-01-01 00:12	15	2	22	13

Table 6 Table showing an example of the pre-processed data for regression.

For the classification problem, a feature for representing whether each row or set of features translated into a serious event, was derived. This was done by checking whether the number of events with high seriousness grading, was more than zero. An example excerpt can be seen in Table 7.

Date	Serious_ Event_ Happens	Seriousness_ grading_ high_count(T-1)	Seriousness_ grading_ low_count(T-1)	Seriousness_ grading_ high_count(T)	Seriousness_ grading_ low_count(T)
2020-01-01 00:00	False	20	18	0	0
2020-01-01 08:00	True	0	0	15	2
2020-01-01 16:00	True	15	2	22	13

Table 7 Table showing an example of the pre-processed data for classification.

4.3 Hyperparameter Tuning

For both GBDT- and LSTM models, many different hyperparameters can be adjusted to adapt the model for the dataset and achieve better performance. One common approach is to perform a grid search to find the parameters of the model that achieve the best performance. An alternative to grid search is random search, and both have advantages and disadvantages. In Figure 15 an illustration of grid search compared to random search can be seen. The figure depicts a grid- and random search done on two parameters, of different importance, with the axes being different parameter values.

Grid search is more or less a brute-force strategy, trying all combinations of parameters, evaluating the model given some scoring that indicates performance, and returning the parameter set that performs best. Random search is similar but takes an arbitrary number of points, or parameter sets, from some distribution within the parameter space at random. In Figure 15, on the left is an illustration of a grid search, and on the right of a random search. The y-axis represents an unimportant parameter which means a parameter that influence the performance less, and in this example the random search method would be preferred. This is because, in this example, random search tests nine different values for the important parameter compared to the grid search, which tests three different values. The chances of finding the optimal parameters are higher for random search in this case. The performance of the model connected to the important parameter is being plotted in green above the grid space.

If there were more than two parameters, for example, four parameters, and twenty different values for each parameter, the grid search method would require to run 160,000 trials. With random search, there is the option of picking a fixed number of points which will be randomly placed in the search space. It could be that the grid search method which is more exhaustive, will find a better set of hyperparameters than random search, but random search could be preferred since it can be more efficient if picking a lower amount of points. There is a trade-off and running both and combining the results could be the best option.

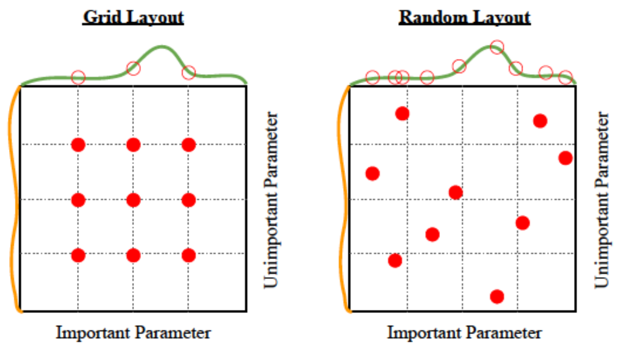


Figure 15 An illustration of the difference between grid search, on the left, and random search[20].

4.3.1 Tuning the models

Scikit-learn's[22] is a Python module that implements multiple useful tools for machine learning such as random search and grid search. Scikit-learn's modules for random- and grid search let the user pick a metric used to evaluate the models during the search.

During tuning for classification, ROC Curve with AUC (Section 4.7.1) was used, and for tuning the regression models, Mean Absolute Error (Section 4.7.4), was used. The parameter space showed in Listing 3 were used for the GBDT models (Section 4.5) and, the parameter space showed in Listing 2 were used for the LSTM models (Section 4.4). Notice that in Listing 2 the parameters are fixed, suitable for grid search, and in Listing 3 parameters are sampled randomly or from distributions, well suited for random search. The parameters in the listings represent;

epochs defines how many iterations of training the model should run.

batch_size defines the size of the batches of training samples that will be propagated through the network.

n_estimators defines how many estimators that should be used during boosting.

learning_rate defines how large steps the optimization function shall take in pursuit of the best solution.

subsample defines the fraction of how much of the complete dataset that should be sampled at each training iteration.

max_depth defines how deep the trees can grow at most.

min_child_weight defines the minimum weight required to create a new node in the tree.

```
param_grid = {
    'epochs' : [10, 50, 100, 200, 300, 400, 500],
    'batch_size' : [10, 20, 40, 60, 80, 100, 128, 256],
}
```

Listing 2: Parameter grid used in optimization for the LSTM models.

```
param_dist = {
    'n_estimators': stats.randint(150, 1000),
    'learning_rate': stats.uniform(0.01, 0.6),
    'subsample': stats.uniform(0.3, 0.9),
    'max_depth': [3, 4, 5, 6, 7, 8, 9, 10, 11 ,12 ,13 ,14 ,15],
    'min_child_weight': [1, 2, 3, 4]
    }
```

Listing 3: Parameter Grid used in optimization for the GBDT models.

4.4 LSTM models

Two LSTM models were implemented, one regression and one classification model. The purpose of the regression model was to forecast the number of serious events that will happen over the next six hours, using the complete dataset containing data for every premise in all regions. The purpose of the classification model was forecasting whether a serious event will occur during the next eight hours for one particular premise.

For the LSTM models (Section 3.3.2) the libraries Keras and Tensorflow[28] were used. Keras is a high level Neural Network library integrated into the Tensorflow package. Tensorflow is a back-end system where data are represented as tensors that can be understood as a generalization of scalars, vectors, matrices, etc. Tensorflow implements a lot of operations that can be done on each tensor, such as applying mathematical operations and various manipulations. Using Tensorflow with Keras makes it relatively easy to model and build rather complex neural networks.

4.4.1 Dense Layer

One of the layers used in the LSTM models is the dense layer. This layer, provided by Keras, is a regular deeply connected network layer. An explanation of the operation executed by the dense layer can be seen in Equation 21 where *activation* can be any activation function, and *dot* is the dot product.

$$output = activation(dot(input, weights) + bias) \tag{21}$$

The reason for using dense layers is for reducing the dimensions and for activating output from the previous layer.

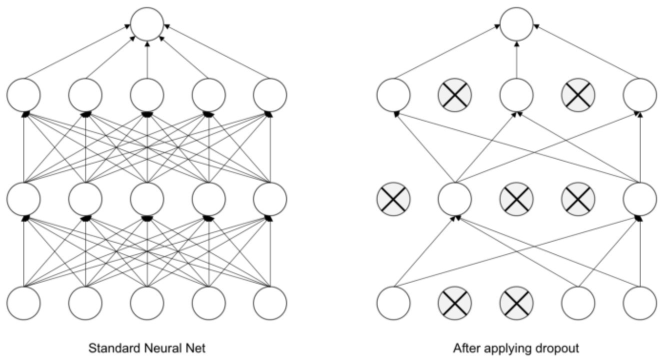


Figure 16 An illustration of a standard neural network and one with dropout applied[25].

4.4.2 Dropout layer

Another layer used in the LSTM model is the dropout layer. This layer, provided by Keras, is a layer for regularizing deep neural networks. Regularization methods are used to avoid overfitting, thus improving the models' ability to generalize. Dropout is applied to the hidden units of the previous layer during training of the network, and it drops a fraction of the hidden units randomly with a probability at each iteration.

When dropping a fraction of the input neurons, the weights associated with the neurons remaining are rescaled to account for the missing neurons. This in effect forces the network to learn a redundant representation of the data and therefore the network cannot rely on an activation of a set of hidden units. In conclusion, since any set of hidden units might be turned off at any time, the network will have to learn the more general and robust patterns of the data. In Figure 16 a visual representation of one iteration of training with and without dropout can be seen.

4.4.3 Regression

The objective with this model was to, using data from the past ten time steps, forecast the number of serious events that will occur over the next six hours on all premises. The model uses the data, normalized, coming from the regression path in Figure 14. The data are split into train and test sets and then passed into the first layer of the model which is an LSTM layer as seen in Figure 17. The second layer is a dense layer (Section 4.4.1), with a Sigmoid activation function to prevent negatively signed predictions and for changing the dimension of the output from the previous layer. The dropout layer (Section 4.4.2), is added at the end to avoid overfitting.

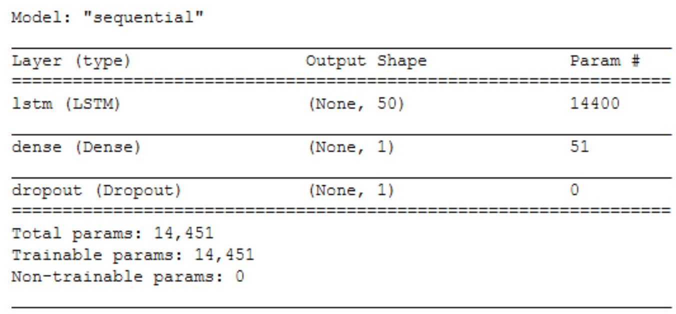
This model was trained using the following hyperparameters and settings, found by using grid search;

batch_size: 128

epochs: 50

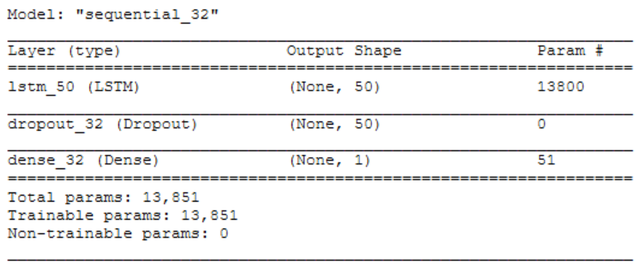
optimization: Adam, which is an optimizer that can be used instead of the common technique stochastic gradient descent, to update weights in the network.

loss function: mean_absolute_error described in Section 4.7.3.



```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm (LSTM)                  (None, 50)                14400
_________________________________________________________________
dense (Dense)                (None, 1)                 51
_________________________________________________________________
dropout (Dropout)            (None, 1)                 0
=================================================================
Total params: 14,451
Trainable params: 14,451
Non-trainable params: 0
_________________________________________________________________
```

Figure 17 The layers used in the LSTM regression model.



```
Model: "sequential_32"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm_50 (LSTM)               (None, 50)                13800
_________________________________________________________________
dropout_32 (Dropout)         (None, 50)                0
_________________________________________________________________
dense_32 (Dense)             (None, 1)                 51
=================================================================
Total params: 13,851
Trainable params: 13,851
Non-trainable params: 0
_________________________________________________________________
```

Figure 18 The layers used in the LSTM classification model.

