University College London

Department of Computer Science

A thesis submitted in partial fulfilment of the requirements for the degree of Master of Science in Computational Finance, University College London

Can I trust you?: Explainability of a Long-Short-Term-Memory model for Corporate Bonds Spread predictions

Author
Erna Ceka

Academic Supervisor
Dr. Guido Germano
Department of Computer Science
University College London

Industrial Supervisor
Mr. Mayank Agrawal
Chief Technology Officer
Intellibonds Ltd

September 11, 2021

Abstract

This study aims at providing interpretability to a long-short term memory (LSTM) model that is used to obtain predictions of corporate bonds' spread changes. Spreads are often used as an indication of the market's assessment of credit risk. Accurately predicting spread changes has important implications for portfolio managers and fixed-income practitioners, as it is one of the main criteria that guides their investment decisions. While LSTM models have high predictive accuracy, they inherently suffer from low explainability. A key concern is their reputation as "black box" models — i.e., it is hard for decision-makers to understand the input transformations through multiple layers of the network and obtain insights of why certain predictions are made.

The objective of this thesis is to identify and implement explainable artificial intelligence (XAI) methods that can shed light into the interpretability of the LSTM model by quantifying the magnitude and the directional impact of individual features on the predicted output. Enhancing model's interpretability not only helps the end-user build trust on the predicted output, but it is also an explicitly stated requirement across various regulatory frameworks such as General Data Protection Regulation (GDPR) set by the European Union, its UK equivalent (UK-GDPR) and Basel III regulation.

A combination of model-specific and model-agnostic XAI methods have been used in this thesis namely, DeepSHAP which is an approximation of Shapley values applied to deep learning models, knowledge distillation with SHAP's tree explainer (using random forest and extreme gradient boosting (XGBoost) as student models), LSTM with layer-wise propagation, and an interpretable multi-variable LSTM using mixture attention mechanisms.

The original contribution of this project is: i) applying various XAI methods to the LSTM model and ii) evaluating their performance based on the criteria of interpretability, computational efficiency, accuracy and fidelity. We conclude that knowledge distillation based on the XGBoost student model is the best method for our use case, because it provides computationally efficient and intuitive explanations with a reasonable level of accuracy and fidelity. The recommended method will be implemented in Intellibonds' spread forecasting platform.

Acknowledgments

First and foremost, I would like to thank my industry supervisor Mayank Agrawal, for providing me with the great opportunity to work on an innovative topic such as the explainability of LSTM models, and for his continuous guidance and valuable advice throughout the process. Along with Mayank, Matthew Marr from Intellibonds has been a continuous source of support and guidance. I would also like to thank my academic supervisor Dr. Guido Germano for his support and valuable advice in writing up this thesis.

I am deeply thankful to my classmate Jonathan Tang for his collaboration, valuable insights and support in conducting the analysis for this thesis. It has been a great pleasure working with him over the past three months.

I would also like to extend my thanks to the Computer Science Department at UCL and in particular to the academic staff working on the MSc Computational Finance programm.

Finally, I wish to thank my husband, my family and friends, for their unconditional support and encouragement throughout my studies.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Deep learning is an emerging approach that has been recently used in addressing prediction problems in time series data. Their inherently non-linear and non-parametric nature in conjunction with the useful property of being universal functional approximators, supports the application of deep learning models to financial time series forecasting [1]. Multilayer perceptron (MLP), which is composed of a series of fully connected layers, was one of the first models used with time series data. However, MLPs are memory-less, and the information only moves forward — from the input layer, through the hidden layers and to the output layer. There are no cycles or loops in the network, which makes MLPs unable to effectively model temporal dependencies [2].

Conversely, a specific class of deep-learning models called recurrent neural networks (RNNs), can be useful in modelling sequential information. Each neuron in the RNN network has an internal memory that preserves information from computations on the previous samples [3]. As you are reading this sentence, you are processing it word by word, while keeping memories of what came before; this gives you a dynamic representation of the meaning conveyed by this sentence. RNNs adopt the same principle. They process sequential data by iterating through the sequence elements and maintaining a state which holds information relative to what it has seen so far [4].

However, RNNs suffer from the vanishing and exploding gradient problem, which makes them hard to train and capture long-term dependencies. One solution to overcome this weakness is to use a more complex network architecture such as long-short-term memory (LSTM) models, which employ structures called "gates" to process information and model long-term dependencies more effectively [5].

In this study, a deep LSTM model is used to predict changes in corporate bonds' spreads [6]. Investing in corporate bonds involves credit risk primarily attributed to: (i) default risk and (ii) recovery risk. Credit spread is often used as an indication of the market's assessment of credit risk [7]. A widely used measure of credit risk is the z-spread which is the basis point spread that would need to be added to the implied spot yield curve such that the discounted cash flows of the bond are equal to its present value [8]. Accurately predicting spread changes is very important for portfolio managers and fixed-income practitioners, as it is one of the main criteria that guides their investment decisions. If the prediction suggests that the spread will widen(tighten) in the future, then investors could sell(buy) corporate bonds and buy(sell) government bonds [9].

While our LSTM model has high predictive accuracy it inherently suffers from low explainability [10]. A key concern for the wider application of LSTMs in quantitative finance, is their reputation as "black box" models — i.e., they are said to lack interpretability of how input data are transformed to model outputs. Therefore it is hard for decision-makers to understand the input transformations through multiple layers inside the network and obtain insights of why certain predictions are made [11]. Model interpretation is very important as it enables the user to trust the predicted output. It also helps fixed-income investors to identify the underlying drivers of predicted spreads and compare them with spread determinants implied by structural models and their domain knowledge. Finally, on the legal side, model interpretability is explicitly stated as a requirement across various regulatory frameworks such as General Data Protection Regulation (GDPR) set by the European Union, its UK equivalent (UK-GDPR) and Basel III regulation. These are the main motivations underpinning this study.

Therefore, the scope of this study is to investigate explainable artificial intelligence (XAI) methods that can enhance the interpretability of the LSTM model in use. There are two main research questions in this thesis.

1. Are there any model-agnostic and/or model-specific methods that can effectively interpret the predictions of the LSTM model?

2. Are the identified methods able to generate computationally efficient and intuitive interpretations with a reasonable level of accuracy?

Four explanation methods are considered appropriate to enhance the interpretability of the LSTM model: i) DeepSHAP which is an extension of Shapley values to deep learning models [12]; ii) knowledge distillation which is the process of transferring the knowledge learned by the LSTM model to a more interpretable student model, in order to achieve faster training and better interpretability, at the expense of accuracy. We have used ensemble methods such as random forest and XGBoost as student models. The results of ensemble methods are then interpreted using SHAP tree explainer [13]; iii) Interpretable multivariable LSTM (IMV-LSTM) which is a model-specific method that updates the architecture of the original LSTM model to obtain both predicted spreads and variable-wise importance scores. This is achieved by updating the hidden state matrix such that each row encapsulates information exclusively from a certain variable and by using a mixture attention mechanism to obtain explanations [14] and iv) Layer-wise relevance propagation LSTM model (LSTM-LRP) which is an alternative model-specific approach that explores the interface between the LRP method for explanations and the LSTM model for predictions [15].

The following hypothesis will be tested:

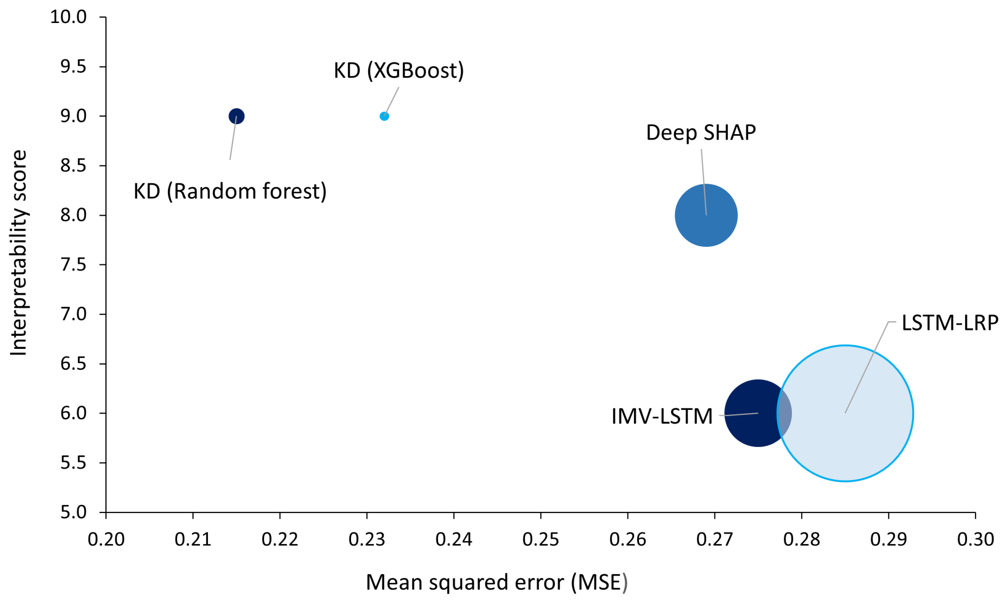
1. DeepSHAP is expected to generate the most reliable importance scores as it is considered a highly accurate approximation of the Shapley values and it is directly applied to the predictions of the original LSTM model. However this method is computationally inefficient as it is requires model re-training for different combinations of features.

2. Knowledge distillation is expected to be the fastest method as SHAP's tree explainer is well integrated with the random forest and XGBoost models. However, this comes at the expense of accuracy and fidelity because instead of trying to interpret the LSTM directly, the predicted labels of the student models are used.

3. Model-specific methods such as LSTM-LRP and IMV-LSTM are expected to perform better in terms of accuracy and fidelity because their network architecture has a closer resemblance to the original LSTM model.

We use the framework explained in Section 4.5 to test these hypothesis across the dimensions of model's interpretability, computational efficiency, accuracy and fidelity. We conclude that knowledge distillation based on the XGBoost student model is the best method for our use case, because it provides computationally efficient and intuitive explanations with a reasonable level of accuracy and fidelity. Figure 1.1, provides a summary of results for the different XAI methods under consideration.

Figure 1.1: Comparing XAI methods based on interpretability, accuracy and computational efficiency. The size of the bubbles represents the computational time required to train the model and to obtain explanations.



Note: A higher interpretability score suggests that the method under consideration is able to provide more intuitive explanations. Accuracy is measured in terms of the MSE score in order to be consistent with the loss function used in the training of the original LSTM model. A higher MSE score indicates lower accuracy. The size of the bubbles represents the computational time required to train the model and to obtain explanations.

This thesis is structured as follows. Chapter 2 outlines the theoretical details of LSTM networks and provides an overview of the relevant literature on spread prediction models and explainability methods. Chapter 3 provides details on the data and the main pre-processing steps, and describes the architecture of the original LSTM spread prediction model. Chapter 4 presents the design, rationale and implementation of four selected explainability methods. It also outlines the framework that is used to assess their performance. Chapter 5 presents the results of the analysis. Chapter 6 summarises the conclusions of the thesis, and presents ideas for further work.

Chapter 2

Literature review

2.1 Deep neural networks for time-series forecasting

Time series forecasting has been traditionally performed by econometric approaches such as autoregressive integrated moving average (ARIMA) models and its many variations [16]. An ARIMA model consists of three components: i) an autoregressive (AR) component; ii) an integrated component and iii) a moving average (MA) component. ARIMA models use a combination of past values and past forecasting errors and have the additional advantage of eliminating non-stationarity through differencing [17].

Even though ARIMA models are widely used in modelling economical and financial time series, they have some drawbacks [18].

- Since ARIMA models are essentially linear regression approaches, they fail to capture any non-linear dependencies in the data.
- Furthermore, they assume that there is a constant standard deviation in errors, which does not necessarily hold in practice.
- Finally, their accuracy significantly diminishes when handling long-term predictions.

Recently, deep-learning has been the emerging approach in addressing prediction problems in time series applications. Deep-learning models consist of multiple processing layers that enables them to learn complex dependencies in the data. Deep-learning models discover intricate structures in large data sets by using the backpropagation learning algorithm in order to update their internal parameters [1]. The architecture of deep learning models has originated from simple structures called artificial neural networks (ANNs) [19].

In its simplest form, an ANN is composed of three layers: the input layer (where the data is fed into the network), the hidden layer (where the information is processed) and the output layer (where the network decides what to do based on the data). ANNs that consist of multiple hidden layers are called “deep neural networks” [10].

ANNs have several advantages that enable them to produce more accurate results than traditional approaches to time series forecasting such as the Box-Jenkins or ARIMA method [20].

1. First, ANNs are non-parametric approaches. As such, there is no need to specify a particular form of the model or to make any assumptions about the statistical distribution of the data. They learn from training examples and capture subtle functional relationships among the data [2].

2. Second, ANNs are inherently non-linear, therefore they are well suited to model complex relationships in the data [20].

3. Finally, ANNs are universal functional approximators. They use parallel processing to approximate a large class of functions with a high degree of accuracy [21].

Until the late 2000s, neural networks were relatively shallow as there was not a reliable way to train very deep neural networks. The key issue was that when back-propagating the gradients, the feedback signal used to train neural networks would vanish/explode as the number of layers increased [19]. In the early stages of deep learning, loss gradients would either be too large or too small to flow backwards efficiently, and the network would take longer to converge [19]. This changed around 2009–2010 with the emergence of several algorithmic improvements that allowed for better gradient propagation rules [1]. These improvements consist of:

- Better activation functions for neural layers (i.e, the introduction of non-linear activation functions such as: sigmoid, hyperbolic tangent, ReLU etc). See Appendix 7.1.

- Better weight-initialization schemes such as Xavier initialization [22], and Kaiming [23].

- Better optimization schemes, such as RMSProp and Adam [24].

The most popular ANN model that is used for the purpose of time series forecasting is the multilayer perceptron (MLP), which is composed of a series of fully connected layers [20]. MLPs are memory-less, and they use the feed forward neural network architecture and a supervised learning technique called back-propagation algorithm to train the neural network [25]. The term feedforward implies that the information moves only forward from the input layer, through the hidden layers and to the output layer. There are no cycles or loops in the network. This makes MLP’s unable to model temporal dependencies. See Appendix 7.1 for further details on ANNs and MLPs.

Conversely, a specific class of deep-learning models called recurrent neural networks (RNNs), can effectively process sequential information. While MLPs pass information without cycles, RNNs have cycles that help them store information from previous inputs. Each neuron in the RNN network has an internal memory that preserves information from computations on the previous samples [3]. This special memory is called recurrent hidden states [1].

The ability to store information is particularly important when modelling sequential data. RNNs process sequential data by iterating through the sequence elements and maintaining a state which holds information relative to what it has seen so far. This is enabled by structures called hidden states which act as internal storage for storing the information presented earlier in the sequence [4]. See Appendix 7.2 for further details on the architecture of RNNs.

Though RNNs have proven successful on tasks such as speech recognition [26], and text generation [27], it can be difficult to train them to learn long-term dynamics. This limitation is associated with the the vanishing and exploding gradient problem, which results from backpropagating gradients through multiple layers of the network, each corresponding to a particular time step [5]. More details are provided in Appendix 7.2.

One solution is to use a more complex network architecture such as long-short-term memory networks (LSTMs), which can effectively capture long-term dependencies. An LSTM model is an RNN model with the capacity of remembering the information from earlier stages in the network. LSTMs address the vanishing gradient problem by using the memory cell to remember the temporal state of the neural network and the gates formed by multiplicative units to control the pattern of information flow [5]. See Section 2.1.1 for more information.

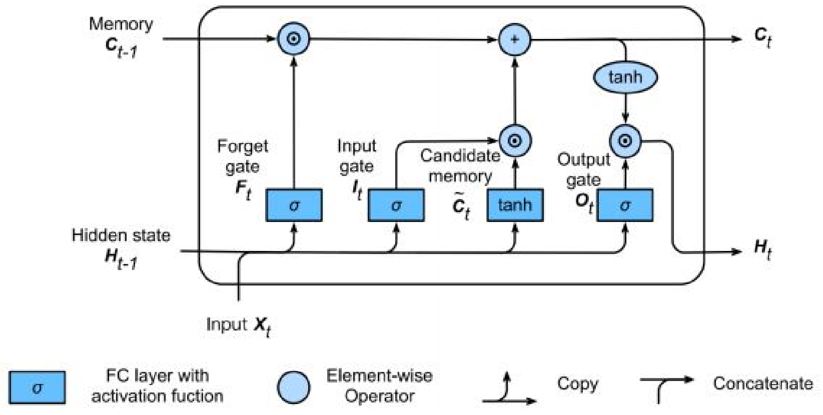
2.1.1 Long-short-term memory models (LSTMs)

The vanishing/exploding gradient problem, causes the parameters' learning process to stop, thus posing a major limitation in using RNNs for modelling long-term dependencies [5]. This problem motivated the introduction of LSTMs, which employ more complex structures called "gates", that can process information and model long-term dependencies more effectively [28].

The key in the LSTM structure is the cell state (memory cell), which conveys data from the past hidden states, while gathering new data from the present input vector. The efficient information processing in LSTM cells is done through special structures called "gates".

These gates enable the cells to optionally let data pass through [4]. The motivation for such a design is to have a mechanism which selectively decides when to “remember” and when to “forget” inputs in the hidden state [29]. The data that enter into the LSTM gates are the input at the current time step $\mathbf{x}_t$ and the hidden state of the previous time step $\mathbf{h}_{t-1}$, as illustrated in Figure 2.1. These vectors are processed by three fully connected layers with sigmoid activation functions, which compute the values of the input, forget and output gates [5].

Figure 2.1: The architecture of an LSTM unit and the information flow through the input, forget and output gates. [30].



We can describe the process of passing information in a LSTM unit using the mathematical notation proposed in Zhang et al. [29]. We assume that our LSTM network consists of u hidden units, and there are m input features for each example. Thus the input vector at time step t corresponding to one single example is denoted as $\mathbf{x}_t \in \mathbb{R}^m$, and the hidden state from the previous time step is denoted as $\mathbf{h}_{t-1} \in \mathbb{R}^u$. These vectors are introduced to the LSTM block, which employs regulated gates to update the hidden state $\mathbf{h}_t$ [31]. The gates at time step t are defined as follows: the forget gate is $\mathbf{f}_t \in \mathbb{R}^u$; the input gate is $\mathbf{i}_t \in \mathbb{R}^u$ and the output gate is $\mathbf{o}_t \in \mathbb{R}^u$ [29]

$$\mathbf{f}_t = \sigma(\mathbf{W}_{\mathrm{xf}}\mathbf{x}_t + \mathbf{W}_{\mathrm{hf}}\mathbf{h}_{t-1} + \mathbf{b}_{\mathrm{f}}) \tag{2.1}$$

$$\mathbf{i}_t = \sigma(\mathbf{W}_{\mathrm{xi}}\mathbf{x}_t + \mathbf{W}_{\mathrm{hi}}\mathbf{h}_{t-1} + \mathbf{b}_{\mathrm{i}}) \tag{2.2}$$

$$\mathbf{o}_t = \sigma(\mathbf{W}_{\mathrm{xo}}\mathbf{x}_t + \mathbf{W}_{\mathrm{ho}}\mathbf{h}_{t-1} + \mathbf{b}_{\mathrm{o}}), \tag{2.3}$$

where $\mathbf{W}_{\mathrm{xf}}, \mathbf{W}_{\mathrm{xi}}, \mathbf{W}_{\mathrm{xo}} \in \mathbb{R}^u \times \mathbb{R}^m$ and $\mathbf{W}_{\mathrm{hf}}, \mathbf{W}_{\mathrm{hi}}, \mathbf{W}_{\mathrm{ho}} \in \mathbb{R}^u \times \mathbb{R}^u$ are the weight parameters corresponding to the current input vector $\mathbf{x}_t$ and the previous hidden state $\mathbf{h}_{t-1}$, and $\mathbf{b}_{\mathrm{f}}, \mathbf{b}_{\mathrm{i}}, \mathbf{b}_{\mathrm{o}} \in \mathbb{R}^u$ are bias parameters. Time t flows over T discrete time steps, starting from

$t = 1$. The computations inside an LSTM unit can be described as follows:

- **Forget gate** uses a sigmoid function to decide what information that will be disregarded from the cell state [32].

- **Input gate** decides which new information is going to be stored in the cell state. The input gate consists of two layers: a sigmoid layer and a tanh layer. The sigmoid layer $\mathbf{i}_t$ decides which values need to be updated, whereas the tanh layer creates a vector of new candidate values $\tilde{\mathbf{c}}_\mathbf{t} \in \mathbb{R}^u$ that will be added into the LSTM memory cell [29]. This leads to the following definition for the candidate memory cell $\tilde{\mathbf{c}}_\mathbf{t}$

$$\tilde{\mathbf{c}}_\mathbf{t} = \tanh(\mathbf{W}_{\text{xc}}\mathbf{x}_t + \mathbf{W}_{\text{hc}}\mathbf{h}_{t-1} + \mathbf{b}_{\text{c}}), \tag{2.4}$$

  where $\mathbf{W}_{\text{xc}} \in \mathbb{R}^u \times \mathbb{R}^m$ and $\mathbf{W}_{\text{hc}} \in \mathbb{R}^u \times \mathbb{R}^u$ are weight parameters of the memory cell and $\mathbf{b}_{\text{c}} \in \mathbb{R}^u$ is a bias parameter.

- **Update of the memory cell state** which consists of a combination of the forget gate and the input gate, in which the input gate $\mathbf{i}_t$ decides how much new information is taken into account via $\tilde{\mathbf{c}}_\mathbf{t}$, and the forget gate $\mathbf{f}_t$ addresses how much of the old memory cell content $\mathbf{c}_{\mathbf{t-1}}$ is retained. The update of the memory cell state is defined as

$$\mathbf{c}_t = \mathbf{f}_t \odot \mathbf{c}_{t-1} + \mathbf{i}_t \odot \tilde{\mathbf{c}}_\mathbf{t}, \tag{2.5}$$

  where $\odot$ is the symbol for the Hadamard (elementwise) product operator.

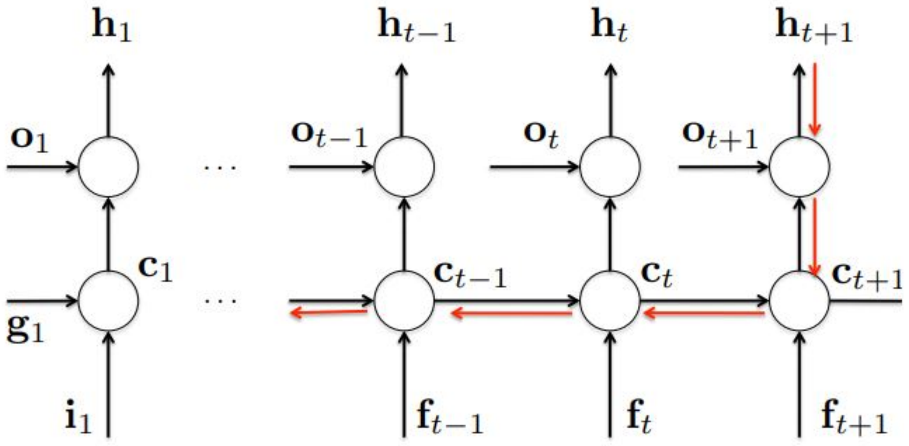
- **Output gate** is used in the computation of the update of the hidden state $\mathbf{h}_t \in \mathbb{R}^u$ and it relies on a sigmoid layer which decides what part of the LSTM memory contributes to the output. When the value of the output gate approximates 1 the entire information is effectively passed through to the predictor, whereas when the value is close to 0 the information is retained only within the memory cell and no further processing is performed. To compute the update of the hidden state, we perform a non-linear tanh function of the memory cell in order to map the values between 1 and 1,

$$\mathbf{h}_t = \mathbf{o}_t \odot \tanh(\mathbf{c}_t). \tag{2.6}$$

Similar to RNNs, the optimal values of the weights and bias parameters, are obtained using back-propagation through time (BPTT) which relies on the application of the chain rule with time dependency [5]. See Appendix 7.2 for further details on BPTT. Figure 2.2 helps better understand this process by unfolding in time the information flow in a LSTM unit.

The advantage of the LSTMs' architecture is that it provides a solution to RNNs' vanishing/exploding gradient. This is attributed to: (i) the additive update function of the

Figure 2.2: Backpropagation through time in LSTMs [33].



cell state and ii) the gating functions which allow the network to decide how much the gradient vanishes, and can take on different values at each time step. See Greff et al. [34] for further information on the topic.

In modelling complex time-series dependencies it is common to use more than one LSTM unit [35]. LSTM units merge to form an LSTM layer [34]. A network consisting of the input layer, one or more LSTM layers and the output layer is called an LSTM network. The learning process in an LSTM network consists of the following steps [35]:

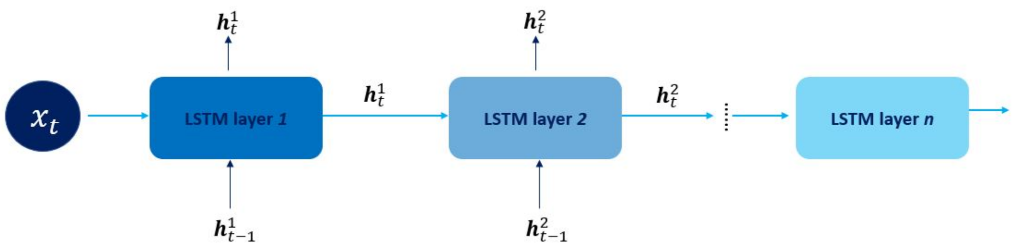
- Compute the output of LSTM units in each layer using Equations (2.1–2.6) (this process is called forward learning).
- Compute the error between the predicted output and the target output.
- The error is then reversely propagated to input gate, cell, and forget gate.
- Based on the backpropagated error, the weight of each gate is updated using an optimization algorithm.

The above steps are repeated for a given number of iterations and the optimal values of weights and biases are obtained through BPTT [34].

It is demonstrated that a shallow LSTM architecture consisting of just one LSTM layer, can not efficiently represent the complex dependencies in time series data [36]. Hence, increasing the depth of the LSTM network is an effective way to improve the modelling of highly nonlinear and long sequences of data [36]. Deep LSTM (DLSTM) is an extension of the vanilla LSTM network, which includes multiple LSTM layers such that each layer contains multiple LSTM units [6].

A simplified representation of a DLSTM network is presented in Figure 2.3.

Figure 2.3: Architecture of a deep LSTM network [31].



The processing of information in a DLSTM network consists of the following steps:

- The input at time t, $\mathbf{x}_t$ is introduced to the first LSTM layer along with the previous hidden state $\mathbf{h}_{t-1}^1$. The superscript 1 represents the first LSTM block. The hidden state at time t, $\mathbf{h}_t^1$ which is computed in line with Equations (2.1-2.6), goes upward to the next time step $t+1$ and also goes forward to the second LSTM layer [31].
- The second LSTM layer uses the hidden state $\mathbf{h}_t^1$ and the previous hidden state $\mathbf{h}_{t-1}^2$ to compute $\mathbf{h}_t^2$, which is then fed into the next time step and the third LSTM block and so on, until the last LSTM block is compiled in the stack [31].

In case of large or complex data, it is demonstrated that such deep networks are better at modelling long-term and non-linear dependencies, compared to a shallow architecture consisting of simply one LSTM layer [31]. Another benefit is that such architecture allows the hidden state at each level to operate at a different timescale. These benefits are important in particular when handling large multivariate time series datasets [37]. For these reasons, the model that is used to obtain spread predictions in this thesis is a DLSTM model as explained in Section 3.2.

2.2 Overview of spread prediction models

This chapter introduces the evolution of spread prediction models in the bonds market. Beforehand, it is important for the reader to have an understanding of the fundamental blocks of bond pricing. Refer to Appendix 7.3 for a summary of key concepts such as: yield-to-maturity, bond's price and z-spreads.

Investing in corporate bonds involves credit risk which is associated with two components: (i) default risk and (ii) recovery risk. Default risk occurs when the counter-party in a financial contract (e.g. bond issuer) is not able too repay the contractual coupon and principal [7]. The recovery risk captures the uncertainty about the proportion of the loss that will be recovered in case of default [9]. Credit spread is often used as an indicator of the market's assessment of credit risk. There are two widely used measures for credit risk: (i) bond yield spreads, and (ii) credit default swaps (CDS) spreads [9].

In line with the discussion in Appendix 7.3, z-spreads are considered a useful tool to measure credit risk and are often used as a basis for making investment decisions. Fixed income investors can profit from abnormal trading by predicting z-spreads effectively [8]. For instance, if investors forecast that the spread will widen (tighten) in the future (i.e. this is translated in a positive (negative) change in z-spreads), they could sell (buy) corporate bonds and buy (sell) government bonds.

Considering its importance for decision-making in the fixed-income markets, change in z-spreads will be the target variable that we aim to predict using the LSTM model described in Section 3.2. Early structural credit risk models were initially used to obtain predictions of credit spreads. See Appendix 7.4 for further details. Despite being useful at yielding insights into credit spread determinants such as: risk-free rate, asset volatility, liquidity, taxation, credit rating, maturity etc. these models do not address the issue of spreads' predictability which is our main concern [38].

Machine learning models that provide a non-parametric mapping between the dependent and independent variables, have prominently been better at forecasting. Because of the nonlinear payoffs of corporate bonds and the high correlation between many of the bond characteristics, machine learning is well suited for such challenging prediction problems [39].

Bianchi et al. [40], use machine-learning techniques to model bond return predictability. The authors find that ANNs significantly outperform other forecasting methods in terms of the out-of-sample R^2. They notice that the success of ANNs lies in their ability to effectively capture the non-linear relationship between bonds' returns and the predictors.

Gu, Kelly, and Xiu [39], compare and evaluate the out-of-sample performance of several machine learning models in predicting the cross-sectional bond returns.[1] The authors show that the traditional unconstrained OLS fails to deliver statistically significant out-of-sample forecasting performance for future corporate bond returns. Whereas, machine learning models such as MLPs and LSTMs, substantially improve the predictive performance for both investment-grade and non-investment-grade bonds.

[1]The machine learning methods include the dimension reduction models (PCA and PLS), penalized methods (lasso, ridge, and elastic net), regression trees (random forests), and neural networks including the feed forward neural networks (FFN). In addition to these methods, the authors use the LSTM model to capture a long memory effect.

The authors also identify the most important bond characteristics in predicting the expected bond returns. They rank the most important features by measuring the reduction in R^2, as a result of reshuffling the values of each feature while holding the other features unchanged. Their results demonstrate that all machine learning models are generally in close agreement regarding the most influential bond-level characteristics, which can be classified into four categories [39]:

- **Bond characteristics** related to interest rate risk such as duration and time-to-maturity.
- **Risk measures** such as downside risk approximated by total return volatility, and systematic risk related to bond market beta, default and term beta.
- **Bond-level illiquidity** measures such as average bid and ask price, volume of trading etc.
- **Bond return characteristics** related to bond momentum and short-term reversal.

An LSTM model with 88.9% out-of-sample accuracy that forecasts the direction of the next-day change in the credit spread, was developed by Xiong et al. [41]. They compared the performance of several models such as: Ornstein-Uhlenbeck process, ensemble methods such as random forest and bayesian additive regression tree (BART), and LSTM model. The authors find that, BART works well in predicting the magnitude of the credit spread, but fails to predict the direction accurately. On the other hand, LSTM yields the best prediction of the next-day credit spread change with a 3-day look-back period. This is associated with the fact that the credit spread follows different patterns among different time regimes and LSTMs are able to capture these temporal dynamics by gradually forgetting the inputs from older data in order to put more weight on current features [41].

While empirical studies provide evidence on the outperformance of deep learning models in predicting credit spreads, these models still remain inherently difficult to interpret. This is a major limitation to their practical use.

2.3 Overview of explainability methods

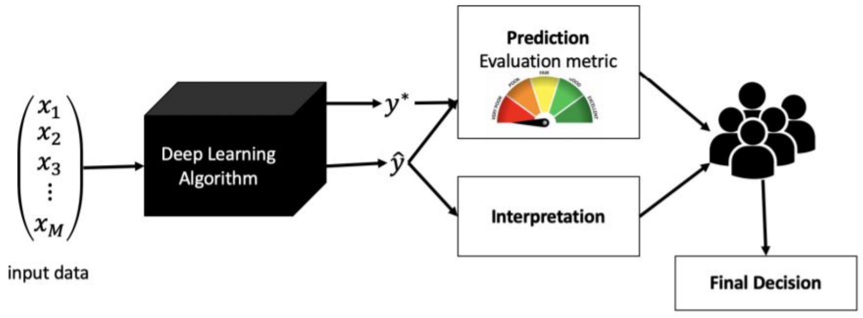
2.3.1 The importance of interpretability

As explained in Section 2.2, deep neural networks have gained popularity in the area of spread predictions due to their ability to substantially outperform conventional theoretical models, in terms of predictive accuracy. The success and application of deep neural networks is attributed to the fact that multiple layers of non-linear functions allow the network to model complex dependencies in the data [42]. However, this comes at the cost of explainability. A key concern for the wider application of deep neural networks is their reputation as "black box" model. It is hard for decision-makers to understand the input transformations through multiple layers inside the network and obtain insights of why certain predictions are made [10].