4.4.4 Classification

The objective of this model was to, given the past ten time steps being eight hours each, classify whether a serious event will occur on one premise in the coming eight hours. The model uses the data, normalized, coming from the classification path in Figure 14. The data are split into train and test sets, and then passed to the first LSTM layer as seen in Figure 18. After the LSTM layer, there is a dropout layer to prevent overfitting and, a dense layer to change the dimension of the output.

This model was trained using the following hyperparameters and settings, found by using grid search;

batch_size: 64

epochs: 20

optimizer: Adam, which is an optimizer that can be used instead of the common technique stochastic gradient descent, to update weights in the network.

class_weight: False : 1 and True : 11. Since there are eleven times more of the class false than the true class.

loss function: binary_crossentropy, a loss function commonly used for binary classification.

4.5 GBDT models

Two GBDT models were implemented, one regression and one classification model. The purpose of the regression model was to forecast the number of serious events that will happen over the next six

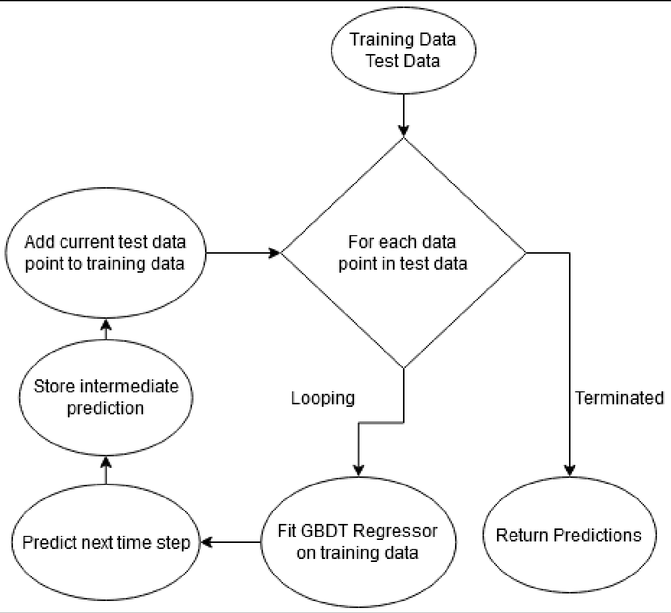

Training Data
Test Data
For each data point in test data
Add current test data point to training data
Store intermediate prediction
Looping
Terminated
Predict next time step
Fit GBDT Regressor on training data
Return Predictions

Figure 19 A flowchart of the walk forward validation algorithm used for regression with GBDT.

hours, using the complete dataset containing data for every premise in all regions. The purpose of the classification model was forecasting whether a serious event will occur during the coming eight hours for one particular premise. There are a variety of different libraries which can be used for implementing GBDT (Section 3.4), but in this implementation the library called XGBoost[9] was used. XGBoost has a simple interface and well-written documentation, and integrates seamlessly with Scikit-learn. It has been widely used for a variety of different problems, making it a versatile implementation of GBDT.

4.5.1 Regression

The objective of this model, the same as the objective of the LSTM model for regression, was to forecast how many serious events that will happen on all premises over the next six hours. It uses the data, not normalized, coming from the regression path in Figure 14. Since GBDT is not usually used for time series forecasting a custom implementation, using walk forward validation in conjunction with GBDT, was introduced. The data are split into train and test sets and then passed into a function which is presented in Figure 19. It loops through all data points in the test set, adds the current test data point to the set for which the GBDT regressor is trained on, predicts the next time step, and stores the intermediate prediction. When all test data have been added to the training set and the last future time step has been predicted, it returns the complete set of predictions which is then used to evaluate performance.

The hyperparameters, indicated through random search, and settings used in the GBDT regression model;

number of estimators: 100

objective: reg:squarederror, meaning regression using squared error as the loss function.

4.5.2 Classification

The objective of this model, the same as the objective for LSTM classification, is to classify whether a serious event will occur on one premise in the coming eight hours. This model receives the data, not normalized, coming from the classification path in Figure 14. The data are split into train and test sets. The XGBoost classifier model is fitted against the train set, and the model is used to make predictions on the test set which returns the predicted class and class probabilities.

The hyperparameters, indicated through random search and settings used with the GBDT classification model;

learning rate: 0.02

scale_pos_weight: 11, because the class distribution being; eleven times more of the false class than the true class.

number of estimators: 1000, but run with early stopping which stops when the performance is not increased.

max_depth: 2

min_child_weight: 4

subsample: 0.59, the fraction of randomly selected samples used to fit each tree.

colsample_bytree: 0.65, the fraction of randomly selected features used to fit each tree.

objective: log:binary, logistic regression for binary classification.

4.6 Implementing Explainability

To explain individual predictions made by the various models the library SHAP[16], (Section 3.5.2), was used. It is a framework for explaining local predictions made by any type of model which makes it useful in this case since one of the machine learning methods is a neural network and the other is based on decision trees. It implements a couple of different explainers, among them DeepExplainer, which is used for deep learning models, and TreeExplainer, which is used for explaining tree models.

4.6.1 Force plots

One of the descriptive plots that SHAP provides is the Force plot. It is a plot that shows which features forces the prediction to be higher or lower. An example of a force plot can be seen in Figure 20. In the figure, the base value represents the predicted outcome if none of the features where used, commonly a mean value of outcomes. The number in bold, where the color red and the color blue intersects, is the predicted outcome or the probability of the predicted outcome. The red bars indicate features that push the prediction higher, and the blue bars represent features that push the prediction lower. For example, the fact that the value of *Feature 0* is zero pushes the prediction higher and the fact that *Feature 19* is zero pushes the prediction lower.

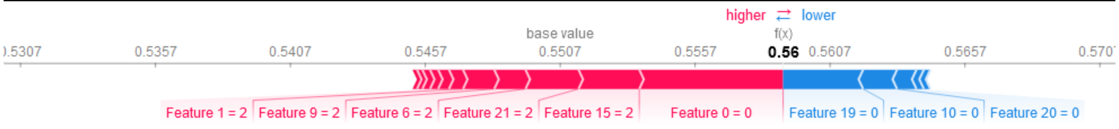


Figure 20 An illustration of a force plot, a tool used for explaining predictions provided by SHAP.

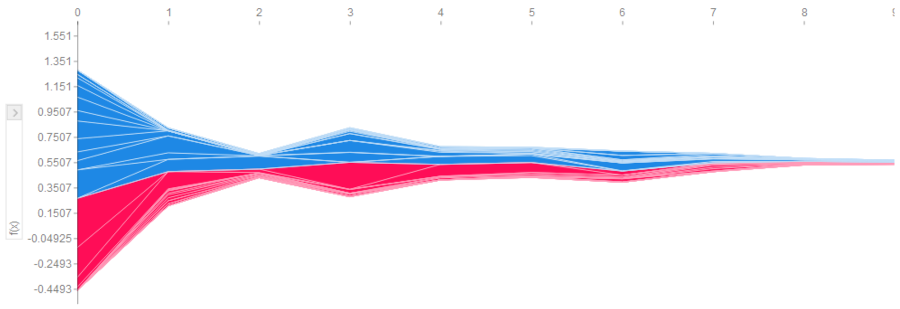


Figure 21 An illustration of a force plot with three dimensional data where the horizontal axis depicts time steps.

4.6.2 DeepExplainer

For explaining the LSTM models a deepexplainer was used. This explainer requires a background dataset upon initialisation, which is the data that are integrated over when calculating the Shapley values. This dataset cannot be too large, because it is a computationally demanding operation. When the explainer is initialised, one can calculate the Shapley values for each feature on a given set of predictions.

The shape of the data used when training an LSTM model is three-dimensional and of the form seen in Equation 22. Instead of having two dimensions where columns are features and rows are samples, it has three dimensions. The first dimension *Rows* is still samples, *Time steps* dimension is every time step for each sample, and *Features* dimension are the features for each time step.

$$(Rows, TimeSteps, Features) \tag{22}$$

When the data inputted into SHAP are three dimensional, the force plots look different. In Figure 21 an example of a force plot when the data have three dimensions can be seen. Here, each time step is plotted against the x-axis, with the forces being plotted against the y-axis. The color red, the same as in the previous force plot, indicates a feature that pushes the prediction higher and vice versa with the color blue. To view one time step in this plot, one would get the same plot as in Figure 20.

4.6.3 TreeExplainer

For explaining the GBDT models a treeexplainer was used. This explainer takes the trained tree model as an argument, then it is possible to calculate the Shapley values for features provided by the test set. Having the Shapley values, the force plot seen in Figure 20 can be produced for any chosen prediction.