As stated by Molnar [43], there is no specific mathematical definition of model's interpretability. One definition was proposed by Miller [44]: "Interpretability is the degree to which a human can understand the cause of a decision". Another one was by Kim et al. [45]: "Interpretability is the degree to which a human can consistently predict the model's result". If the users do not trust a model, they will not use it. This trust is highly impacted by how much the human understands the model's behaviour and the underlying drivers of its predictions, as opposed to seeing it as a black box.

As suggested by Lipton [11], a black-box model should have a performance metric that measures how well the estimator $\hat{y}$ approximates to the actual value y^*, and another metric for interpretability which measures the degree of how much the user understands the predictions of the model. Ultimately, the final decision of whether to trust or not the output of the model resides in the decision maker.

Figure 2.4: Example of an interpretable deep learning model which with measures both model performance and interpretability [11].



Interpretability enables decision makers to understand which input features affect the output the most and identify the problem. In the context of bonds' spread predictions, interpretability is important for the following reasons:

- In order to make an investment decision based on predicted spread changes, fixed-income investors need to trust the model that is used for forecasting.
- Model interpretation helps fixed-income investors to identify the underlying drivers

of predicted spreads and compare them with spread determinants implied by theoretical models and domain knowledge.

- Model interpretability empowers the user with the flexibility to consider additional features and to make informed decision in the feature engineering process, in order to increase the predictive accuracy of the final output.
- Finally, model interpretability is explicitly stated as a requirement by the GDPR regulation set by the European Union and other similar regulatory frameworks that aim to increase algorithmic transparency.

2.3.2 Explainability methods (XAI)

Feature-based explanations and in particular **importance weights** are the most popular approach for explainability, enabling the user to get an intuitive understanding of the model by identifying the contribution of each feature on model's predictions [46]. Importance weights assign to each feature x_i in an instance $\mathbf{x}$, a weight $\phi_i(f, \mathbf{x})$ which represents the contribution of feature x_i to the prediction $f(\mathbf{x})$, where f is the black-box model which is being used for predictions. The weights are real numbers and the sign indicates the directional impact of the feature on the predicted output [46]. This is an intuitive way for the decision-maker to understand the underlying drivers of model's predictions.

A useful property of importance weights explainers is that the sum of the importance weights of all features in an instance is equal to the prediction of the model on that instance minus the bias of the model.[2] This is called feature-additivity and it is an important property that yields user-friendly explanations, because the feature importance values and their corresponding signs provide information on the magnitude and the directional impact of each feature on the predicted output.

Formally, for a model f and an instance $\mathbf{x}$, the feature additivity property suggests that the attributed weights $\phi_i(f, \mathbf{x})$ should meet the criteria in Equation 2.7.

$$\sum_{i=1}^{M} \phi_i(f, \mathbf{x}) = f(\mathbf{x}) - f(\mathbf{b}), \tag{2.7}$$

where M represents the number of variables that are fed into the black-box model. A large number of explanatory methods rely on the feature-additivity property [47]-[48]. Lundberg and Lee [12] have unified the feature-additive methods by showing that the only set of feature-additive importance weights that verify three properties: local accuracy, miss-

[2]The bias of a model is the model prediction on an input that brings no information, usually referred to as the reference or baseline input. For example, the zero-vector is a common baseline in predictive tasks etc.

ingness, and consistency, are given by the Shapley values from cooperative game theory. More on this in Section 4.1.

Before we introduce the specific interpretability models, we will first categorise them based on two important dimensions: i) the classes of models to which the method is applicable (i.e., model-specific vs. model-agnostic) and ii) the scope of explanations at which the method covers (i.e., local vs. global).

Model specific vs. model agnostic methods

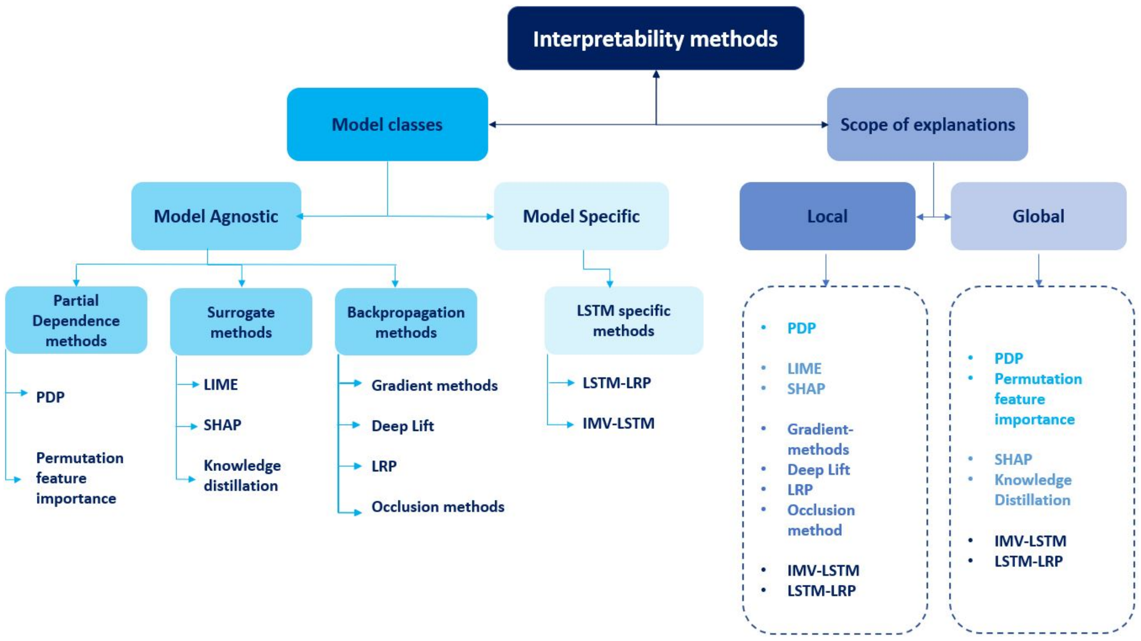
Model-specificity is directly related to the class of models that the interpretation method can be applied. Model-agnostic methods are applicable to any black-box model in general. On the other hand, model specific methods are applied only to certain classes. For instance, feature importance for ensemble methods can only be carried out to models with the corresponding structure [43]. The main advantage of model agnostic methods is that they allow for the same method to be applied across different target models. Model-specific methods on the other hand, have the advantage of generating more accurate interpretations for the specific model classes [43].

Local vs. global explanations

Another categorisation of XAI methods is based on the scope of the explanations. Methods that provide explanations on model behaviour at a specific data point are called local explainability methods. On the other hand, methods that provide aggregated information on model's interpretability are called global explainability methods [43].

An overview of interpretability methods based on the categorisation above is given in Figure 2.5.

Figure 2.5: Overview of interpretability methods based on model class and scope of explanations.



2.3.2.1 Model agnostic methods

The literature for model agnostic methods can be divided into two main research streams: i) partial dependence methods and ii) surrogate models.

Partial dependence methods

Partial dependence methods show the marginal effects of changing input features on the predicted outcome of a machine learning model. By marginalizing the model output over the distribution of the features in set C, we get a function that shows the relationship between the features in set S that we are interested in, and the predicted outcome [43].

The main disadvantage of PDP methods is the assumption of feature independence (i.e. they assume that the features for which the partial dependence is computed are not correlated with the other features in the model). Furthermore, these methods are computationally inefficient and it is difficult to visualise contributions of many features in the same graph. Therefore, partial dependence methods are not recommended for the interpretability of the LSTM model [43].

Another widely used method is permutation feature importance, which measures the importance of a feature by calculating the increase in model's prediction error after permuting the values of a given feature, while keeping the other features unchanged. The permutation feature importance is a **global explanatory method** and it was introduced by Breiman [43]. It has similar disadvantages to the partial dependence methods described above.

Surrogate models

The core idea underpinning surrogate models is that they use basic interpretable machine learning algorithms, such as linear regression and decision trees to approximate the predictions of complex black-box models. Distance functions are often used to estimate how close the predictions of the surrogate model approximate the black-box model.

Local interpretable model-agnostic explanations (LIME) is a surrogate model which enables local approximations around an individual prediction using a simple interpretable model such as a decision tree or a linear model [47]. LIME generates perturbed samples around the instance of interest $\mathbf{x}$, for which we want to have an explanation of model's prediction. On this new data, LIME then trains an interpretable model, which is weighted by the proximity of the sampled instances to the instance of interest. The learned model should be a good approximation of the machine learning model predictions locally if the model is not very complex [47].

Although LIME is a powerful and straightforward method, it has several disadvantages. First, it does not provide meaningful results for highly non-linear functions. Second, the poor choices of parameters could lead to missing out on important features. Third, the random perturbation and the feature selection methods that LIME uses, can result in unstable importance scores. This is because for the same prediction, different interpretations can be generated due to random perturbation of samples, which can be problematic for deployment [47].

SHAP (Shapley additive explanations) is a game-theory inspired method that provides **local** and **global** interpretability by computing the contribution of each feature on model's prediction. Computing Shapley values in closed form requires exponential time in the number of features. Therefore, Lundberg and Lee [12] introduced model-agnostic (KernelSHAP) and model-dependant (DeepSHAP and TreeSHAP) methods to effectively approximate Shapley values. See Section 4.1 for more details.

Knowledge distillation is a **global** model-agnostic approach which performs post-hoc explanations on a trained black-box model. This method is based on the idea that an interpretable student model could achieve better performance while providing interpreatability, by learning directly from the predicted labels of another model referred to as the teacher instead of learning from the original dataset. The student model is then used to obtain interpretability for the predictions of the teacher model. Commonly used student models are decision trees and ensemble methods [13]. See Section 4.2 for further details.

2.3.2.2 Model specific methods

We will group the model-specific methods in two categories: i) methods specific to deep neural networks (we will in particular focus on the backpropagation methods) and ii) methods specific to LSTM networks.

DNN-specific back-propagation methods

We have summarised below some of the main backpropagation methods. Some of these techniques rely on the model's local gradient information while other methods aim to redistribute the function's value on the input variables, typically by reverse propagation from the output layer to the input layer [49].

Gradient-based methods are **local** explainability methods, that use information on the gradient to quantify how much a change in each input dimension would result in a change of the prediction in a small neighborhood around the input. **The simple gradient method** obtains relevance scores for each feature by computing the partial derivatives of the prediction function. Each feature has a score estimated based on the gradient values with respect to the predicted output. **Gradient** $\times$ **input**, is a variation of the simple gradient method [50]. The feature attribution is computed by taking the (signed) partial derivatives of the output with respect to the input and multiplying them with the input itself.

Integrated gradients compute a contribution score for each feature by taking the integral of the gradients along a straight path from a baseline instance to the input instance. This method satisfies the desirable property of feature additivity, namely that the attributions sum up to the difference between the target output evaluated at $\mathbf{x}$ and the target output evaluated at the baseline. See Equation (2.7). However, it is computationally inefficient and it does not generate stable interpretations for highly non-linear models [51].

DeepLift is a **local** interpretation method applied to the class of deep-neural networks. Similarly to integrated gradients, DeepLIFT estimates the importance of each predictor by comparing the model prediction using an actual predictor value to a reference value. However, instead of actually integrating gradient values within the range of interest, DeepLIFT defines "multipliers" which are similar to partial derivatives, but over finite differences instead of infinitesimal ones [50]. DeepLIFT can be considered as a fast approximation of integrated gradients. It is not recommended to use DeepLIFT in models with multiplicative interactions such as LSTMs, because it loses the summation-to-delta property [52].

Layerwise relevance propagation (LRP) is a **local** interpretability method, where each neuron in the network gets assigned a relevance score, starting with the output neuron whose relevance is set to the prediction function's value, i.e. to $f(\mathbf{x})$. Based on the LRP propagation rule the relevance scores are redistributed iteratively, from higher-layer neurons to lower-layer neurons. For instance, for a linear layer of the form $z_j = \sum_i z_i w_{ij} + b_j$, and given the relevance of the output layer R_j, the input neurons' relevances R_i are computed through the following summation:

$$R_i = \sum_j \frac{z_i w_{ij}}{z_j + \epsilon \ \mathrm{sign}(z_j)} R_j, \tag{2.8}$$

where ϵ is a positive parameter that acts as a stabiliser. See Section 4.4 for further details. This rule is commonly referred to $\epsilon - LRP$. Other rules to compute LRP exists [53].

LSTM-specific methods

The existing literature on the interpretability of the LSTM models can be grouped in three main categories:

1. **Methods that perform interpretability analysis post-training** by perturbation on training data, such as surrogate models or gradient based methods described above.

2. **Methods that build attention mechanism on hidden states of the LSTM network** to characterise the importance at different time steps [54]. Current attention mechanisms are mainly applied to hidden states across time steps [55]. Since hidden states encode information from all input variables, the derived attention is biased when attempting to measure the importance of individual variables [14].

3. **Methods that update the internal structure of the vanilla LSTM model**, so that it can be used for both forecasting and obtaining variable importance scores. In this study we will focus on two LSTM specific models pertaining to this category.

   - **Interpretable multivariable LSTM model (IMV-LSTM)**, which enables the network to learn variable-wise hidden states, with the aim to capture different dynamics in multi-variable time series and distinguish the contribution of each variable to the prediction. IMV-LSTM updates the hidden state matrix of the LSTM network to encode information exclusively from individual variables, in order to ensure both forecasting and variable importance scores based on attention mechanisms [14]. See Section 4.3 for further details.

- Arras et al. [15], introduced the **LSTM-LRP** method which integrates the LRP method to the LSTM architecture. See Section 4.4 for further details.

Table 2.1 summarises the interpretability methods reviewed above and identifies the methods that are more suitable for LSTM networks.

Table 2.1: Summary of XAI methods and their application to LSTMs.

Method	**Local**	**Global**	**Application to LSTMs?**	**Why?**
PDP	✓	✓		i) assumption of feature independence; ii) computationally expensive.

Permutation importance		✓		i) computationally expensive;
				ii) assumes feature independence;
				iii) linked to the error of the model not model's predictions.

LIME	✓			i) not meaningful results for highly non-linear models; ii) poor choice of parameters affects importance scores; iii) generates unstable interpretations.
SHAP	✓	✓	☑	i) unifies feature-additive methods; ii) efficient model-specific methods (DeepSHAP and TreeSHAP).
Knowledge distillation		✓	☑	i) student models trained on predicted LSTM labels; ii) well-integrated with TreeSHAP to obtain explanations.
Gradient methods	✓			i) attribution scores strongly affected by noisy gradients; ii) scores are sensitive to small input variations; iii) can suffer from vanishing gradient problem; iv) integrated gradient is computationally expensive
DeepLift	✓			i) not recommended for LSTMs; ii) it losses summation to delta property.
LRP	✓			i) efficiently estimates relevance scores; ii) not suitable for models with multiplicative interactions (eg. LSTM).
IMV-LSTM	✓	✓	☑	i) uses variable-wise hidden states and attention mechanisms for forecasting and importance scores.
LSTM-LRP	✓	✓	☑	i) Adapts LRP technique to the LSTM architecture.

Chapter 3

Data and LSTM model for spread predictions

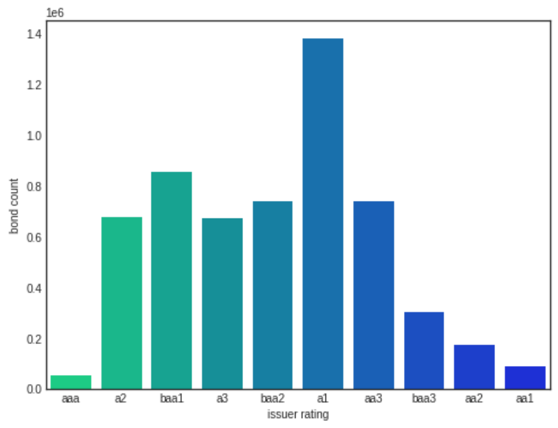
This chapter explains the data used in the LSTM spread prediction model. Furthermore, the architecture of the LSTM model is explained in detail in Section 3.2. The model selection and parameter optimisation is outside the scope of this thesis. The analysis presented in this report aims at providing interpretability for the predictions of the LSTM model. The data pre-processing and the set up of the LSTM model was completed prior to the industry placement. This section is structured as follows:

- First, we describe the data and the main pre-processing steps to obtain the final input features that are fed into the LSTM model. We also explore the distribution of the bonds in our sample and the histogram of the target output.
- Second, we explain the architecture of the LSTM model that was used to obtain predictions of z-spread changes.