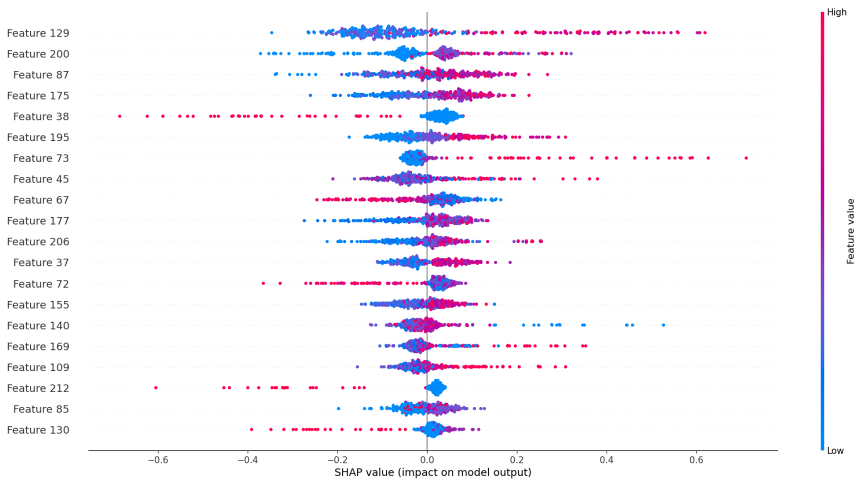


Figure 22 An illustration of a global force summary plot that show the influence of features on a model in general.

4.6.4 Global

To get global explanations of the models, a summary force plot integrated into SHAP, was used. In Figure 22 an example of this plot can be seen. For example for *Feature 129* low feature values, represented in blue, pushes the prediction down, and high feature values, represented in red, pushes the prediction up. Opposed to that, high values of *Feature 38* pushes the predictions down, and low values push it higher, but not by a lot. These plots are helpful to get a general idea of what features the model finds important for its prediction.

4.7 Evaluation Method

For each model a test set containing 30% of the data were used for testing the performance. Different metrics of evaluation were used for classification and regression. For classification *Accuracy* in conjunction with *Geometric mean(G-Mean)* and *Receiver operating characteristic curves(ROC)* were used, and for regression *Mean Absolute Error(MAE)* and *Root Mean Squared Error(RMSE)*.

4.7.1 Receiver operating characteristic curves

One popular metric for evaluating a classification model is the accuracy metric, which is a metric you get from the formula in Equation 23. This metric takes the amount of correctly classified points divided by the total amount of classified points. This metric can be misleading if the classes are unbalanced. For example, if 90% of the data are of the class *False* a model would perform with 90% accuracy if it just classified *False* for each point. 90% accuracy is considered good performance, but a model which classifies one class for each sample is not an intelligent one. This is where the receiver operating characteristic, or ROC, curve can be a helpful addition, as a metric of evaluation.

$$\frac{(TP+TN)}{(TP+TN+FP+FN)} \tag{23}$$

A ROC curve shows how well a model performs in distinguishing between classes, and it does not get influenced by the imbalance of classes. In Figure 23 there is an example of two ROC curves. Along the x-axis the *False Positive rate(FPR)* is plotted, and along the y-axis the *True Positive rate(TPR)* is plotted.

The formula for determining the true positive rate, also called sensitivity, can be seen in Equation 24. This is the percentage of actual positives that are correctly classified. The formula for determining the false positive rate can be seen in Equation 25. This is the percentage of samples that are falsely classified as true.

To draw one point on one of the curves, seen in Figure 23, one sets a threshold value for the prediction probability and calculates the TPR and FPR which is plotted against the axes in the graph. After that the threshold value is adjusted and the calculations repeated. When the curve is drawn, for all the threshold values, one can calculate the Area Under The Curve, or AUC. This is a metric that tells the user the probability of classifying the classes correctly. In Figure 23 there is one blue, one orange, and one dashed line. The blue line is the ROC curve for a GBDT model and it has an AUC value of 0.652 which means 65.2% probability of classifying correctly. The orange line is the ROC curve for a dummy classifier that just guesses during classification, and it has an AUC value of 0.512 which means 51.2% chance of guessing correctly. The dashed line is there to show, similarly to the orange line, a ROC baseline curve of randomly guessing classes.

$$TPR = \frac{TP}{TP+FN} \tag{24}$$

$$FPR = \frac{FP}{TN+FP} \tag{25}$$

4.7.2 Geometric Mean

Geometric mean is the combination of sensitivity, the TPR, and specificity, the complement of sensitivity. The formula for calculating the sensitivity, $S_{sensitivity}$ can be seen in Equation 26 and this gives an idea of how well the model classifies the true class. TP are true positives, and FN are false negatives. The formula for calculating specificity, denoted $S_{specificity}$ can be seen in Equation 27 and this gives an idea of how well the model classifies the false class. TN are true negatives and FP are false positives.

$$S_{sensitivity} = \frac{TP}{(TP+FN)} \tag{26}$$

$$S_{specificity} = \frac{TN}{(FP+TN)} \tag{27}$$

The formula for geometric mean, or g-mean, denoted g can be seen in Equation 28 and this gives a combined score that takes both classified true positives and the true negatives into account.

$$g = \sqrt{S_{sensitivity} \times S_{specificity}} \tag{28}$$

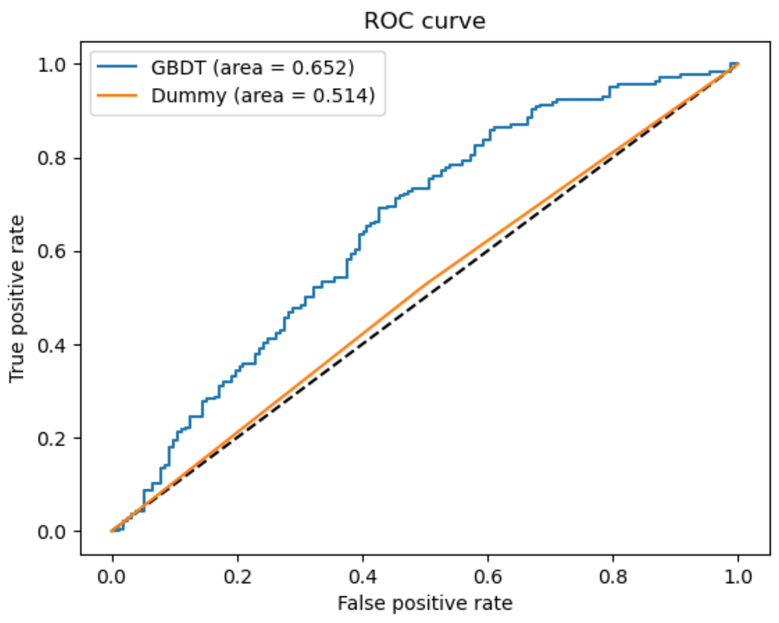


Figure 23 An illustration showing two receiver operating characteristic curves.

4.7.3 Mean Absolute Error

Mean absolute error(MAE), is a common measurement for evaluating regression models. In general, it is the average absolute error, meaning the average error not caring about the sign, the lower the better. MAE is a good metric for evaluation if one does not care too much about outliers in the data. The formula can be seen in Equation 29.

$$MAE = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i| \tag{29}$$

4.7.4 Root Mean Squared Error

One metric that is more suited if one cares about how the model handles outliers in the data, is *Root Mean Squared Error(RMSE)*. RMSE is the squared root of mean squared error, or MSE (Section 3.2). Since MSE returns the square of the error, it is not a metric that is easily evaluated, therefore the square root is applied. The complete formula for calculating RMSE can be seen in Equation 30. As seen in the formula, a higher error means a higher RMSE value. As a result, missing outliers give a higher RMSE value.

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2} \tag{30}$$

5 Result

In this section the results from the implementation of the models, and the explanations from the framework SHAP (Section 4) are presented. The results of the two types of machine learning approaches, GBDT and LSTM, are compared. There are two sections, Section 5.1 where quantitative results, as well as visualisations of the evaluation are displayed and described. Section 5.2 where explanations of the models using SHAP are showcased.

5.1 Model Evaluation

This section presents the results given by the evaluation metrics described in Section 4.7, along with some visual representations of the results.

5.1.1 Regression

The regression models were evaluated using MAE (Section 4.7.3) and RMSE (Section 4.7.4). The prediction outputs given by the regression models are numeric, and is the predicted number of serious events that will happen in the coming six hours. As seen in Table 8, the GBDT model performed, at best, with a MAE of 1.080 and RMSE of 1.469 when evaluating against the test data set, on the regression task. This means that on average it was off by 1.080 while predicting the amount of serious events, while not penalizing for bigger errors, and 1.469 while penalizing bigger errors, or outliers. As seen in Table 8 the LSTM model performed slightly better than the GBDT implementation on both metrics. The decrease of both the MAE and RMSE values with the LSTM model indicates that it performed better both when penalizing for bigger errors and not.

	MAE	RMSE
GBDT Regression	1.080	1.469
LSTM Regression	0.897	1.328

Table 8 Table showing evaluation metrics for both regression models.

The regression models performance can be seen by looking at the metrics in Table 8, but for a visual representation of the results, a line chart of both the GBDT and LSTM version were produced. The GBDT regression model performance on the test data can be seen in Figure 24. In the plot, the x-axis is dates, where each plotted point represents a six hour interval from 2020-05-22 to 2020-07-01. The y-axis the number of serious events, the blue line the historic data that the model is trained on, the green line the actual values in the test data set, and the red line the predicted values. The plot shows that the GBDT model predictions has a lot of spikes trying to predict the outliers, sometimes successfully, but most times it is quite far off. The spikes where the model makes incorrect predictions are penalized more by the RMSE equation, and the calculated value of RMSE is 1.469 as seen in Table 8. This value means that on average it is off by almost 1.5 in the predictions, given that the higher errors are penalized more.

The LSTM model performs better than GBDT as seen in Table 8. Figure 25 shows a line chart for the LSTM model, similar to the visualisation for the GBDT model. The LSTM model generalizes more compared to the GBDT model, as it does not try to predict the outliers. The RMSE metric for the LSTM model is 1.328, and even though it misses a lot of the outliers in the actual or true output, it performs better on average.

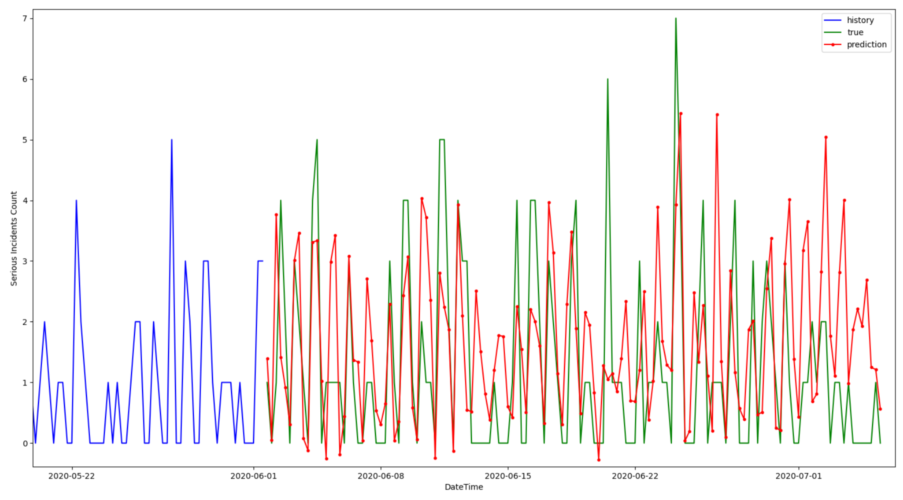


Figure 24 Line chart illustrating the performance of the GBDT model on the regression task.

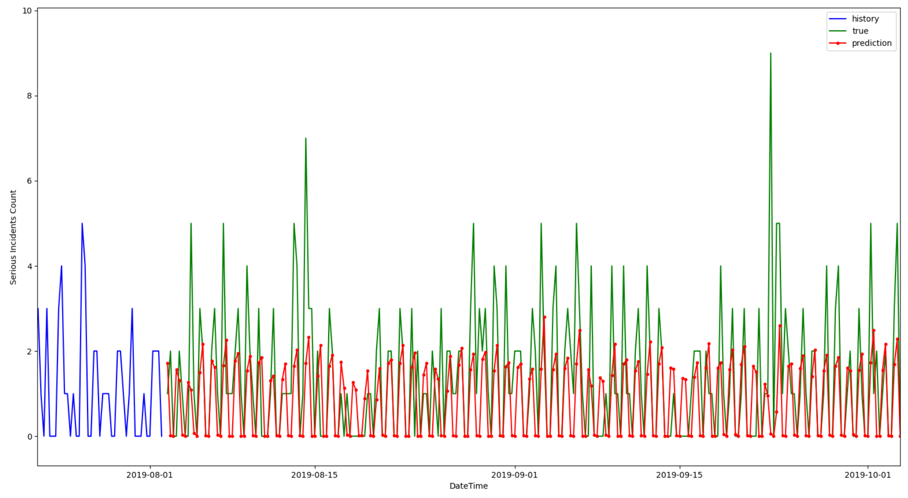


Figure 25 Line chart illustrating the performance of the LSTM model on the regression task.

5.1.2 Classification

The prediction outputs given by the classification models are binary, but it is also possible to retrieve the probabilities of each classification.

The classification models were evaluated using three different metrics, ROC AUC (Section 4.7.1), geometric mean (Section 4.7.2), and accuracy, which is the percentage of correctly classified instances through the total amount of instances. The *Dummy classification* model is a stratified naive classifier that looks at the distribution of classes in the test set and uses that distribution as a probability of classifying either class. It achieves an accuracy of 91.7% which is an excellent performance accuracy wise. However, the percentage of samples with the class false in the dataset is also 91.7%, and therefore, it is evident that it classifies false for all samples. This is also hinted by the values of geometric mean and ROC AUC, in geometric mean if it does not manage to classify any actual true class the geometric mean will be zero.

As seen in Table 9, the LSTM model performs much better than the Dummy one, but the best performance is achieved by the GBDT model. The best performance of the GBDT model has a ROC AUC value of 74.4%, which is at what percentage it manages to distinguish the two classes. The geometric mean score means that on a geometric average, it manages to correctly classify both classes with a percentage of 70.3%, and the accuracy means that 71.6% of all the instances in the test data set were classified correctly.

	ROC AUC	**Geometric Mean**	**Accuracy**
GBDT Classification	74.4%	70.3%	71.6%
LSTM Classification	72.9%	67.3%	69.6%
Dummy Classification	50.0%	0%	91.7%

Table 9 Table showing the evaluation metrics for both classification models.

The confusion matrix seen in Figure 26 shows how the LSTM model performed during classification of the test set. A confusion matrix is a useful tool for visualising how a classifier performs with respect to true- and false positives as well as true- and false negatives. The y-axis where it says *True Label* represents the actual class and the x-axis is the predicted class. The LSTM model manages to correctly predict 158 of the true class, and 1959 of the false class, out of a total of 3039 data points.

The GBDT model has a better performance overall for classification, as seen in Table 9. The confusion matrix for the GBDT model can be seen in Figure 27. It is better at correctly classifying the true class, with 168 true positives and 2009 true negatives. In total, the GBDT model correctly classifies 2177 of the 3039 entries, which results in 71.6% accuracy.

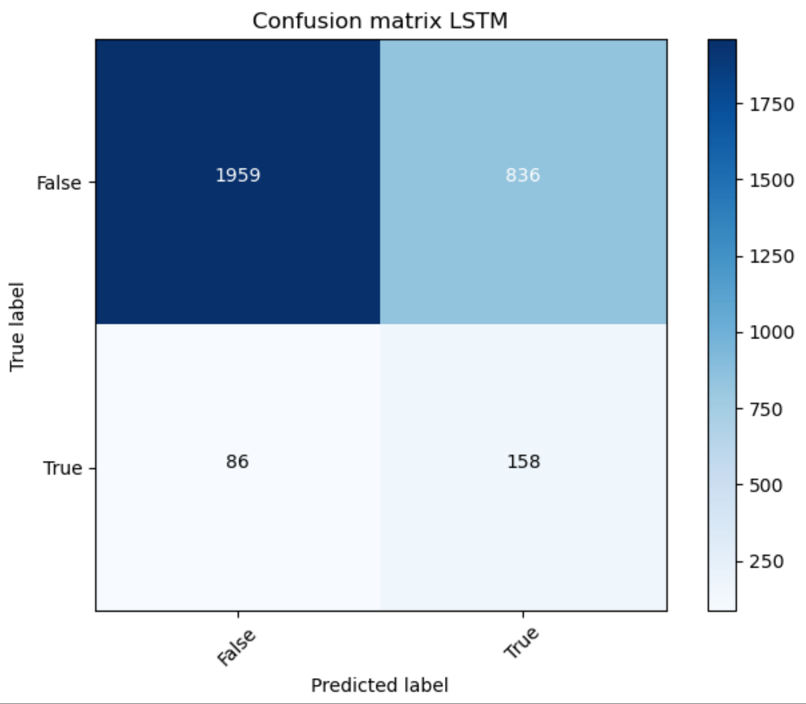


Figure 26 A confusion matrix showing the results from the LSTM model on the classification task.

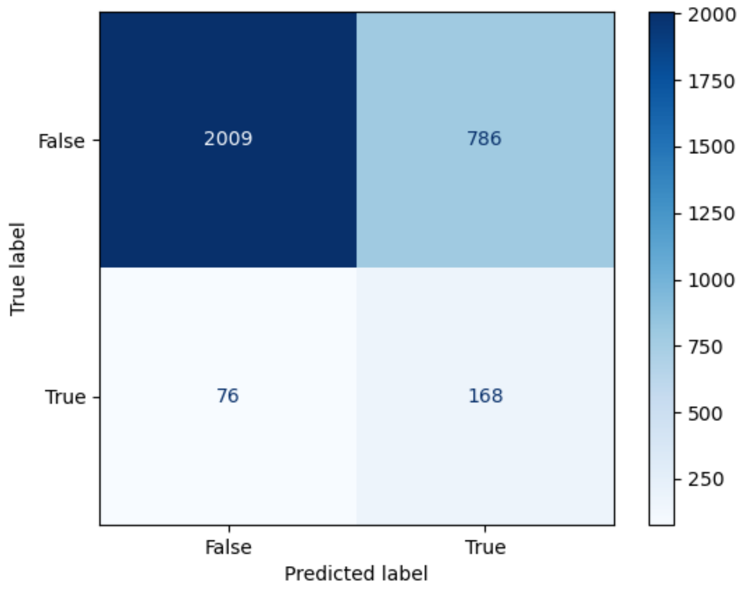


Figure 27 A confusion matrix showing the results from the GBDT model on the classification task.

5.2 Explanations

This section is dedicated to presenting and describing explanations for local predictions given by both the LSTM models (Section 5.2.2), and GBDT models (Section 5.2.1). This section contains force plots to showcase the explanations given by SHAP, for each of the different models and their different prediction outcomes.

5.2.1 GBDT

In this section the explanations for the GBDT models, both regression and classification are showcased.

Regression high prediction According to the explanation of a local prediction seen in Figure 28, the model predicts that four serious events will occur the coming six hours. The force plot shows that the prediction is pushed towards this high value mostly because of the features *SeriousnessGrade_3(t-3)*, which means the number of serious events that happened three time steps ago and, *DaysUntilClosed_sum(t-8)*, which means the sum of days it took for each event to close, that were closed eight time steps ago. These features being 0 and 307 respectively, is what pushes the model prediction towards a higher amount. On the other hand, the feature that pushes the prediction down the most is *SeriousnessGrade_1(t-5)*, which means the number of not serious events that happened five time steps ago. The fact that this feature has the value of 8 makes the model lean towards a lower amount in its prediction.