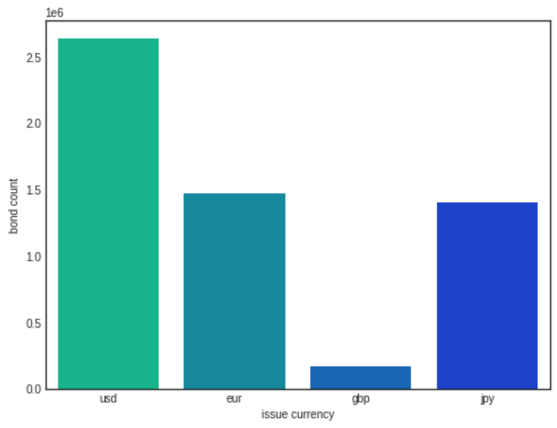
3.1 Data

The data used in this report consists of corporate bonds sourced from IHS Markit. The data was provided by Intellibonds. Originally there were 21 input features, for each observation. There are a total of 4.57 million observations, consisting of $36,388$ unique bonds, covering a time horizon from 31/08/2020 to 28/05/2021 for each bond. The unique identifier for each individual bond is the ISIN code. Figure 3.2, shows the distribution of unique bonds in terms of credit rating, issue currency and high-level sector categories. All the bonds in our sample are investment-grade only, the majority of the sample namely 46.3% are USD-denominated, and 54% of the bonds are issued by financial firms.

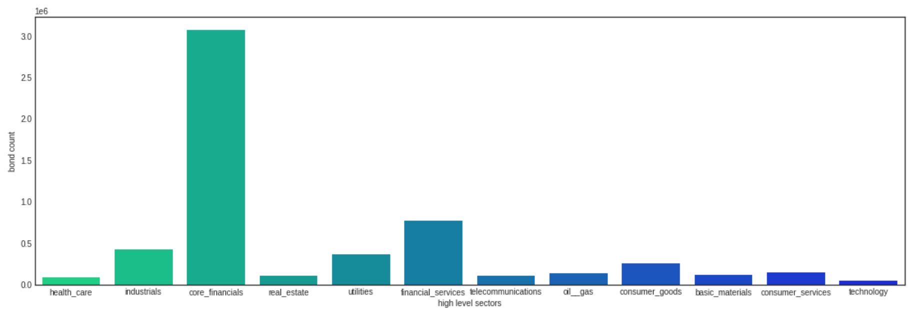
Figure 3.1: Distribution of unique bonds



(a) rating



(b) issue currency



(c) high-level sector

The original data consists of 21 features that provide bond-specific information related to fundamental pricing characteristics such as: yield-to-maturity (YTM), option adjusted spread (OAS), seniority, credit rating, duration, convexity etc. See Appendix 7.5 for a detailed description. The original features have been pre-processed into 58 features that are eventually used as input variables in the spread-prediction model. The main pre-processing steps consist of:

- **Removing duplicates and dropping bonds with missing observations**. Any bond with more than 80% of its data missing is removed from the sample. For bonds with less than 80% of the data missing, a rolling ordinary least squares (OLS) model is constructed to impute missing values.

- **Feature engineering using variable groupings**. Some variables are grouped together on the basis of other variables such as by maturity type, country, sector, credit rating etc. The grouped variables are further processed into weighted average, standard deviation, count, nominal rank, and percentile rank.

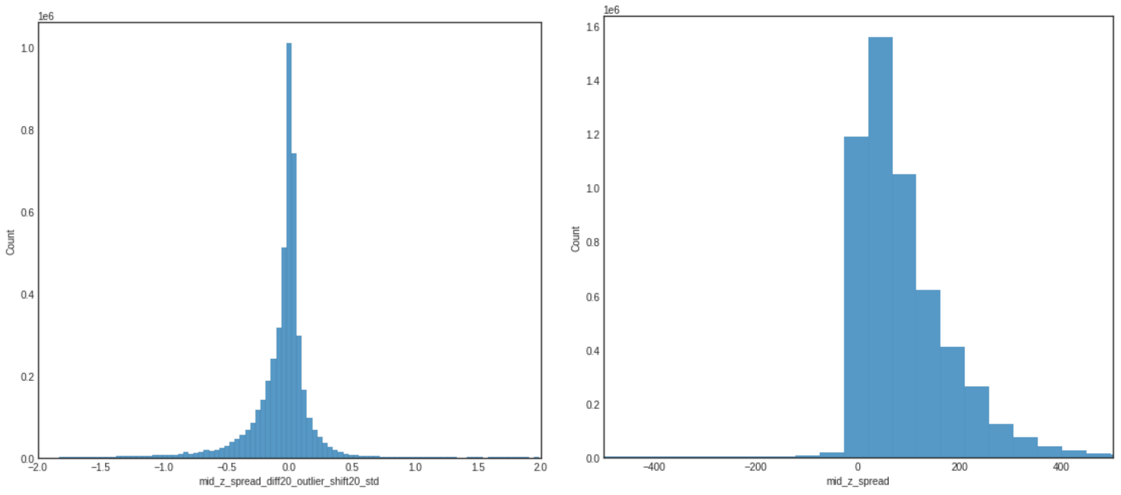
- **Standardisation**. All variables are standardised with the standard scaler,

$$z = \frac{x - \mu}{\sigma}, \tag{3.1}$$

  where x is the input variable, μ is the mean, and σ is the standard deviation of variable x.

The target output $\mathbf{y}$, also referred to as the vector of “spread changes”, is the **change in z-spreads** over the upcoming 20 working days. This should not be confused with the actual z-spread which represents the constant spread that will make the price of a security equal to the present value of its cash-flows, when added to the yield at each point on the spot-rate treasury curve. The histograms of the target output and the mid z-spreads are shown below.

Figure 3.2: Histogram of the target variable (i.e spread changes) and the mid z-spreads.



(a) target output   (b) mid z-spread

We have split our data into three subsets: train, test and validation set. In order to avoid model bias, we need to have a representative sample of unique bonds in each subset. Therefore, we have constructed these subsets, in a way that ensures that for each unique bond in our sample the first 70% of the observations belong to the train set, 20% of the observations to the test set and 10% of the observations to the validation set. We have split the data sequentially in order to make sure that the test set consists of the most recent 20% observations for each bond.

Given that the input fed into the LSTM spread prediction model has a temporal dimension, the pre-processed features were reshaped from a 2-dimensional matrix to a 3-dimensional matrix, ensuring that for each observation we have 58 corresponding features with a 10-day temporal sequence for each variable. We have further removed all the bonds that did not have enough data to construct the 10-day sequences for each feature. The final shape of the input data for the train, test and validation set is given in the table below.

Table 3.1: The final shape of the input data for the train, test and validation set.

Set	**Shape of the data**
X_train	(190324, 10, 58)
X_test	(101960, 10, 58)
X_val	(47582, 10, 58)

Note: The first dimension reflects the number of observations in each set. The second dimension represents the number of time-steps for each input variable. The third dimension reflects the number of input variables for each observation.

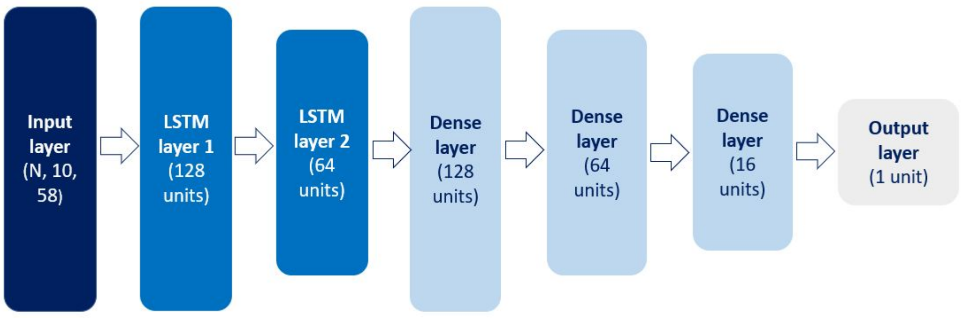
Since the pre-processed features are difficult to explain to the end user, the features are grouped into more representative categories to allow for better practical understanding. For instance, for spread changes grouped by maturity type, several variables were engineered in the pre-processing stage by computing metrics such as: the standard deviation, weighted average value, count etc. While these variables are useful to enhance the predictive accuracy of the LSTM model, they are not easy to understand by the end user. For this reason, we have grouped such variables together in a high-level category named "Maturity group-level spread changes", which should be interpreted as the level of spread changes for bonds that belong to the same maturity class. We end up with a total of 32 feature groupings.

The 58 pre-processed features are fed into the LSTM model to obtain predictions of spread changes over the next month. The aim of this thesis is to help the user interpret the results of the LSTM model. We employ the explainability methods described in Section 4, to quantify the contribution of individual variables on the predicted outcome for each unique bond in the sample. The contributions at the group level are then estimated by summing up the contributions of individual variables that belong to that group.

3.2 LSTM model for spread predictions

The current LSTM model uses the 58 pre-processed features with 10 time-steps for each feature as inputs. It gives the predicted value one-step ahead for the target-variable which is the changes in z-spread over 20 working days. The architecture of the network is a deep LSTM model, which consists of six hidden layers. The first two are standard LSTM layers followed by four dense fully-connected layers, with a rectified linear unit as the activation function. Finally, the output layer is a single neuron which generates the predicted change in the z-spread. Between the LSTM and the dense layer, there is a flattening layer since the outputs of the second LSTM layer is in 3d, and the dense layer requires the inputs to be in 2d. The model was constructed in Keras with a TensorFlow backend. The model architecture is shown in Figure 3.3.

Figure 3.3: Architecture of the LSTM model for spread predictions.



The loss function that is used to evaluate the performance of the LSTM model is the mean squared error (MSE), which is computed as

$$\text{MSE} = \frac{1}{N}||\mathbf{y} - \hat{\mathbf{y}}||_2^2, \tag{3.2}$$

where N is the number of observations, $\mathbf{y}$ is the actual output vector, $\hat{\mathbf{y}}$ is the predicted output vector and $||\mathbf{y} - \hat{\mathbf{y}}||_2^2$ is the squared L2-norm of the difference between the target output and the predicted output defined as $\sum_{i=1}^{N}(y_i - \hat{y}_i)^2$.

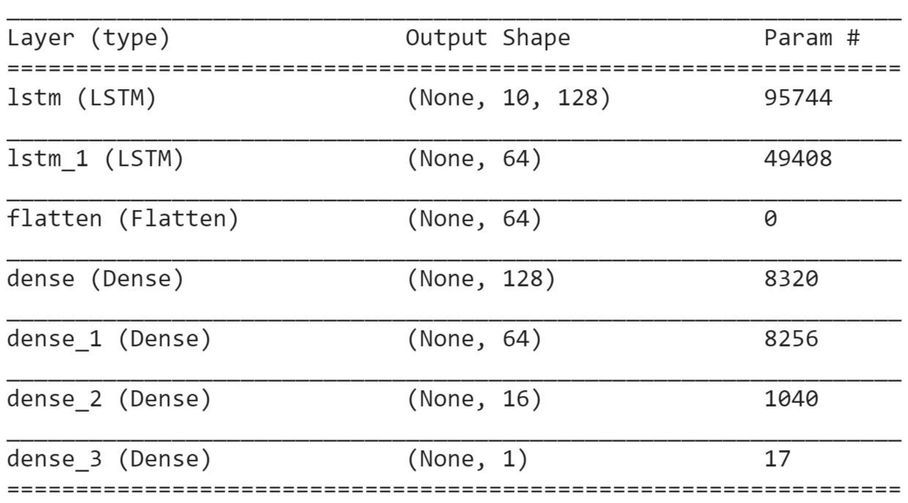
In order to control overfitting, a penalty term is included to reduce model complexity. This penalty term is based on the L2-regularisation, also known as “ridge regression”, and adds the L2-norm of the model parameters as a penalty term to the loss function

$$\mathbf{J}(\mathbf{y}, \mathbf{w}) = \frac{1}{N}||\mathbf{y} - \hat{\mathbf{y}}||_2^2 + \lambda||\mathbf{w}||_2. \tag{3.3}$$

The loss function is minimised using the Adam optimizer, which is an extension of the stochastic gradient descent that combines the best properties of the adaptive gradient algorithm (AdaGrad), and the root mean square propagation (RMSProp) [56]. Adam optimizer is commonly used for deep learning models because it effectively handles the sparse gradients on noisy problems, it works well with large data, it is computationally efficient and it requires little tuning of hyperparameters. Moreover, it is shown to have a faster and stable convergence compared to other optimisation algorithms [56].

The weights of the LSTM model are initialised via the TensorFlow and Keras' random seed, where the initial seed is set to 123. The learning rate in the training process is set to $1x10^{-6}$. The LSTM model is trained on 4.57 million examples with 1000 epochs and 1024 batches (i.e. batch training is used so that the weights are updated once every 1024 examples are seen). The early stopping is implemented so that if the loss on the validation set is the same for 100 epochs, then the training stops, and convergence has been reached. Figure 3.4 shows the total number of trainable parameters (i.e. weights and bias terms) for each layer in our deep LSTM model.

Figure 3.4: Trainable parameters in the LSTM network.



```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm (LSTM)                  (None, 10, 128)           95744
_________________________________________________________________
lstm_1 (LSTM)                (None, 64)                49408
_________________________________________________________________
flatten (Flatten)            (None, 64)                0
_________________________________________________________________
dense (Dense)                (None, 128)               8320
_________________________________________________________________
dense_1 (Dense)              (None, 64)                8256
_________________________________________________________________
dense_2 (Dense)              (None, 16)                1040
_________________________________________________________________
dense_3 (Dense)              (None, 1)                 17
=================================================================
```

It takes roughly 10 to 12 minutes on CPU time, to train the LSTM model on the investment grade bonds universe. For high-yield and other types of bonds, these models took around 30 seconds to train per epoch. No training of this model is required for this thesis, since the final model's weights are loaded and used to obtain LSTM predictions which will be then fed into the XAI methods described in Section 4.

Chapter 4

Methodology

The main objective of this thesis is to identify and implement explainability methods that would help the end-user understand and intuitively interpret spread predictions generated by the LSTM model described in Section 3.2. In this chapter, we describe the implementation of four XAI methods that are suitable for LSTM's interpretability. This chapter is structured as follows:

1. First, we describe the implementation of two model-agnostic methods, namely **SHAP deep explainer** and **knowledge distillation**.
    - **SHAP deep explainer** is a high-speed approximation algorithm for SHAP values in deep learning models. We use this method to obtain local and global explainability.
    - **Knowledge distillation** is another method of interest, which aims at explaining a complex model such as LSTM by distilling it into an interpretable student model. For the implementation of this method we have used two student models: random forest and XGBoost.
2. Second, we focus on two LSTM-specific methods namely, **IMV-LSTM** and **LSTM-LRP** that provide model interpretability by enhancing the architecture of the conventional LSTM model in order to obtain predictions and variable importance scores simultaneously.
3. Finally, we establish a framework to compare various XAI methods, that would help us identify the one that is more feasible for our LSTM spread prediction model. The chosen method will be implemented in Intellibond's platform.

4.1 SHAP deep explainer

SHAP is a widely used technique for model explainability and it is motivated by the computation of Shapley values from cooperative game theory. Shapley values provide a mathematical way to estimate the contribution of each feature on model's predictions [12]. For linear regression models the impact of each feature is the weight of the feature times its value. However this is not the case for more complex models. A more accurate way to estimate feature contributions in complex non-linear models is based on Shapley values, which quantify the average marginal contribution of a feature value across all possible combinations. A closed form solution to estimate Shapley values is

$$\phi_i(f, \mathbf{x}) = \sum_{S \subseteq M \setminus \{i\}} \frac{S!(M - S - 1)!}{M!} [f(\mathbf{x}_{\mathrm{S} \cup \{\mathrm{i}\}}) - f(\mathbf{x}_{\mathrm{S}})], \tag{4.1}$$

where M is the number of input features, $\mathbf{x}$ is the feature vector for an instance of the dataset, f is the black-box model, $\mathbf{x}_{\mathrm{S}}$ represents the values of the input features in the subset S and $\mathbf{x}_{\mathrm{S} \cup \{\mathrm{i}\}}$ represents the value of the input features in the subset $\mathrm{S} \cup \{\mathrm{i}\}$. The sum in Equation (4.1) enumerates over all subsets S of features in instance $\mathbf{x}$ that do not contain feature i. $S!$ represents the number of permutations of feature values that appear before feature value x_i. Similarly $(M - S - 1)!$ shows the number of permutations of feature values after the i-th feature. The difference term in the above equation is the marginal contribution of adding feature i to the subset S.

This method requires retraining of the model on all possible feature subsets $S \subseteq M$. This method assigns an importance score to each feature that represents the effect that the inclusion of that feature has on model's prediction. Since the effect of removing a feature depends on other features in the model, the difference $f(\mathbf{x}_{\mathrm{S} \cup \{\mathrm{i}\}}) - f(\mathbf{x}_{\mathrm{S}})$ is computed for all possible subsets. The Shapley values are then computed as weighted averages over all possible differences.