Regression low prediction In the explanation seen in Figure 29, the model predicts that there will be close to zero serious events the coming six hours. The force plot shows that the prediction is pushed towards this low value mostly because of the features *Closed_1(t-4)*, which means the number of closed events four time steps ago and, *Gender_2(t-4)*, which means the number of events involving a person of gender category two, four time steps ago. These features having the values 4 and 2 respectively, is what makes the model lean towards a lower amount in its prediction. The feature *DaysUntilClosed_sum(t-3)*, which means the sum of days it took for each event to close, that were closed three time steps ago, having the value 166 is what forces the prediction higher, the most.

```
Prediction index:  2
y:  1.0 yhat: 3.771
Sum SHAP + base_val: 3.771 Base_VAL: 1.225
```

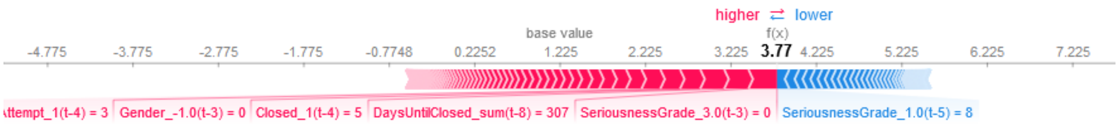
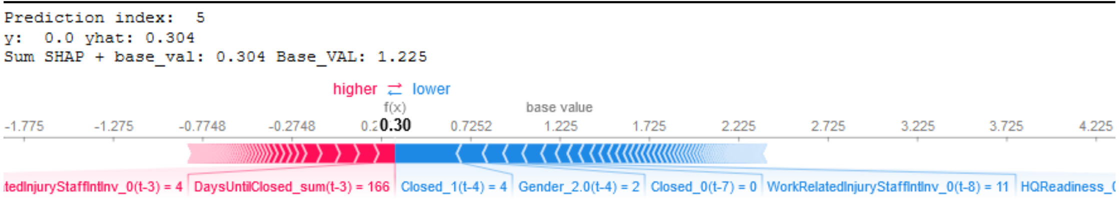


Figure 28 Force plot explaining influential features for the GBDT model on the regression task when predicting a high value.

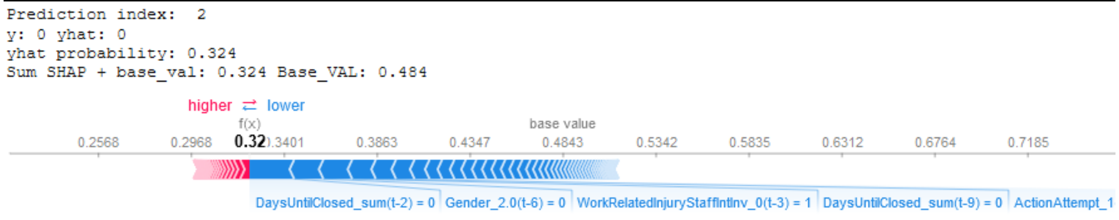


```
Prediction index:  5
y:  0.0 yhat: 0.304
Sum SHAP + base_val: 0.304 Base_VAL: 1.225
```

Figure 29 Force plot explaining influential features for the GBDT model on the regression task when predicting a low value.

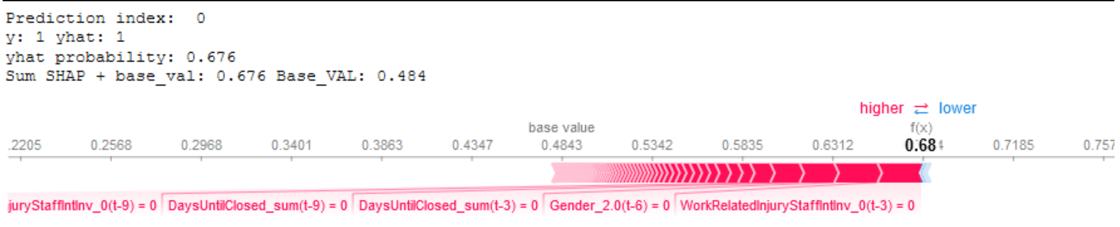
Classification false prediction The explanation shown in Figure 30 concerns a correctly classified false class. The value 0.32 is the probability of the class being true, meaning the probability that the class is false is $1 - 0.32 = 0.68$. The explanation shows that the two main features that increase the probability of the class being false are *DaysUntilClosed_sum(t-2)* which means the number of days it took for the closed events to close two time steps back and *Gender_2(t-6)*, which means the number of events where the gender category was two, six time steps back. The fact that these features have the values 0 and 0 is the main reason why the prediction leans toward classifying false.

Classification true prediction The explanation shown in Figure 31 concerns a correctly classified true class. The value 0.68 is the probability of the class being true. The explanation shows that the features which increase the probability of the class being true, the most, are *WorkRelatedInjuryStaffIntInv_0(t-3)* which means the number of events where an internal investigation concerning work-related injury was not started, three time steps ago and, *Gender_2(t-6)*, which means the number of events where the gender category was two, six time step ago. The fact that these features have the values 0 and 0 respectively, is the main reason why the prediction leans toward classifying true.



```
Prediction index:  2
y: 0 yhat: 0
yhat probability: 0.324
Sum SHAP + base_val: 0.324 Base_VAL: 0.484
```

Figure 30 Force plot explaining influential features for the GBDT model on the classification task when predicting the false class correctly.



```
Prediction index:  0
y: 1 yhat: 1
yhat probability: 0.676
Sum SHAP + base_val: 0.676 Base_VAL: 0.484
```

Figure 31 Force plot explaining influential features for the GBDT model on the classification task when predicting the true class correctly.

5.2.2 LSTM

In this section, the explanations for LSTM models both regression and classification are showcased. For each local explanation, a plot showing the forces during each time step up until the prediction, and a force plot showing the last time step, are displayed.

Regression high prediction The explanation shown in Figure 32 is a force plot, which shows the forces during all time steps up until the prediction. Since there is more red than blue color, the final prediction will be high. The last time step, the ninth, which is one of the most defining time steps for the final outcome, can be viewed in Figure 33. This plot explains a prediction where the model forecast that 2.057 serious events would happen. The features that force the prediction higher are *StaffIntInv_0*, which means the number of events where an internal investigation for finding out if an error were committed by staff was not commenced, in the previous time step, and *Gender_2*, which means the number of events where the gender category were two in the previous time step. These features have the values 2 and 1 respectively, which forces the prediction higher, the most. There are no major features that force the prediction down.

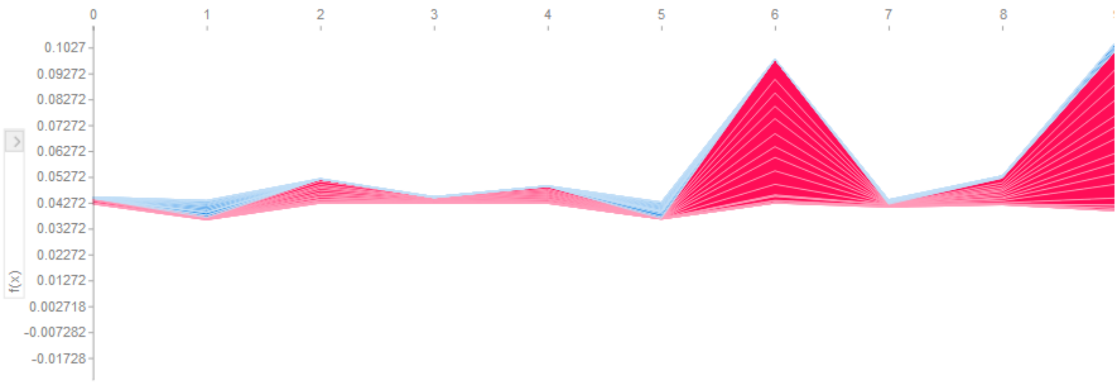
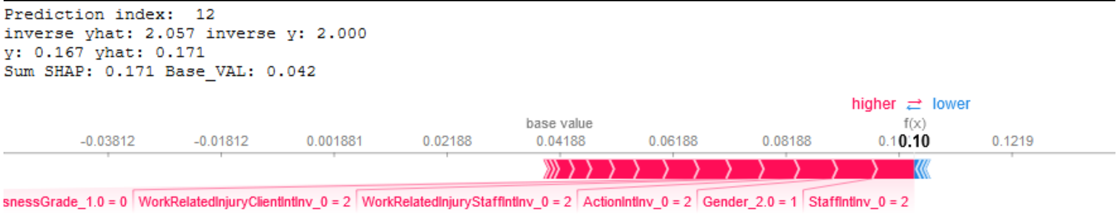


Figure 32 Force plot explaining influential features of the LSTM model on the regression task when predicting a high value, with all time steps.



```
Prediction index:  12
inverse yhat: 2.057 inverse y: 2.000
y: 0.167 yhat: 0.171
Sum SHAP: 0.171 Base_VAL: 0.042
```

Figure 33 Force plot explaining influential features of the LSTM model on the regression task when predicting a high value, for one time step.

Regression low prediction The explanation shown in Figure 34 is a force plot which shows the forces during all time steps until the prediction. Since the color blue is more dominant, the final prediction will be low. The eighth time step, which is one of the defining time steps for the final outcome, can be viewed in Figure 35. This force plot is for a prediction where the output was 0.002. In this force plot it is shown that, in the eighth time step, the features that mainly force the prediction lower are *Gender_2*, the number of events where the gender category was two in the previous time step, and *ActionAttempt_1*, which means the number of events caused by someone in the previous time step. The fact that these features have the values 2 and 3 respectively, is mainly what forces the prediction lower.