An important disadvantage which limits the practical use of the closed form solution of Shapley values is its computational complexity. The exact computation of the Shapley value is computationally expensive because there are 2^M possible combinations of the feature values for which the model needs to be retrained. Additionally, the absence of a feature is approximated by drawing random instances of that feature, which increases the variance of the estimation of Shapley values [43].

Lundberg and Lee [12] proposed KernelSHAP as an alternative kernel-based estimation for Shapley values that is inspired by local surrogate models. For explaining the output to any model, SHAP's kernel explainer incorporates the LIME method by constructing local surrogate models for every feature. The authors argue that KernelSHAP achieves similar accuracy while being computationally more efficient compared to the closed form solution of Shapley values.

One important novelty of KernelSHAP is that it represents Shapley values as an additive feature attribution method, thus providing a unified measure to other feature additive approaches such as: LIME, DeepLift, LRP etc [12].

While KernelSHAP improves the sample efficiency of Shapley values, it has several disadvantages. This method is still computationally slow for practical use and it ignores feature dependencies [43]. By restricting the attention to specific model classes, Lundberg and Lee [12] have developed faster model-specific approximation methods to compute Shapley values, such as DeepSHAP for deep learning models and TreeSHAP for ensemble methods.

4.1.1 DeepSHAP

DeepSHAP i.e, SHAP deep explainer, works in a similar way to KernelSHAP however it has the additional advantage of using the extra knowledge about the architecture of deep networks to improve its computational performance. It provides an efficient estimation approach for deep neural networks, and it works as a combination of the DeepLIFT algorithm, and the Shapley values.

DeepLift uses a "summation-to-delta" property that attributes to each input x_i a value $C_{\Delta x_i \Delta_o}$ that represents the effect of setting this input to a reference value as opposed to its original value, on the predicted output [50].

$$\sum_{i=1}^{n} C_{\Delta x_i \Delta o} = \Delta o \tag{4.2}$$

where $o = f(x)$ is the model output, $\Delta o = f(x) - f(r)$, $\Delta x_i = x_i - r_i$ and r is the reference input. DeepSHAP effectively approximates Shapley values by recursively passing DeepLIFT's multipliers, which are defined in terms of SHAP values, backwards through the network [12].

The DeepSHAP algorithm approximates Shapley values for M input features in $\mathcal{O}(M^2)$ network evaluations, compared to $\mathcal{O}(2^M)$ of the exact computation of Shapley values. While this is an improvement, it is still not ideal for a large dataset. Given that the complexity of DeepSHAP scales linearly with the number of background data samples, passing the entire training set will give very accurate expected values, but it will be computationally expensive.

As a result, in order to further reduce the computational cost of this method, SHAP deep explainer, performs a secondary approximation by reducing the number of samples for which Shapley values are calculated. Since the variance of the expectation is scaled by

$\sim \frac{1}{N}$ where N is the number of training examples, using a smaller subset as a training sample can still give vary accurate DeepSHAP values for the test set. In our implementation we use 500 examples of the train set as an approximation for the reduced sample. In order to implement DeepSHAP method to our LSTM model we have used the `DeepExplainer` package in python which is supported for TensorFlow and Keras.

4.1.2 Implementation of DeepSHAP to our data

As explained in Section 4.1.1, DeepSHAP is a computationally efficient method for deep neural networks that approximates the exact computation of Equation (4.1). It generates feature importance scores for each observation in our test sample (i.e., local explanations). The idea behind SHAP feature importance is intuitive: features with large absolute values are important. The feature importance scores are signed values, and the sign represents the directional impact of the feature on the predicted output. For a given example, the SHAP values produced from DeepExplainer should sum up to that example's predicted output. Global explanations are a summation over the absolute SHAP values for every feature, across all examples in the test set

$$I_j = \sum_{i=1}^{N} |\phi_j^{(i)}|. \tag{4.3}$$

A pseudocode explaining the implementation of DeepSHAP to our spread prediction model is provided in Algorithm 1.

Algorithm 1: DeepSHAP implementation for the LSTM spread prediction model

Input: train data $\{\mathbf{X}_{\text{train}}, \mathbf{y}_{\text{train}}\}$, test data $\{\mathbf{X}_{\text{test}}, \mathbf{y}_{\text{test}}\}$ and trained LSTM model f.

Returns: i)local explainability for each unique bond in the test data; ii) global explainability.

Algorithm:

1. Load the original trained LSTM model, training features $\mathbf{X}_{\text{train}}$, and test features $\mathbf{X}_{\text{test}}$.
2. Instantiate the shap.DeepExplainer object using the trained LSTM model and a subset of 500 observations from the train set.
3. Get DeepSHAP values for all the observations in the test set. These values provide importance scores for each features at each time-step over the 10-day sequence. Compute the variable-level importance scores by summing up the importance scores for each time-step. Use the variable-wise importance scores to obtain local explainability for each unique bond in our test sample.
4. Aggregate local explainability results as suggested by Equation (4.3) to get global explanation scores.
5. Local and global attributions at the group level are estimated by summing up the importance scores of individual features that fall under each group.

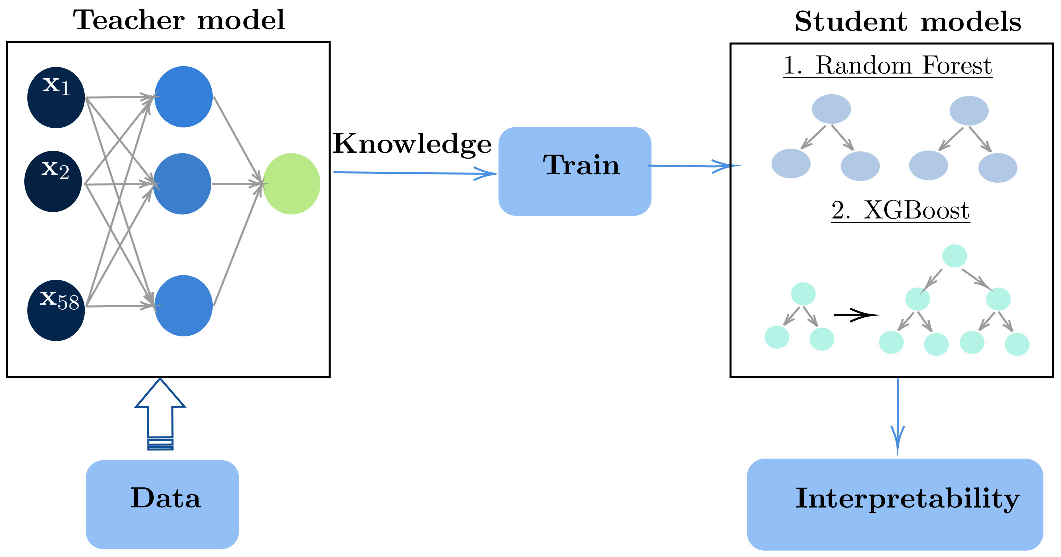
Deep SHAP provides fairly accurate feature importance scores for the predictions of the LSTM model as it is directly applied on the trained model and it achieves similar results as the closed form solution of the Shapley values while being computationally more efficient. However, the improvement in computational time is not as significant as to justify its practical application.

4.2 Knowledge distillation

Knowledge distillation refers to the process of transferring the knowledge learned by the teacher model (usually a highly complex non-linear model) to a more interpretable student model, in order to achieve faster convergence and better interpretability at the expense of accuracy [13]. This concept can be applied to our spread prediction model. Instead of trying to interpret the LSTM directly, we use the predicted labels of the trained LSTM model to train a student model that mimics the input-output relationship discovered by the teacher model. The student model is then used to make predictions on the test set. It may perform slightly worse than the teacher model but it will often be much faster and more importantly, it will increase model's interpretability.

Ensemble methods such as random forest and XGBoost consistently achieve better out-of-sample predictive accuracy compared to soft decision trees due to their ability to control over-fitting and reduce model variance [57]. Therefore we have used ensemble methods as opposed to soft decision trees as student models in the knowledge distillation process of the LSTM spread prediction model. This process is visualised in Figure 4.1.

Figure 4.1: Knowledge distillation of the LSTM model for spread predictions. Random forest and XGBoost models are trained on the predictions of the LSTM teacher model.



4.2.1 Random forest and XGBoost as student models

4.2.1.1 Random forest

As the first student model in the knowledge distillation of the LSTM teacher model, we have used a random forest regressor that fits a number of decision trees on different sub-samples of the dataset and averages them out to improve the predictive accuracy and control over-fitting [58]. Random forest uses the bootstrap aggregation or bagging technique which repeatedly selects a random sample with replacement from the training set B times, and fits a regression tree to each selected sample [59].

The main advantage of random forest compared to soft decision trees is that it selects a random subset of the features at each candidate split in the learning process. This helps de-correlate the trees in the forest and reduce model's variance [57]. In summary, the random forest algorithm works as follows:

Algorithm 2: Random forest regressor [57].

```
for $b = 1, 2, ..., B$:
    1. Sample with replacement $n$ training examples from $\mathbf{X}_{\text{train}}$ and $\mathbf{y}_{\text{train}}$. Denote the
       selected samples as $\mathbf{X}^b_{\text{train}}$, $\mathbf{y}^b_{\text{train}}$.
    2. Train a random-forest tree $T_b$ on the bootstrapped data, by recursively repeating
       the following steps, until the minimum node size $n_{\text{min}}$ is reached:
        • Select $m$ variables at random from the $M$ variables in the train set.
        • Pick the variable/split point that leads to the highest reduction of the loss
          function, among the $m$ subset of features.
        • Split the node into two daughter nodes.
Output the ensemble of trees $\{T_b\}_1^B$.
After training is completed, predictions for unseen examples $\mathbf{x}'$ in the test set, can
be made by averaging the predictions from all the individual regression trees on
$\mathbf{x}'$ i.e. $\hat{f} = \frac{1}{B}\sum_{b=1}^{B} f_b(\mathbf{x}')$.
```

The most important hyper-parameters that affect the accuracy and efficiency of the random forest model are [58]:

- `n_estimators`: the number of trees in the random forest;
- `max_features`: the maximum number of features to be considered when searching for the best split;
- `max_depth`: the maximum depth of the tree.
- `min_samples_split`: the minimum number of samples to split at each node.
- `min_samples_leaf`: the minimum number of samples required to be at a leaf node.

The computational complexity of training a random forest model is of the order $O(n^2MT)$, where n is the number of training samples, M is the number of input features and T is the number of trees [59].

4.2.1.2 Extreme Gradient Boosting Algorithm (XGBoost)

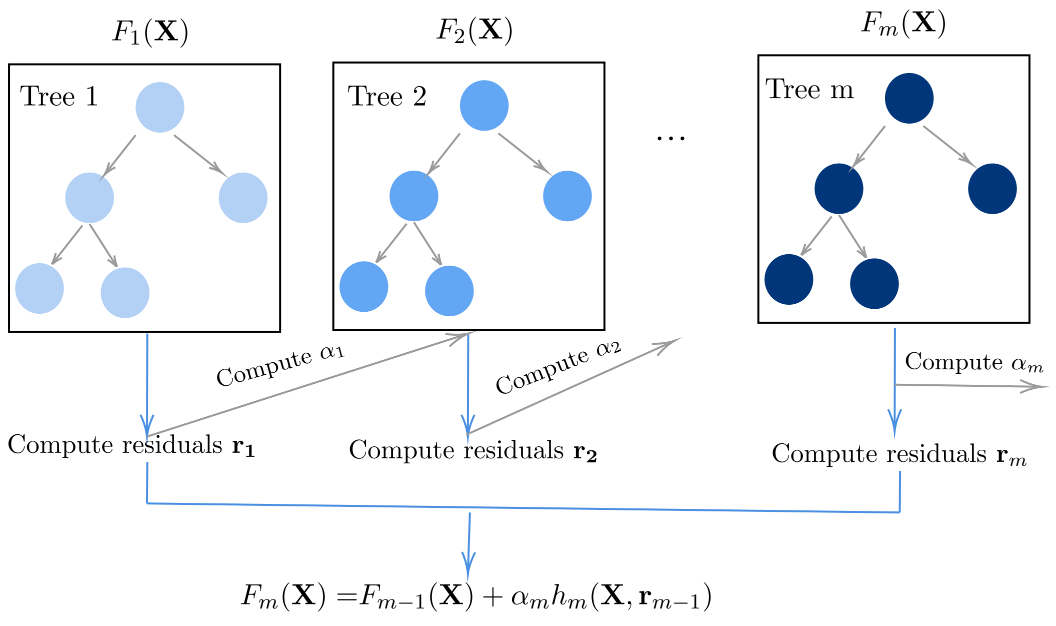
We have also used XGBoost as a student model, given its popularity for being computationally efficient. XGBoost was introduced by Chen and Guestrin [60], and it is based on the concept of gradient tree boosting, which aims to accurately predict the target variable by combining the estimates of a set of weaker learners. A weak learner refers to a learning algorithm that only predicts slightly better than randomly. Boosting consists of the sequential building of the trees, where each subsequent tree aims to reduce the errors of the previous tree. Gradient boosting refers to the application of the boosting process in conjunction with a gradient descent algorithm to minimise the loss function when adding new trees.

XGBoost provides an efficient way to implement gradient boosting, because it minimizes a regularized (L1 and L2) objective function that combines the loss function with a penalty term α that aims at reducing the complexity of the model. The training is then done iteratively, adding new trees that predict the residuals or errors of prior trees that are then combined with previous trees to make the final prediction [60]. This is formally defined in Equation (4.4)

$$F_i(\mathbf{X}) = F_{i-1}(\mathbf{X}) + \alpha_i h_i(\mathbf{X}, \mathbf{r}_{i-1}) \tag{4.4}$$

where $\mathbf{X}$ and $\mathbf{y}$ represent the input matrix and the target vector respectively, F is a function that minimises the loss or mean squared error between the target label $\mathbf{y}$ and the predicted label $\hat{\mathbf{y}}$, α and $\mathbf{r}_i$ refer to the regularisation parameter and the residuals computed for the i^{th} tree and h_i is a function that is trained to predict residuals $\mathbf{r}_i$. This process is illustrated in Figure 4.2.

Figure 4.2: Illustration of the XGBoost model based on the concept of gradient tree boosting where each subsequent tree aims to reduce the errors of the previous tree.


$F_1(\mathbf{X})$
$F_2(\mathbf{X})$
$F_m(\mathbf{X})$
Tree 1
Tree 2
...
Tree m
Compute α_1
Compute α_2
Compute α_m
Compute residuals $\mathbf{r_1}$
Compute residuals $\mathbf{r_2}$
Compute residuals $\mathbf{r}_m$
$F_m(\mathbf{X}) = F_{m-1}(\mathbf{X}) + \alpha_m h_m(\mathbf{X}, \mathbf{r}_{m-1})$

To compute α, the estimated residuals $\mathbf{r}_i$ are used, in conjunction with the optimisation problem

$$\underset{\alpha}{\text{argmin}} = \sum_{i=1}^{m} \mathcal{L}\big(\mathbf{y}_i, F_{i-1}(\mathbf{X})\big) + \alpha h_i(\mathbf{X}, \mathbf{r}_{i-1} \tag{4.5}$$

where $\mathcal{L}\big(\mathbf{y}, F(\mathbf{X})\big)$ is a differentiable loss function, and m is the total number of week learners used in the gradient tree boosting algorithm. The pseudo-code for the XGBoost algorithm is provided below:

Algorithm 3: XGBoost algorithm

```
- Initialise the model $F_0(\mathbf{X})$.
for $1 \leq i < M$ do
    - Compute the gradients.
    - Fit a week learner on the training set.
    - Update the model $F_i(\mathbf{X})$ as given in Equations 4.4 and 4.5.
end
```

XGBoost was developed to increase the speed and performance of the gradient tree boosting, while introducing regularization parameters to reduce overfitting [60]. XGBoost is therefore a preferred method for our analysis because it combines the benefits of the random forest model (i.e. reduced variance and over-fitting) while being at the same time computationally efficient. The main benefit of XGBoost from a computational viewpoint is its block structure for parallel learning. For faster computing, XGBoost can make use of multiple cores on the CPU. Also, cache is managed efficiently, and for large datasets as in the case of this thesis, XGBoost can deal with optimising disk space for data that cannot fit into memory directly.

From an algorithmic viewpoint, the algorithm can handle sparse data better than other tree methods. The computational complexity of training with XGBoost models is of the order $\mathcal{O}(Tdxn)$, where t is the number of trees, d is the height of the trees, x is the number of non-missing entries in the data and n is the number of observations in the training sample [60].

Parameters like the number of trees or iterations, the rate at which the gradient boosting learns, and the depth of the tree can significantly affect the performance of the XGboost algorithm and they could be optimally selected through the hyper-parameter optimisation technique described in Section 4.2.3. The most important hyper-parameters of the XGBoost model are:

- `max_depth`: the maximum tree depth for the base learners,
- `subsample`: the subsample ratio of the training instances,
- `colsample_bytree`: the subsample ratio of columns when constructing each tree,
- `min_child_weight`: the minimum sum of instance weight required in every child node,
- `gamma`: the minimum loss reduction required to make another partition on a leaf node in the tree.

4.2.2 Feature importance using Tree SHAP

After training the random forest and XGBoost student models, we have employed SHAP tree explainer to obtain local and global interpretability for the predicted labels. Lundberg et al. [61] proposed TreeSHAP which is a faster model-specific approximation of KernelSHAP, that is applied to tree-based machine learning models. TreeSHAP has the computational advantage of estimating the SHAP values for tree-based models in polynomial time instead of exponential. Compared to KernelSHAP, this method reduces the computational complexity from $\mathcal{O}(TL2^M)$ to $\mathcal{O}(TLD^2)$, where T refers to the number of trees, L is the maximum number of leaves in any tree and D represents the maximal depth of any tree [43]. This is achieved by pushing all possible feature subsets down the tree at the same time [61].

Despite, its computational efficiency, this method has the disadvantage of generating inconsistent feature attributions. Empirical evidence shows that in some cases features that have no influence on the prediction function f can get a TreeSHAP estimate different from zero. The non-zero estimate can happen when the feature is correlated with another feature that actually has an impact on the prediction [43].

TreeSHAP produces local explanations for each observation in the test set. Global explanations are then estimated in line with Equation (4.3). Similar to deep explainer, TreeSHAP preserves the feature-additivity property. The global and local attributions at the group level are estimated by summing up the individual attributions of all the variables that fall under that group.

4.2.3 Implementation of knowledge distillation to our data

The random forest and XGBoost models are instantiated using the `scikit-learn` package in Python. Two generic approaches to parameter search are provided in `scikit-learn`: i) `GridSearchCV` which exhaustively considers all parameter combinations for a predetermined parameter grid and ii) `RandomizedSearchCV` which samples a given number of candidates (i.e. parameter combinations) from a parameter space with a specified distribution. For our use case, the parameter search space for grid searching is too big and the data is large, therefore a randomised search on the hyper-parameters is preferred. This method of finding optimal hyper-parameters discovers better models through searching a wider range of spaces, and in a reduced computational time [62].

Both grid search and random search, have successive halving counterparts in `scikit-learn`, which can be much faster at finding a good parameter combination.
`HalvingRandomSearch` uses a search strategy that starts evaluating all the candidates (i.e.

parameter combinations) with a small number of resources and iteratively selects the best candidates, using a greater number of resources. The candidates are sampled at random from the parameter space and the number of sampled candidates is determined by the `n_candidates` parameters. The `halving` parameter determining the fraction of candidates that should be selected for the next iteration is set to 2. K-fold cross validation with $k = 5$, is used to save and re-use partitions of the training set for a more effective hyper-parameter searching. The motivation behind the use of less accurate but faster methods to tune hyper-parameters is attributed to the large dataset. The best hyper-parameters found with the `HalvingRandomSearch` method for the random forest and XGBoost student models, are shown in Table 4.1 and Table 4.2.

Table 4.1: The optimal hyper-parameters found for the random forest student model.

Hyperparameter	**Grid of values**	**Best value**
`n_estimators`	`np.linspace(200,500,10,dtype=int)`	200
`max_features`	`['auto','sqrt']`	`'auto'`
`max_depth`	`np.linspace(10,110,10,dtype=int)`	10
`min_samples_split`	`np.linspace(2,20,10,dtype=int)`	14
`min_samples_leaf`	`np.linspace(2,20,10,dtype=int)`	18

Table 4.2: The optimal hyper-parameters found for the XGBoost student model.

Hyperparameter	**Grid of values**	**Best value**
`max_depth`	`np.linspace(5,15,10,dtype=int)`	12
`subsample`	`np.linspace(0.5,1,10)`	0.78
`colsample_bytree`	`np.linspace(0.5,1,10)`	0.55
`min_child_weight`	`np.linspace(10,25,10,dtype=int)`	25
`gamma`	`np.linspace(0,5,10)`	2.78

Note: The first parameter in the `np.linspace` function represents the start of the interval, the second parameter represents the end of the interval and the third parameter represents the number of values to draw from the pre-defined interval.

We have further set the `warm_start` attribute of the random forest model to `True` to make the training faster. When fitting an estimator repeatedly on the same data for multiple parameter values, it may be possible to reuse aspects of the model learned from the previous parameter value, in order to save time. When `warm_start` is set to true, the existing fitted model attributes are used to initialize the new model in a subsequent call to fit.

In algorithm 4, we provide a step-by-step implementation of the knowledge distillation approach to our LSTM model for spread predictions.

Algorithm 4: Implementation of knowledge distillation for the LSTM spread prediction model.

Input: train data $\{\mathbf{X}_{\text{train}}, \mathbf{y}_{\text{train}}\}$, test data $\{\mathbf{X}_{\text{test}}, \mathbf{y}_{\text{test}}\}$ and trained LSTM model f.

Returns: i) local explainability for each unique bond in the test sample and ii) global explainability.

Algorithm:

1. Load the original trained LSTM model f, train set feature matrix $\mathbf{X}_{\text{train}}$ and the test set feature matrix $\mathbf{X}_{\text{test}}$.

2. Obtain soft labels by running the predict method of the trained LSTM model on the $\mathbf{X}_{\text{train}}$, in order to get $\hat{\mathbf{y}}_{\text{train}} = f(\mathbf{X}_{\text{train}})$.

3. Use the soft labels obtained in Step (2), as target labels to train the random forest and XGBoost student models.

    - To train the random forest and XGBoost student models, we first need to reduce the dimensionality of the input matrix from 3d to 2d, by removing the temporal dimension. This is achieved by retaining only the last value of the 10-day sequence for each input variable. Then we instantiate the `fit` method on random forest and XGBoost models, using $\hat{\mathbf{y}}_{\text{train}}$ as target labels.
    - Search for the optimal hyper-parameters using `HalvingRandomSearch` method. Set the `warm_start` attribute of the random forest model to `True` to make the training faster.
    - Use the best random Forest and XGBoost models to obtain predictions on the test set i.e. $\hat{\mathbf{y}}_{\text{test}} = s(\mathbf{X}_{\text{test}})$, where s refers to the optimised student models.

4. Instantiate the `shap.TreeExplainer` object on the predictions of the best student models.

5. Use `shap.TreeExplainer` to obtain local explanations for each unique bond in the test set.

6. Use `shap.TreeExplainer` to obtain global explanations as suggested by Equation (4.3).

7. Estimate group-level importance by summing up the individual contributions of features that fall under each group.

4.3 Interpretable multivariable LSTM (IMV-LSTM)

The main limitation of the LSTM model described in Section 3.2, is its lack of interpretability. The LSTM model in use combines information from all variables into the hidden states of the network. Hence, it is intractable to distinguish the contribution of individual variables into model's prediction through the sequence of hidden states [63].

Guo et al. [14] introduced IMV-LSTM as a way to achieve a unified framework of forecasting and obtaining variable-wise importance scores. IMV-LSTM makes use of the hidden state matrix, such that each row of the hidden matrix encodes information exclusively from a certain variable. This provides a way to track the contribution of individual features in the predicted output by using a mixture attention mechanism that encapsulates the hidden state of each feature. More on this in Section 4.3.2.

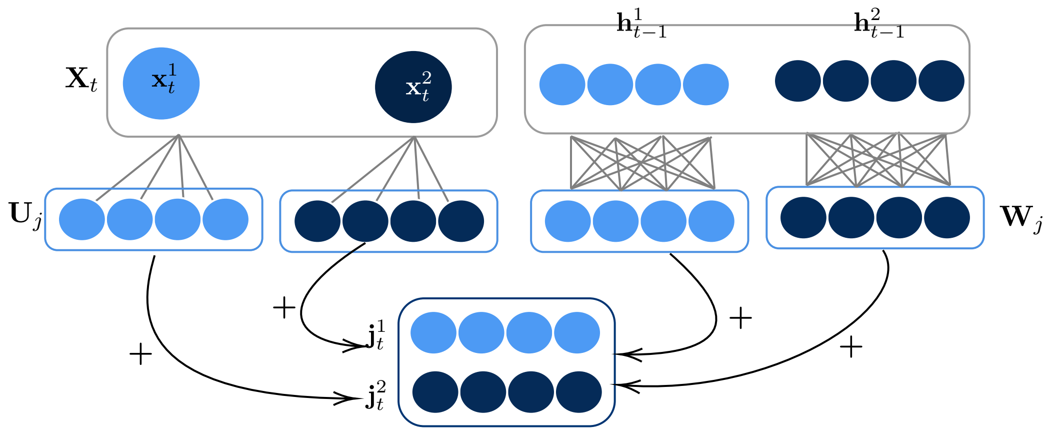
4.3.1 Updates to the existing LSTM

The main update to the standard LSTM network is that in IMV-LSTM each row of the hidden state matrix captures information exclusively from a specific variable. We have denoted the hidden state and gate matrices in the IMV-LSTM network with tildes, to differentiate from the notation used in the standard LSTM network in Section 2.1.1.

We define the hidden state matrix at time step t for a specific instance as $\tilde{\mathbf{H}}_t \in \mathbb{R}^m \times \mathbb{R}^u$, where m is the number of features and u is the number of hidden units. The row $\mathbf{h}_t^m \in \mathbb{R}^u$ of $\tilde{\mathbf{H}}_t$ is the hidden state vector that encapsulates information specific to input m.

The calculation of gate interactions and the update of cell states and hidden states is similar to the standard LSTM cell explained in Section 2.1.1. The only difference is that instead of using a hidden state matrix $\mathbf{H}_t$ which receives information from all the variables, we use matrix $\tilde{\mathbf{H}}_t$ where each row encodes information exclusively from a specific variable. This is further illustrated in Figure 4.3.

Figure 4.3: An illustrative example of the IMV-LSTM network with a two-variable input sequence and a hidden matrix with 4-dimensions per variable.



Light blue and dark blue colors correspond to two variables. U_j represent the input to hidden transition weights, W_j is the hidden to hidden transition weights and $\mathbf{j}_t$ shows the derivation of the hidden state update for each variable. The hidden state update is computed in line with Equations (2.1-2.6) outlined in Section 2.1.1.

4.3.2 Mixture attention mechanism for interpretability

Attention mechanisms in deep learning were first presented by Bahdanau et al. [64]. The main idea was that when the model predicts an output it only uses parts of the input where the most relevant information is concentrated instead of the entire sequence. With the attention mechanism framework, the model is able to selectively focus on valuable parts of the input sequence and hence learn the association between them. The learned attention is then used to enhance model's interpretability [64].

The idea of attention mechanisms was incorporated in the construction of the IMV-LSTM model. Once the IMV-LSTM cells are constructed, the importance scores can be calculated. After feeding a sequence of $\{\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_T\}$ where $\mathbf{x}_t \in \mathbb{R}^m$ is the input vector corresponding to one instance of the dataset, we obtain a sequence of hidden state matrices for each time step. The sequence of hidden state vectors specific to variable m is $\{\mathbf{h}_1^m ... \mathbf{h}_T^m\}$ [14]. The aim of the mixture attention mechanism is to obtain temporal variable-wise feature importance scores (i.e. importance scores for each variable at each time step). This is achieved in two steps:

1. Temporal attention is first applied to the sequence of hidden states corresponding to each variable, in order to obtain the summarized history of each variable [14]

$$\alpha_t^m = \frac{\exp(f_m(\mathbf{h}_t^m))}{\sum_k \exp(f_m(\mathbf{h}_k^m))} \tag{4.6}$$

   where $f_m(\cdot)$ is a function specific to variable m, e.g. neural networks.

2. Then by using the history of the hidden states for individual variables over each time step , variable-level attention is obtained

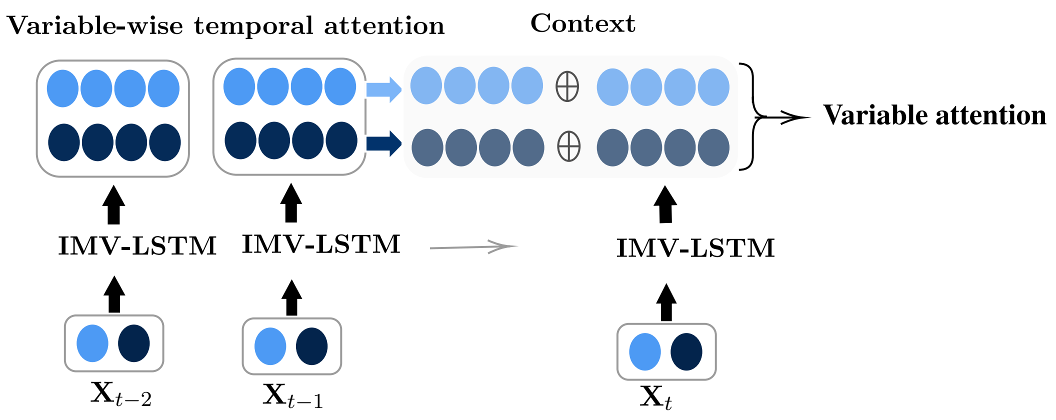
$$\mathbf{g}^m = \sum_t \alpha_t^m \mathbf{h}_t^m. \tag{4.7}$$

   The authors compute a context vector $\mathbf{g}^m$ for a particular feature m as the temporal attention weighted sum of hidden states of variable m [14].

Elements of the attention mechanism vectors are normalized (i.e. sum to one) to reflect the relative importance of the corresponding variable or the temporal importance with respect to the prediction.

A visual representation of the mixture attention mechanism is provided in Figure 4.4

Figure 4.4: The process where the inputs at the previous time step $\mathbf{X}_{t-1}$ are fed into the IMV-LSTM model, and explained using the mixture attention mechanism.



When the model sees a new input $\mathbf{X}_t$, then it adds further context to the attention mechanism,producing the variable attention scores for each feature.

4.3.3 Implementation of the IMV-LSTM model to our data

We have implemented the IMV-LSTM model and the mixture attention mechanism in `Pytorch`. To train the IMV-LSTM model we have used Adam optimizer [56].[1] See Section 3.2 for further details. We consider a loss function based on the mean absolute error (MAE) as the performance metrics. The reason why we have selected MAE and not other metrics

[1]Adam is a replacement optimization algorithm for stochastic gradient descent for training deep learning models as it is computationally efficient. Adam uses the most useful properties of the AdaGrad and RMSProp algorithms to provide an optimization algorithm that can handle sparse gradients on noisy problems.

such as MSE and RMSE is because MAE generates a more stable learning for our use case

$$\mathcal{L}(\mathbf{y}, \hat{\mathbf{y}}) = MAE = \frac{1}{N} \sum_{i=1}^{N} |y_i - \hat{y}_i|, \tag{4.8}$$

where $\mathbf{y}$ is the vector of target labels and $\hat{\mathbf{y}}$ is the vector of predicted labels.

The initial weights in the IMV-LSTM network are initialised via the Xavier method, due to its advantages in reducing the effect of the vanishing gradient problem for the hyperbolic tangent activation function [22]. Weights are sampled from a uniform distribution U

$$W \sim U\Big(-\frac{\sqrt{6}}{\sqrt{n_i + n_{i+1}}}, \frac{\sqrt{6}}{\sqrt{n_i + n_{i+1}}}\Big), \tag{4.9}$$

where n_i is the number of incoming connections into the layer, and n_{i+1} is the number of outgoing connections of the layer.

In order to find the best combination of hyper-parameters for the IMV-LSTM model, we have conducted random search over 10 iterations. This method consists of randomly using 10 combinations of hyper-parameters to train the model and ultimately selecting the combination that generates the lowest MAE in the validation set. This technique allows us to control the number of attempted hyper-parameter combinations, unlike grid search, where every possible combination is used [62].

The best combination of hyper-parameters for the IMV-LSTM model is:

Table 4.3: The optimal hyper-parameters found for the IMV-LSTM model.

Hyperparameter	**Grid of values**	**Best value**
`n_units` [2]	`np.linspace(10,30,10,dtype=int)`	14
`batch_size`	`np.linspace(32,128,20,dtype=int)`	122
`learningrate`	`np.linspace(1e-6,1e-5,20)`	9.52e-06
`step_size` [3]	`np.linspace(5,15,10,dtype=int)`	7
`gamma` [4]	`np.linspace(0.0001,0.001,10)`	3e-4
`epochs`	`np.linspace(10,50,20,dtype=int)`	47

In Algorithm 5, we provide a step-by-step implementation of the IMV-LSTM with the mixture attention mechanism.