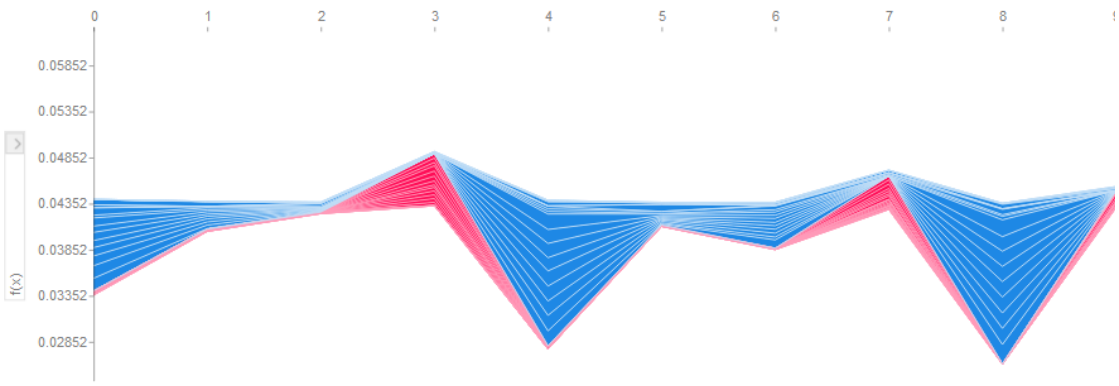
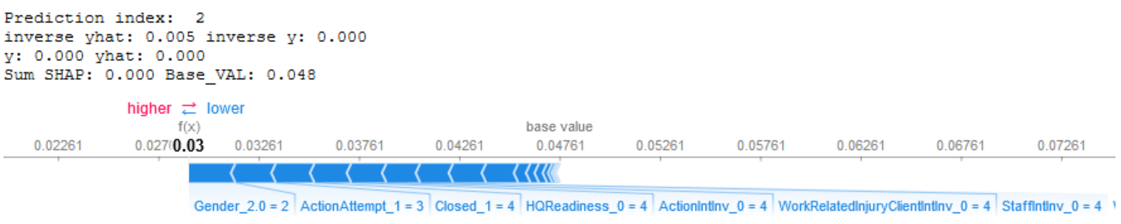


Figure 34 Force plot explaining influential features of the LSTM model on the regression task when predicting a low value, for all time steps.



```
Prediction index:  2
inverse yhat: 0.005 inverse y: 0.000
y: 0.000 yhat: 0.000
Sum SHAP: 0.000 Base_VAL: 0.048
```

Figure 35 Force plot explaining influential features of the LSTM model on the regression task when predicting a low value, the last time step.

Classification false prediction The explanation shown in Figure 36 is a plot which shows the forces during all time steps up until the prediction. Since the color blue is more dominant, the final prediction will be false. The fourth time step, which is one of the defining time steps for the final outcome, can be viewed in Figure 37. This force plot describes a prediction where the output probability of the class being true was 0.35, which means that the probability of the class being false was 75%. In this force plot, the features that mainly force the prediction probability for classifying true lower are *Gender_2*, which is the number of events during the last time step where the gender category was two, and *SeriousnessGrade_1*, the number of not serious events in the last time step. These features, having the values 0 and 0 is the main reason why the model classifies the class false.

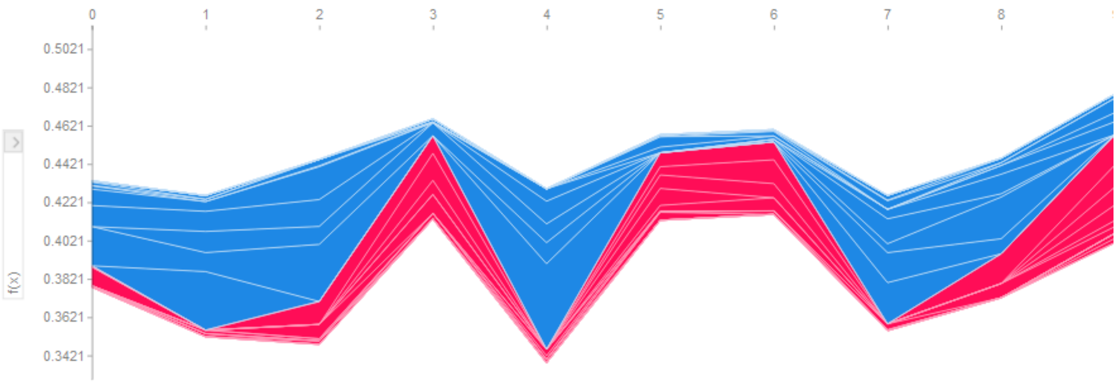
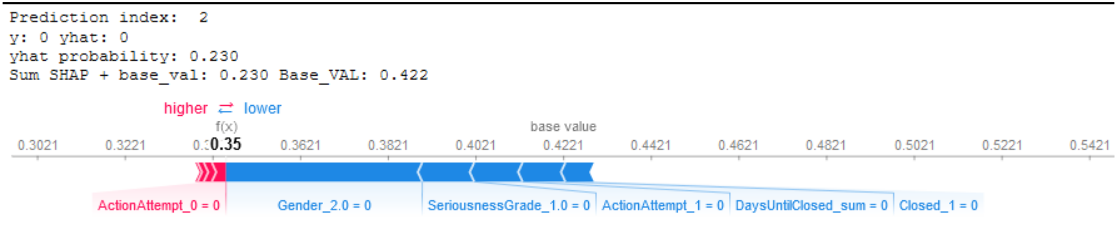


Figure 36 Force plot explaining influential features of the LSTM model on the classification task when predicting a false class, with all time steps.



```
Prediction index:  2
y: 0 yhat: 0
yhat probability: 0.230
Sum SHAP + base_val: 0.230 Base_VAL: 0.422
```

Figure 37 Force plot explaining influential features of the LSTM model on the classification task when predicting a false class, the last time step.

Classification true prediction The explanation shown in Figure 38 is a force plot which shows the forces during all time steps up until the final prediction. Since the color red is more dominant, the final prediction will be true. The seventh time step, which is one of the defining time steps for the final outcome, can be viewed in Figure 39. This force plot describes a prediction where the output probability of the class being true was 0.52. In this plot, the features that mainly force the prediction probability higher are *Gender_2*, which means the number of events in the previous time step where the gender category was two and, *ActionAttempt_1*, which is the number of events in the previous time step that were caused by a person. These values being 0 and 0 respectively, are the main reasons why the model classifies as the class true.

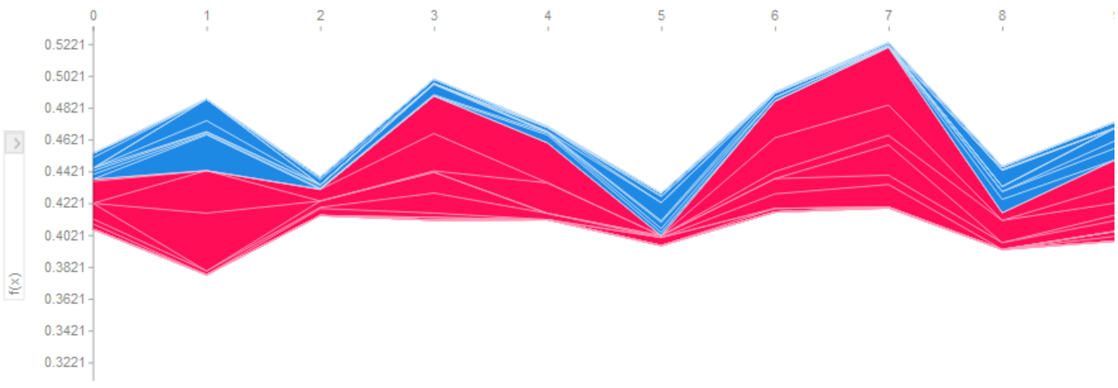
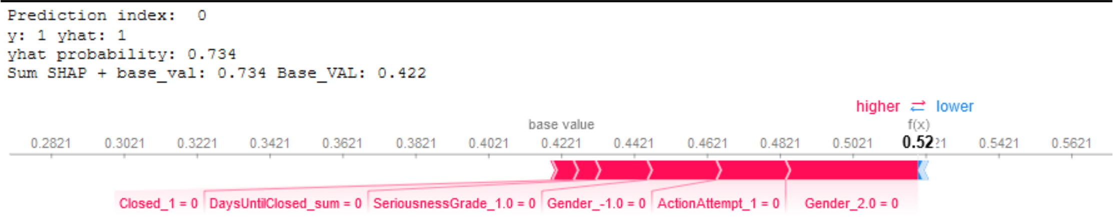


Figure 38 Force plot explaining influential features of the LSTM model on the classification task when predicting a true class, for all time steps.



```
Prediction index:  0
y: 1 yhat: 1
yhat probability: 0.734
Sum SHAP + base_val: 0.734 Base_VAL: 0.422
```

Figure 39 Force plot explaining influential features of the LSTM model on the classification task when predicting a true class, the last time step.

5.2.3 Global explanations

A global explanation describes in general which features that matter the most for a model's predictions. The global explanations for the two best performing models, GBDT for classification and LSTM for regression are showcased in this section. The spread of feature values are more well distributed in the LSTM global explanation and that is because those values are normalized.

In Figure 40 the global explanation for the GBDT model can be seen. It shows the top twenty most influential features and how they affect the outcome. For example, the feature *Gender_2(t-6)*, which means the number of events six time steps ago where the gender category was two. In this plot, most of the dots are plotted on the left side of the center line, and it does not seem to be any segmentation of the colors. This means that regardless of the feature value being high or low, the features usually forces the predicted outcome towards predicting false.

In Figure 41 the global explanation for the LSTM model can be seen. Same as the GBDT global explanation, this shows the top twenty most influential features and how they affect the outcome. For example the feature *WorkRelatedInjuryStaffIntInv_0(t-4)*, which means the number of events four time steps back where an internal investigation concerning a work related injury on staff was not commenced. This feature has a clear segmentation of the colors and a large chunk of blue dots right next to the center line. This means that the model is influenced into predicting a high amount when the feature value is high, rising depending on how high it is. If the feature value is low, it influences the prediction towards a lower amount, be that as it may, a small influence.