Algorithm 5: Implementation of the IMV-LSTM model with the mixture attention mechanism.

Input: feature matrix $\mathbf{X}$, target vector $\mathbf{y}$, loss function $\mathcal{L}$, optimizer, epoch scheduler.

Returns:i) predictions of spread changes for each observation in the test set and ii) variable importance scores and variable-wise temporal importance.

Algorithm:

1. Create an IMV-LSTM model class that does the following:
   - Initialises the parameters of the IMV-LSTM model using Xavier initialisation method.
   - Updates the LSTM cell using the Equations (2.1-2.6) described in Section 2.1.1.
   - Uses the mixture attention mechanism to obtain variable-wise temporal importance in line with Equation (4.6) and variable importance in line with Equation (4.7).
2. Use random search optimisation for 10 iterations to find the best combination of hyper-parameters with the lowest MAE in the validation set.
3. Use the best combination of hyper-parameters for the IMV-LSTM model to obtain predictions in the test set.
4. Use the mixture attention mechanism to obtain variable-wise and temporal explainability of model's predictions.

4.4 LSTM-LRP model

LSTM-LRP provides an alternative model-specific interpretability method which integrates the LRP technique that is used to obtain explanations with the LSTM model which is used to generate predictions [15]. For a given input $\mathbf{x}$, LRP redistributes the output $f(\mathbf{x})$ progressively starting from the final layer of the network and backpropagating over lower layers until the input layer is reached. The computation of the relevance scores for each intermediate lower-layer neuron, depends on the type of the connection involved.

1. **Weighted connection** refers to the estimation of an upper layer neuron z_j as: $z_j = \sum_i z_i w_{ij} + b_j$, where z_i are the lower layer neurons and w_{ij}, bj are the connection weights and biases. Given the relevance R_j of the upper layer neurons z_j, the

relevance redistribution into lower-layer neurons is computed

$$R_i = \sum_j \frac{z_i w_{ij}}{z_j + \epsilon \ \mathrm{sign}(z_j)} R_j, \tag{4.10}$$

where ϵ is a positive parameter and $\epsilon \ \mathrm{sign}(z_j)$ acts as stabilizer that pushes the denominator away from zero, and has the effect of absorbing some relevance when the weighted activations are weak or contradictory [53]. A large value of ϵ tends to keep only the most salient factors of explanation.

2. **Multiplicative connection** refers to the estimation of the upper layer neuron z_j as the product of two lower layer neuron values z_g and z_s, i.e. $z_j = z_g z_s$. In such multiplicative interactions, as present in the computation of input, forget and output gates in the LSTM cell, z_g refers to gate neurons whose value ranges between $[0, 1]$ because of the sigmoid activation function, and z_s is the source neuron.

4.4.1 Extending LRP to LSTMs

A standard LSTM network with fully connected layers, as presented in Section 2.1.1 is very powerful in modelling complex non-linear sequential tasks. However, this model is difficult to interpret. With the further goal of increasing model's interpretability Arras et al. [15], propose some modifications which simplify the interface between the LSTM network and LRP explainability method.

The authors suggest that the LRP backward propagation procedure is made simpler if memory cell states $\mathbf{c}_t$ are non-decreasing. In this way the contribution of each input to each memory cell is well-defined, and the problem that a negative and a positive contribution will cancel each other out is avoided. For non-decreasing memory cells and backward analysis with LRP, the updated forward pass rules for a single LSTM cell are changed under the assumptions outlined in [15]

$$\mathbf{z}_t = a_g \sigma(\mathbf{w_z x}_t + \mathbf{b_z}) \qquad \text{cell input} \tag{4.11}$$

$$\mathbf{i}_t = \sigma(\mathbf{u_i y}_{t-1} + \mathbf{b_i}) \qquad \text{input gate} \tag{4.12}$$

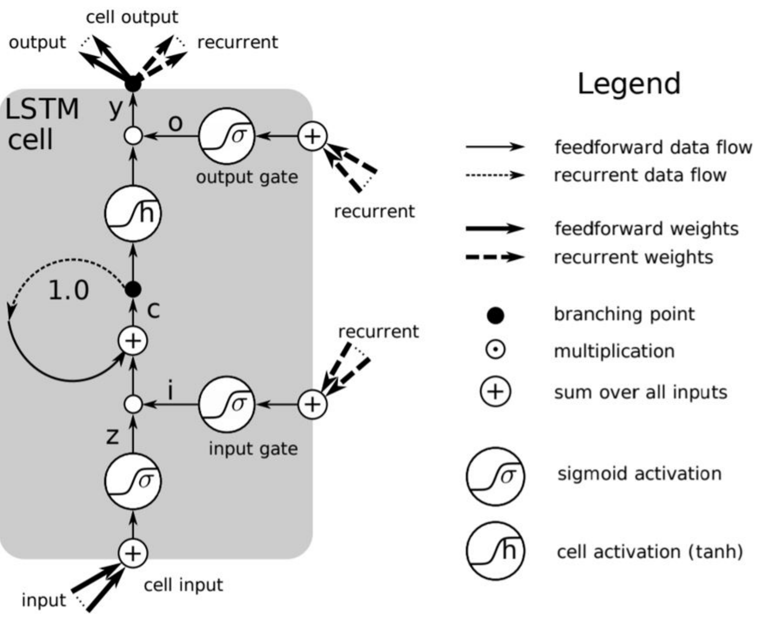
$$\mathbf{c}_t = \mathbf{i}_t \odot \mathbf{z}_t + \mathbf{c}_{t-1} \qquad \text{cell state} \tag{4.13}$$

$$\mathbf{o}_t = \sigma(\mathbf{u_o y}_{t-1} + \mathbf{b_o}) \qquad \text{output gate} \tag{4.14}$$

$$\mathbf{y}_t = \mathbf{o}_t \odot a_h \tanh(\mathbf{c}_t) \qquad \text{cell output,} \tag{4.15}$$

where $\mathbf{z}$ is the cell input activations, $\mathbf{i}$ is the input gate activations, $\mathbf{o}$ is the output gate activations, $\mathbf{y}$ is the cell output activations, $\mathbf{c}$ is the memory cell states, $\mathbf{w}$ are the weights between the inputs and the gates, **u** are the weights between the cell output activations with a one-step time delay and the output gate, **b** is the bias, and $\odot$ is the element-wise product [15]. Figure 4.5 depicts the forward pass rules outlined above for the LSTM-LRP cell.

Figure 4.5: The architecture of the LSTM-LRP cell [49].



4.4.2 Implementation of LSTM-LRP to our data

The main idea is to pre-train the network in a vanilla LSTM network, using the training set, and tuning with the validation set for the optimal weights. Then, to assemble the importance score the LSTM-LRP model is used on the testing set. The implementation used in the original paper is a bi-directional LSTM network presented by Warnecke et al. [65], with a TensorFlow support. The bi-directional LSTM was converted to a uni-directional LSTM for the purpose of creating a sensible spread prediction model, given that the ability to look ahead to the future should be removed for financial applications.

The updated LSTM model takes the same three-dimensional input matrix as the original spread prediction model, and outputs the predicted spread one time-step ahead. The LSTM-LRP network has a similar architecture to the vanilla LSTM network described in Section 3.2. It consists of one hidden layer of 58 LSTM-LRP cells, and an output layer of one summation cell with no activation function and no bias. Between the hidden layer, and the output layer there is a flattening layer since the inputs to the summation cell must be in 2d and not 3d. The initial weights in the LSTM-LRP network are initialised via the Xavier method as explained in Section 4.3.3. Hyperparameter tuning is done using randomised search over 5 iterations, where the number of epochs, learning rate, batch size, and momentum were searched over. The optimal hyper-parameters are given in Table 4.4.

Table 4.4: The optimal hyper-parameters for the LSTM-LRP model.

Hyperparameter	Grid of values	Best value
`momentum`	`[0,0.2,0.4]`	0.2
`learning_rate`	`[1e-1,1e-3,1e-6]`	1e-3
`epochs`	`[10,50,100]`	10
`batch_size`	`[32,64,128]`	64

Training of the models is implemented with early stopping where if the validation loss is the same for 25 epochs, then training stops. After training and testing the model, importance scores are distributed iteratively starting from the output layer which equals the predicted output, and assigning each neuron in the lower layers of the network an importance score, until reaching the input layer. These values are then used as the local importance score for each feature, as defined in Equation (4.10).

We have briefly summarised the implementation of this approach in Algorithm 6.

Algorithm 6: LSTM-LRP Implementation

1. Train vanilla LSTM network on training set and optimise the hyperparameters with the validation set.
2. Do one pass over the LSTM-LRP network, and obtain the predicted output which is the predicted spread changes.
3. Estimate relevance scores based on the LRP method starting from the outputs to the inputs to get importance score for each feature.

4.5 Assessment framework

In this section we describe a framework to assess the practical application of the XAI methods to our LSTM spread prediction model. Ultimately, one of the objectives of this thesis is to recommend to Intellibonds an explainability method that is computationally efficient and gives intuitive explanations. To compare the XAI methods described in the "Methodology" section, we consider a number of desirable properties:

1. **Interpretability** identifies whether the method generates intuitive explanations. To measure intepretability we qualitatively check whether the ranking of the most important features confirms the expectations of the academic literature and the industry players.

2. **Algorithmic complexity** describes the computational complexity of the method. This property is an important consideration especially when handling large data, as it is the case for our spread prediction problem.

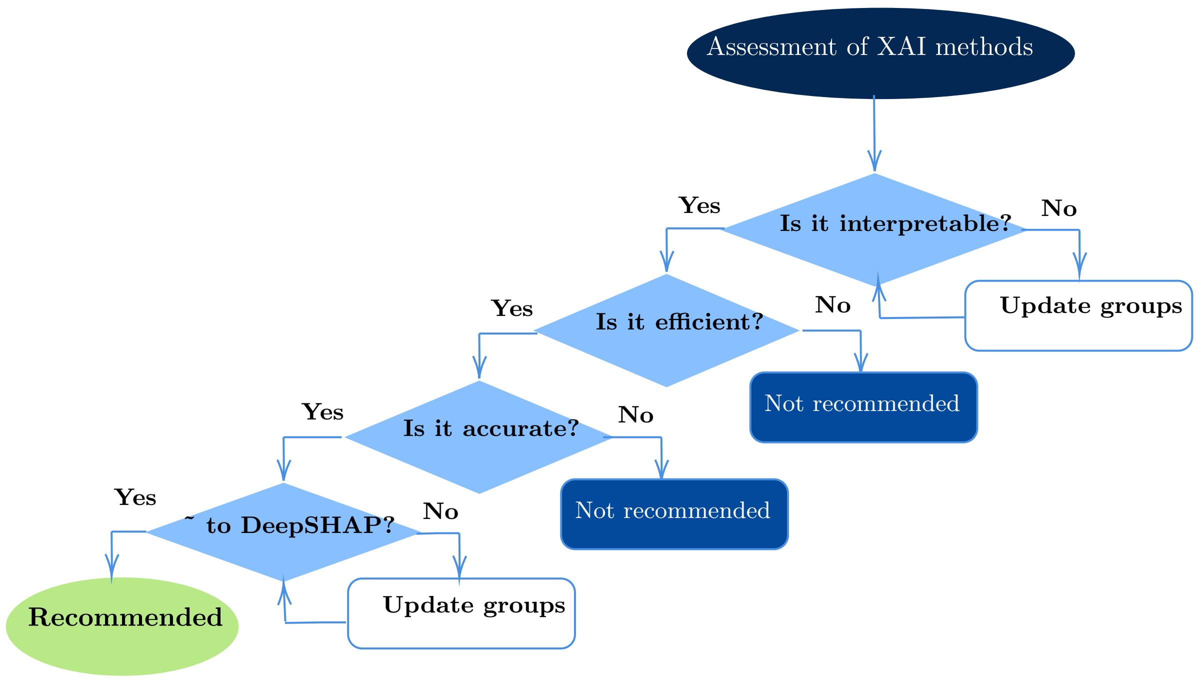
3. **Accuracy** answers the question: how well does an explanation method predict on unseen data? High accuracy is especially important if the explanatory method is used to generate predictions in place of the teacher model.

4. **Fidelity** identifies how well the explanations approximate the predictions of the black box model.

5. **Ranking similarity** measures how well the feature importance ranking generated by XAI methods match the ranking obtained by the DeepSHAP method outlined in Section 4.1. The closer the feature ranking is to DeepSHAP, the more reliable is the method. Ideally, one would use DeepSHAP to obtain explainability of the spread predictions model, as DeepSHAP directly accounts for the predictions of the original LSTM model and it is considered an accurate approximation of the Shapley values. However its computational inefficiency impedes the practical application of this method.

We have summarised in Figure 4.6 a step-by-step approach to select the most appropriate method for our use case.

Figure 4.6: Framework to evaluate XAI methods.



Note: Given that the importance scores are computed and reported at the group-level, it may be worth regrouping the features when ranking similarity or interpetability is low, prior to excluding the method entirely.

Chapter 5

Results and Discussion

This chapter evaluates the performance of the explainability methods outlined in Chapter 4, based on the assessment criteria explained in Section 4.5.

1. First, we report the importance scores for each method and analyse how intuitive the explanations are, by comparing the top ranked features with the theoretical findings from the literature and the industry's domain knowledge.

2. We then evaluate the algorithmic complexity of each method based on the computational time it takes to generate local and global explanations.

3. This is followed by an analysis of accuracy based on performance metrics such as mean absolute error and mean squared error. We compare the performance of XAI methods with the performance of the original LSTM model.

4. Furthermore, we estimate the fidelity of each method by comparing how close the predictions of the XAI methods are to the predictions of the original LSTM model. We also compare the similarity of local explanations to the DeepSHAP method for a randomly selected unique ISIN in the test sample.

5. We have also evaluated the global ranking similarity to the DeepSHAP method. Ranking similiarity is estimated as the number of feature groupings that were assigned to the same percentile as in DeepSHAP.

The dataset used to generate the results reported in this chapter is a subset of the original data which consists of 70,000 bonds in the training set, 20,000 in the validation set, and 30,000 in the test set. Across the three subsets, the bonds cover approximately 10 months of data from August 2020 to May 2021. The results reported are computed in a `GoogleColab` environment.

We have summarised the performance of XAI methods in Table 5.1. The results suggest that knowledge distillation where XGBoost is used as the student model, outperforms the other explainability methods.

Table 5.1: Summary of results for different XAI methods.

	Interpret-ability	**MSE**	**MAE**	**Training speed (GPU)**	**Ranking Similarity**
Original LSTM model	-	0.269	0.155	-	-
			Explainability Methods		
SHAP Deep Explainer	☑	0.269	0.155	1h 5min 29s	baseline
Knowledge Distillation (Random Forest)	☑	0.215	0.144	6min 20s	10
Knowledge Distillation (XGBoost)	☑	0.232	0.144	36s	9
IMV-LSTM	-	0.275	0.154	1h 8m 39s	6
LSTM-LRP	-	0.285	0.224	2h 20m 38s	6

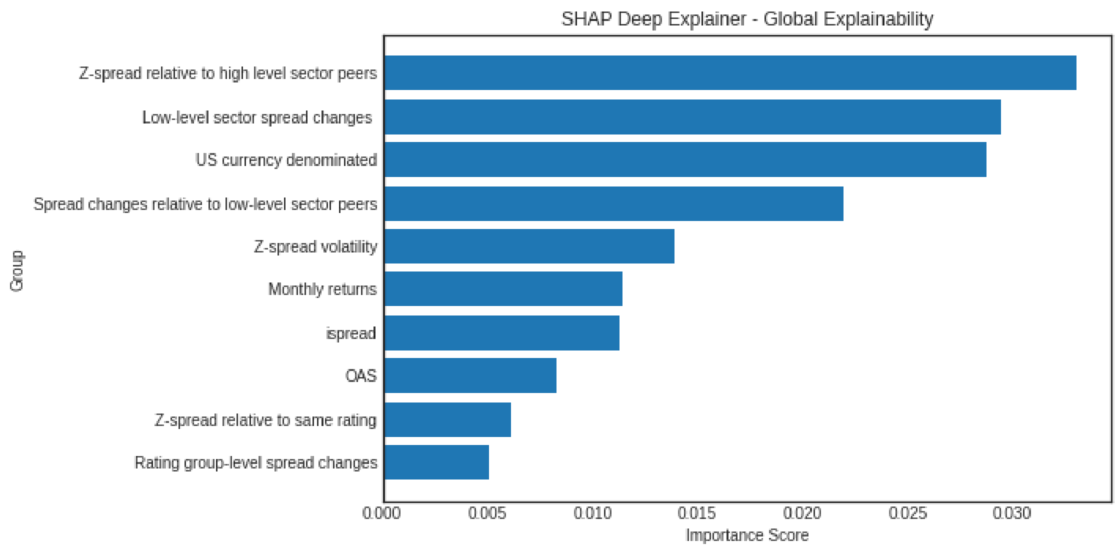
5.1 Results

The methods described in Chapter 4 generate local explanations for each unique ISIN in our sample, while global explanations are obtained by aggregating the local information. For SHAP-based explanations (i.e. DeepSHAP and knowledge distillation), global explanations are obtained in line with Equation (4.3). For IMV-LSTM and LSTM-LRP, global importance scores are estimated by averaging out the local explanations obtained for each bond on the test set. These two methods generate temporal importance scores for each variable. The importance score at the feature-level is then estimated by summing up the values across the 10-day sequence corresponding to that feature. For the sake of simplicity we have reported only the global explainability results, focusing on the top 10 most important feature groupings for each method.