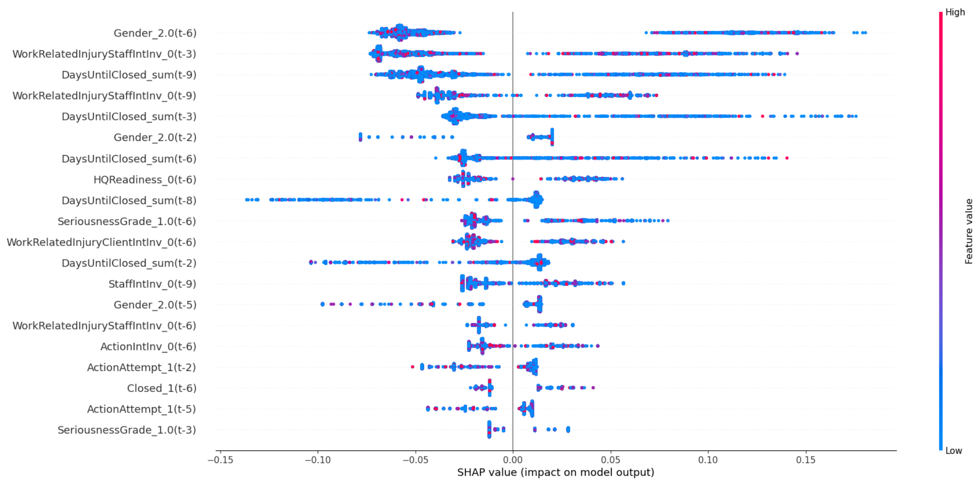


Figure 40 Summary plot showing the globally influential features of the GBDT model on the classification task.

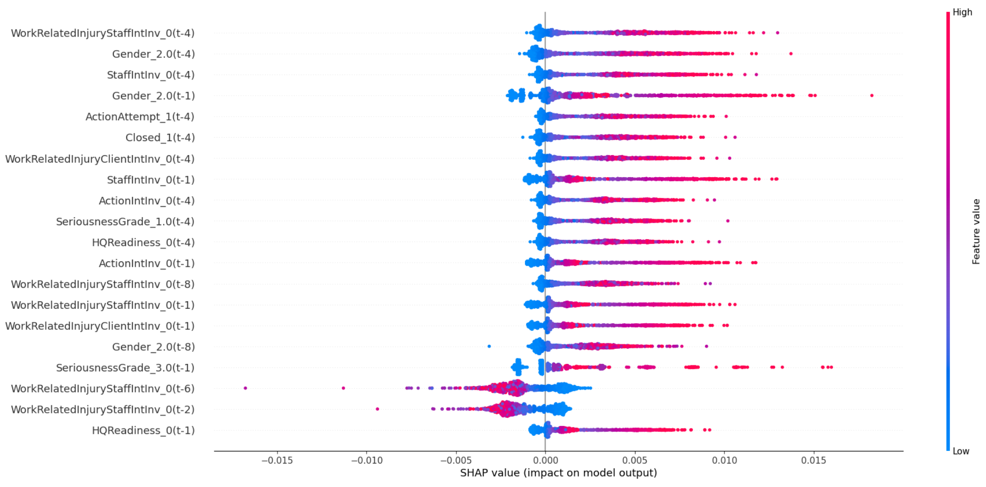


Figure 41 Summary plot showing the globally influential features of the LSTM model on the regression task.

6 Discussion

In this section, the results and performances of the ML models are discussed. The model that performed best on the classification task was the GBDT implementation, and on the regression task it was the LSTM implementation.

The performances of the two best performing models are both interesting. The best performance on the classification task was achieved by, not surprisingly for me, the GBDT implementation. This classification model performs pretty well, equally well in classifying true positives and negatives. The LSTM model is not far behind the GBDT model, but I think that since GBDT is such a powerful ML technique for classification tasks it edges past the LSTM implementation. The LSTM implementation performed, also not surprisingly for me, better than GBDT on the regression task, probably because LSTM is well adapted to time series regression forecasting, as opposed to GBDT. GBDT had problems generalizing, probably because it does not have a memory cell such as the LSTM, which does the job of filtering out the information that will not be needed in the coming time steps.

The SHAP framework is a great tool which helps in understanding why the model makes certain predictions, but it is important to be clear that they do not show causality. If SHAP, for example, shows that the model predicts that a serious event will happen in the coming time step, because of the feature *Gender_2(t-1)* having some value, which is the number of events in the previous time step where the gender category was two, that does not mean that it was the case in real life. However, for a user of the model it can be essential to know on which basis it predicts that a serious event will happen in the coming time step. This way, the user can disregard said prediction if it seems unreasonable.

6.1 Classification task

The best performer for the classification task was the GBDT implementation. This was indicated by the evaluation metrics in Section 5.1. The LSTM model performed slightly worse than GBDT model, both on classifying the actual true and false classes correctly. Probably because GBDT being a powerful method for classification tasks. In this context, there is one class that is more important than the others, and that is the true positives. True positives mean anticipating serious events before they happen and that is naturally the most important class to be able to classify correctly. The ROC curve can help pick a threshold value where the true positive rate is as good as it can be, taking into account how many false positives that are reasonable. It is a trade-off and something the user needs to decide upon.

The accuracy of the model was 71.6%, which means that almost three-quarters of the test data were correctly classified. This is not an exceptional result, but it is an indication that to an extent the model can actually anticipate serious events before they happen. My thoughts beforehand were that the model would find it difficult to distinguish between the two classes, since serious events could happen at random. I had my doubts that a model could find patterns in the data, indicating that a serious event will happen. It seems based on the results produced by this implementation, in 70% of the test samples, it can. It is difficult to reason about why the performance is not higher. Perhaps because there are not enough strong patterns for when serious events happen. Random search was used to find the optimal parameters, but perhaps the most optimal hyperparameters were not found. This would probably not influence the performance by a lot.

Looking at the global explanation for the GBDT model, two of the prominent features from multiple different time steps back, are *Gender_2* and *WorkRelatedInjuryStaffIntInv_0*. The influence of the number of events that occurred where the gender category was two, in real life this would probably not have any implication on whether a serious event would happen in the next eight hours or not. Same goes for the influence of the number of events where an internal investigation for a work-related injury was not launched. However, these are two features that the model commonly find important in distinguishing whether a serious event will happen.

6.2 Regression task

The best performer for the regression task was the LSTM implementation. This was indicated by the evaluation metrics in Section 5.1. This was not very surprising given that LSTM regression commonly is well suited for regression tasks in time series forecasting. It is used for stock market prediction and this task is similar to that, as it tries to predict a numerical value for the next time step, given values a few time steps back.

The LSTM model does on average miss by less than 1 in predicting the number of serious events that will happen in the coming time step, which is decent given that the actual output ranges from zero to nine. It correctly predicts zeros but does not predict outliers well, meaning that the spikes in the actual data are where the model fails to predict close to the exact amount. However, rather a model that generalizes well and that is able to predict the general trend. The performance of the GBDT implementation was a bit worse, but mostly because it failed to generalize like the LSTM. The GBDT model does not have the LSTM memory cell at hand and that could be one reason why. It uses all data up until the point which shall be predicted, while the LSTM version, through the memory cell continuously makes decisions on which data to forget and to pass on, going forward. There could be other reasons why the LSTM model is superior compared to GBDT but I believe this is the most distinguishing.

Looking at the global explanation for the LSTM model, seen in Figure 41 two of the prominent features are *Gender_2* and *WorkRelatedInjuryIntInv_0*. Similarly to the global explanation of the classification model, where these features also were prominent, in real life there would probably not be a connection between these features and the number of serious events happening in the next time step. One explanation that could be reasonable and applicable to real life, is *HQReadiness_0*, which is the number of events where head quarters were not on alert. There could be a connection with this feature and how many serious events that will happen the next time step.

6.3 Conclusion

This thesis set out to investigate two things, firstly whether machine learning models could be applied to the objective of forecasting serious events. Secondly, if the models could be made interpretable. Given these objectives, the approach was to formulate two forecasting tasks for the models and then use the Python framework SHAP to make them interpretable. The first task was to forecast how many serious events that will happen in the coming six hours. The second task was to predict if a serious event will happen in the coming eight hours. GBDT and LSTM models were implemented, evaluated and compared on both tasks. Given the problem complexity of forecasting, the results matches those of previous related research. On the classification task, the best performing model achieved an accuracy of 71.6% and on the regression task it missed by less than 1 on average.

Given that the raw data are categorical and not very detailed, as well as that time series forecasting is a difficult problem, the performance of the models is in my opinion quite decent and very interesting. It is interesting that it can find trends in the data and perform pretty good on the regression task. This coupled with the explanations gives an insight into why the model behaves as it does. Some features that are chosen as influential do not seem reasonable, but some could possibly have an impact in the real world.

Two features that are very prominent in both local and global explanations for both of the best performing models are, *Gender_2* and *WorkRelatedInjuryStaffIntInv_0*. Both of them occur multiple times in the global explanations at different number of time steps back. Both features seem to influence the predictions both higher and lower at different time steps back, which in essence means that the features are not strictly connected to predicting a high or low value, nor a high or low classification probability. What seems to a deciding factor is, which time step the value of these features was high or low. It makes sense that the *when*, very much matters.

In conclusion, to answer the research question, the different models actually successfully predict the future outcome the majority of the time. The models were successfully made interpretable but with limited domain knowledge it is difficult to evaluate the provided explanations.

7 Future Work

This section covers possible future work. It presents both work that can be done in order to improve the performance, and the interpretability of the models.

Causal interpretability The models in this project are interpretable, meaning that there are explanations of why the predicted outcomes of the models are of a certain value. As described in the Section 6, these explanations do not imply causality but association. The next step for the interpretability in this project would be to further investigate causal interpretability. There are three requirements formulated by Zhao and Hastie in the paper *Causal interpretations of black-box models*[37] to achieve causal interpretability:

1. A good predictive model which closely approximates the real relationship.
2. Domain knowledge to ensure the causal structure makes sense.
3. Partial Dependence Plots to show more clearly the dependence between certain features and output values.

The first and third of these requirements are partially met. The groundwork is laid, but due to time constraints, this project does not investigate this further.

Alternatives models There are alternatives to *Long Short-Term Memory(LSTM)*, one being *Gated Recurrent Unit(GRU)* and another called *Transformer networks*[13]. Switching from LSTM to a GRU or Transformer network could improve the performance on both the regression and classification task. The difference between LSTM and GRU is discussed in Section 3.3.2. There are alternatives to XGBoost such as CatBoost[23] and Light GBM[14]. These alternatives for GBDT can be implemented in trying to achieve even better performance.

Further data engineering A machine learning model is only as good as the quality of the data. If one could augment the data at hand, it could also improve the performance. Doing further feature engineering, perhaps by using the framework *FeatureTools* which enables the user to run automated feature engineering, could improve the results. It could be that by doing additional feature engineering one could find more strong links between input and output, which could improve the performance even further.

References

[1] *Kaggle: Your Machine Learning and Data Science Community*. Jan 2021. – URL https://www.kaggle.com. – [Online; accessed 20. Jan. 2021]

[2] AHMAD, Muhammad A. ; ECKERT, Carly ; TEREDESAI, Ankur: Interpretable machine learning in healthcare. In: *Proceedings of the 2018 ACM international conference on bioinformatics, computational biology, and health informatics*, 2018, S. 559–560

[3] AHMED, Nesreen K. ; ATIYA, Amir F. ; GAYAR, Neamat E. ; EL-SHISHINY, Hisham: An empirical comparison of machine learning models for time series forecasting. In: *Econometric Reviews* 29 (2010), Nr. 5-6, S. 594–621

[4] BOJER, Casper S. ; MELDGAARD, Jens P.: Kaggle forecasting competitions: An overlooked learning opportunity. In: *International Journal of Forecasting* (2020)

[5] BONTEMPI, Gianluca ; TAIEB, Souhaib B. ; LE BORGNE, Yann-Aël: Machine learning strategies for time series forecasting. In: *European business intelligence summer school* Springer (Veranst.), 2012, S. 62–77

[6] *Vanilla Recurrent Neural Network*. Sep 2020. – URL https://calvinfeng.gitbook.io/machine-learning-notebook/supervised-learning/recurrent-neural-network/recurrent_neural_networks. – [Online; accessed 23. Sep. 2020]

[7] CERQUEIRA, Vitor ; TORGO, Luis ; SOARES, Carlos: Machine learning vs statistical methods for time series forecasting: Size matters. In: *arXiv preprint arXiv:1909.13316* (2019)

[8] CHATFIELD, Chris: *Time-series forecasting*. CRC press, 2000

[9] CHEN, Tianqi ; GUESTRIN, Carlos: Xgboost: A scalable tree boosting system. In: *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*, 2016, S. 785–794

[10] *File:Gated Recurrent Unit.svg - Wikimedia Commons*. Sep 2020. – URL https://commons.wikimedia.org/wiki/File:Gated_Recurrent_Unit.svg. – [Online; accessed 8. Oct. 2020]

[11] FAN, Hongxiang ; JIANG, Mingliang ; XU, Ligang ; ZHU, Hua ; CHENG, Junxiang ; JIANG, Jiahu: Comparison of Long Short Term Memory Networks and the Hydrological Model in Runoff Simulation. In: *Water* 12 (2020), Nr. 1, S. 175

[12] GALLOTTI, Riccardo ; BARTHELEMY, Marc: Anatomy and efficiency of urban multimodal mobility. In: *Scientific reports* 4 (2014), Nr. 1, S. 1–9

[13] JADERBERG, Max ; SIMONYAN, Karen ; ZISSERMAN, Andrew u. a.: Spatial transformer networks. In: *Advances in neural information processing systems* 28 (2015), S. 2017–2025

[14] KE, Guolin ; MENG, Qi ; FINLEY, Thomas ; WANG, Taifeng ; CHEN, Wei ; MA, Weidong ; YE, Qiwei ; LIU, Tie-Yan: Lightgbm: A highly efficient gradient boosting decision tree. In: *Advances in neural information processing systems*, 2017, S. 3146–3154

[15] KLUYVER, Thomas ; RAGAN-KELLEY, Benjamin ; PÉREZ, Fernando ; GRANGER, Brian E. ; BUSSONNIER, Matthias ; FREDERIC, Jonathan ; KELLEY, Kyle ; HAMRICK, Jessica B. ; GROUT, Jason ; CORLAY, Sylvain u. a.: Jupyter Notebooks-a publishing format for reproducible computational workflows. In: *ELPUB*, 2016, S. 87–90

[16] LUNDBERG, Scott M. ; LEE, Su-In: A unified approach to interpreting model predictions. In: *Advances in neural information processing systems*, 2017, S. 4765–4774

[17] MCKINNEY, Wes u. a.: pandas: a foundational Python library for data analysis and statistics. In: *Python for High Performance and Scientific Computing* 14 (2011), Nr. 9

[18] MENG, Yuan ; YANG, Nianhua ; QIAN, Zhilin ; ZHANG, Gaoyu: What Makes an Online Review More Helpful: An Interpretation Framework Using XGBoost and SHAP Values. In: *Journal of Theoretical and Applied Electronic Commerce Research* 16 (2021), Nr. 3, S. 466–490

[19] MOLNAR, Christoph: *Interpretable Machine Learning*. 2019. – https://christophm.github.io/interpretable-ml-book/

[20] *A Practical Guide To Hyperparameter Optimization.* https://nanonets.com/blog/hyperparameter-optimization/. – (Accessed on 11/23/2020)

[21] OUSSOUS, Ahmed ; BENJELLOUN, Fatima-Zahra ; LAHCEN, Ayoub A. ; BELFKIH, Samir: Big Data technologies: A survey. In: *Journal of King Saud University-Computer and Information Sciences* 30 (2018), Nr. 4, S. 431–448

[22] PEDREGOSA, Fabian ; VAROQUAUX, Gaël ; GRAMFORT, Alexandre ; MICHEL, Vincent ; THIRION, Bertrand ; GRISEL, Olivier ; BLONDEL, Mathieu ; PRETTENHOFER, Peter ; WEISS, Ron ; DUBOURG, Vincent u. a.: Scikit-learn: Machine learning in Python. In: *the Journal of machine Learning research* 12 (2011), S. 2825–2830

[23] PROKHORENKOVA, Liudmila ; GUSEV, Gleb ; VOROBEV, Aleksandr ; DOROGUSH, Anna V. ; GULIN, Andrey: CatBoost: unbiased boosting with categorical features. In: *Advances in neural information processing systems*, 2018, S. 6638–6648

[24] RIBEIRO, Marco T. ; SINGH, Sameer ; GUESTRIN, Carlos: " Why should I trust you?" Explaining the predictions of any classifier. In: *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*, 2016, S. 1135–1144

[25] SACHDEVA, Aashay: Experimentation with Variational Dropout -Do Subnetworks exist inside a Neural Network? In: *Medium* (2019), Jun. – URL https://aashay96.medium.com/experimentation-with-variational-dropout-do-subnetworks-exist-inside-a-neural-network-e482cbbea7dd

[26] SAMEEN, Maher I. ; PRADHAN, Biswajeet: Severity prediction of traffic accidents with recurrent neural networks. In: *Applied Sciences* 7 (2017), Nr. 6, S. 476

[27] SCHMIDHUBER, Jürgen ; HOCHREITER, Sepp: Long short-term memory. In: *Neural Comput* 9 (1997), Nr. 8, S. 1735–1780

[28] SEBASTIAN RASCHKA, Vahid M.: *Python Machine Learning*. Packt, 2019

[29] SHAPLEY, Lloyd S.: A value for n-person games. In: *Contributions to the Theory of Games* 2 (1953), Nr. 28, S. 307–317

[30] SOLUTION, West C.: *Isap Revision – Kontroll på säkerhet*. https://isap.se/. – (Accessed on 03/10/2021)

[31] STEC, Alexander ; KLABJAN, Diego: Forecasting crime with deep learning. In: *arXiv preprint arXiv:1806.01486* (2018)

[32] *Block-distributed Gradient Boosted Trees*. Mar 2020. – URL http://tvas.me/articles/2019/08/26/Block-Distributed-Gradient-Boosted-Trees.html. – [Online; accessed 8. Oct. 2020]

[33] WERBOS, Paul J.: Backpropagation through time: what it does and how to do it. In: *Proceedings of the IEEE* 78 (1990), Nr. 10, S. 1550–1560

[34] *West Code Solutions AB*. Jan 2021. – URL https://www.westcode.se. – [Online; accessed 10. Jan. 2021]

[35] WIKIPEDIA CONTRIBUTORS: *Iris flower data set — Wikipedia, The Free Encyclopedia*. 2020. – URL https://en.wikipedia.org/w/index.php?title=Iris_flower_data_set&oldid=979784701. – [Online; accessed 23-September-2020]

[36] ZHANG, Yanru ; HAGHANI, Ali: A gradient boosting method to improve travel time prediction. In: *Transportation Research Part C: Emerging Technologies* 58 (2015), S. 308–324

[37] ZHAO, QINGYUAN ; HASTIE, TREVOR u. a.: CAUSAL INTERPRETATIONS OF BLACK-BOX MODELS. In: *Journal of business & economic statistics: a publication of the American Statistical Association: Duplicate, marked for deletion* 2019 (2019)