5.1.1 Global explanations

We observe that the top-ranked features based on the DeepSHAP method contain mostly groups with one feature (i.e. "US curency denominated", "z-spread volatility", "monthly returns", "ispread", "OAS"). This reinforces the idea that groups with multiple features have a weakened effect in explaining importance to the prediction. This may be due to signed importance scores cancelling each other out when computing the sum at the group-level. Groups with only one feature are also easier to understand. The most important feature identified by DeepSHAP is "Z-spread relative to high level sector peers", which implies that a bond's performance relative to its industry peers strongly affects the predicted change in z-spread for that bond.

Figure 5.1: Global explainability for SHAP's deep explainer.

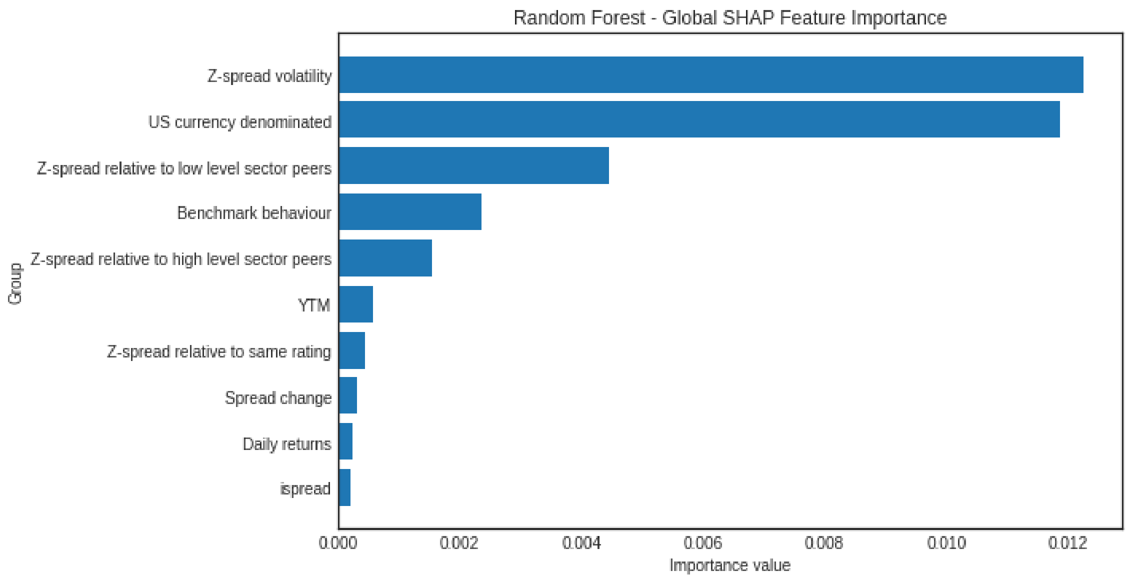


For the knowledge distillation method, the top ranked groups are fairly consistent across the two student models. We observe that the most important features in predicting spread changes are: "z-spread volatility","benchmark behaviour", "US currency denominated", and "z-spread relative to low level sector peers". We also observe that the ranking similarity to DeepSHAP for this method is higher than the other models. Specifically, we observe that for the random forest model, 10 feature groupings out of 32 are categorised in the same ranking percentile as in DeepSHAP, while for the XGBoost model the ranking similiarity to DeepSHAP is 9. This is much higher than the model-specific methods, where only 6 features are ranked in the same percentile as in DeepSHAP. [1]

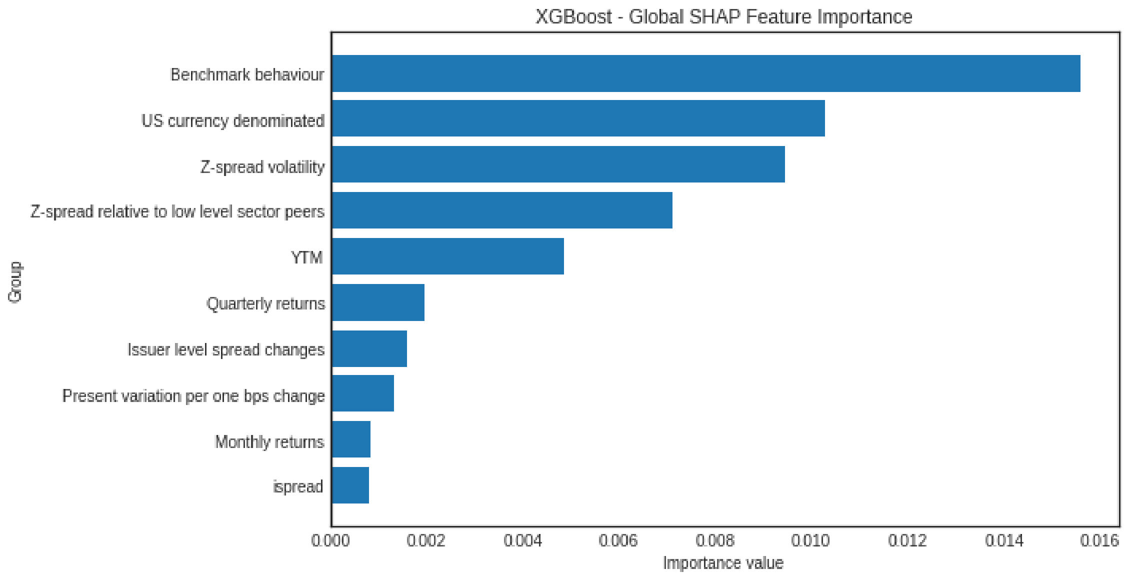
[1]The ranking similarity analysis answers the question: how many features are ranked in the same percentile as in DeepSHAP? We have split the feature ranking in four percentiles for each method i.e. features that belong to the 25th percentile are the ones with a ranking score ≤ 8. We have then counted the number of features that were ranked in the same percentile as in Deep SHAP.

Figure 5.2: Global explainability for knowledge distillation.

(a) Random Forest



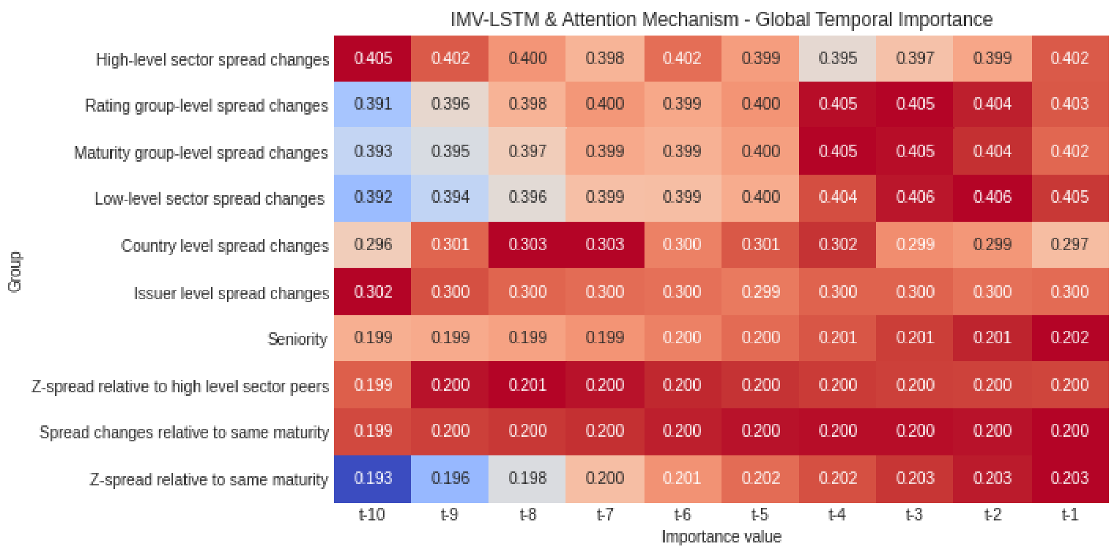
(b) XGBoost



The top groups identified by the LSTM-LRP and the IMV-LSTM method are generally in line with the results presented so far. However, the key indicator of “z-spread volatility” is not ranked in the top ten. In these two model-specific approaches, the importance scores of each group can be tracked over the ten-day sequence. This is not available for the other methods since only the last day of the ten-day sequence is considered to generate the scores. For LSTM-LRP, the importance scores are most prominent on the first day. On the other hand, “spread changes relative to same rating” and “ispread” are also important on day three and four. For the IMV-LSTM, some groups become more important as the sequence progresses. Most notably on days seven to nine, the importance scores become more prominent for most of the groups in our sample. This demonstrates one advantage of the mixture attention mechanism where the most recent data have a higher impact on the feature importance scores compared to earlier observations.

Figure 5.3: Global explainability for IMV-LSTM.

(a) Temporal importance



(b) Variable importance

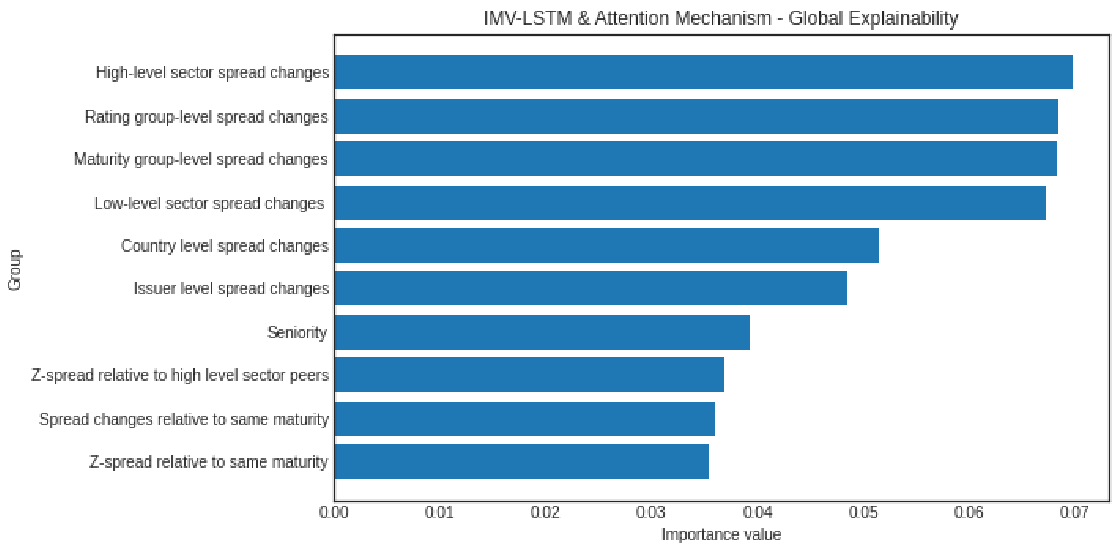
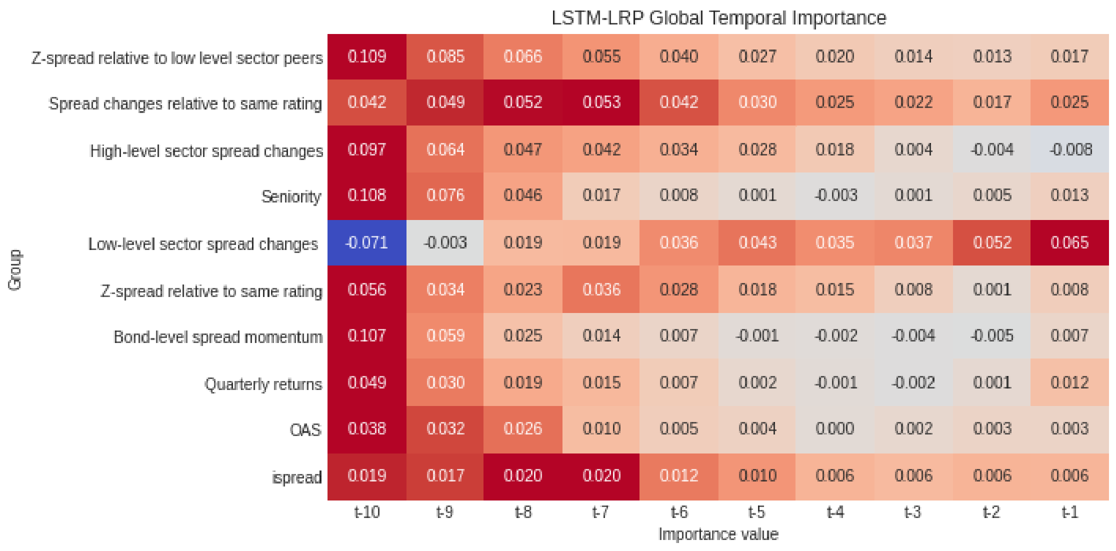


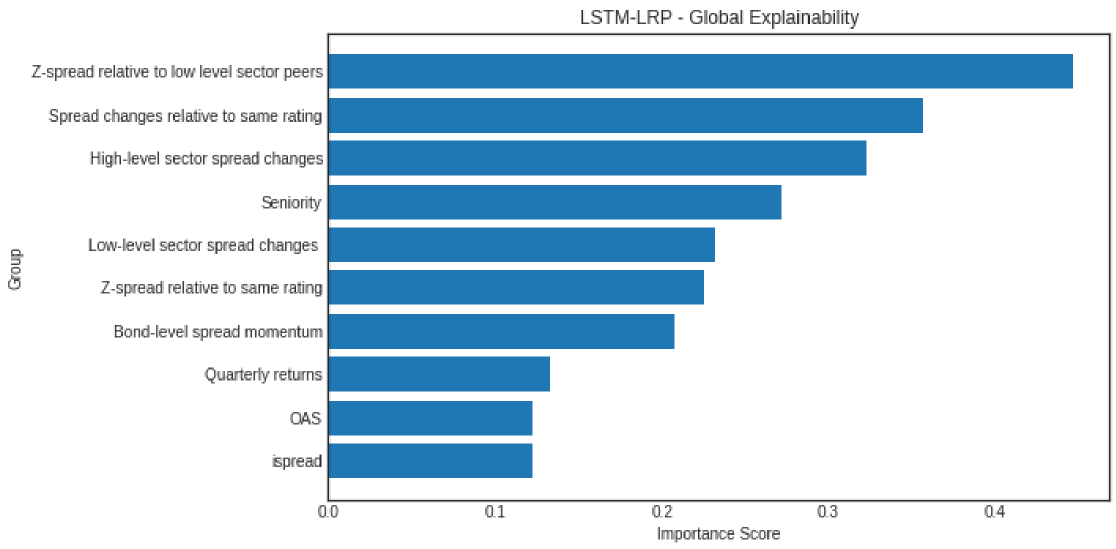
Figure 5.4: Global explainability for LSTM-LRP.

(a) Temporal importance



Group	t-10	t-9	t-8	t-7	t-6	t-5	t-4	t-3	t-2	t-1
Z-spread relative to low level sector peers	0.109	0.085	0.066	0.055	0.040	0.027	0.020	0.014	0.013	0.017
Spread changes relative to same rating	0.042	0.049	0.052	0.053	0.042	0.030	0.025	0.022	0.017	0.025
High-level sector spread changes	0.097	0.064	0.047	0.042	0.034	0.028	0.018	0.004	-0.004	-0.008
Seniority	0.108	0.076	0.046	0.017	0.008	0.001	-0.003	0.001	0.005	0.013
Low-level sector spread changes	-0.071	-0.003	0.019	0.019	0.036	0.043	0.035	0.037	0.052	0.065
Z-spread relative to same rating	0.056	0.034	0.023	0.036	0.028	0.018	0.015	0.008	0.001	0.008
Bond-level spread momentum	0.107	0.059	0.025	0.014	0.007	-0.001	-0.002	-0.004	-0.005	0.007
Quarterly returns	0.049	0.030	0.019	0.015	0.007	0.002	-0.001	-0.002	0.001	0.012
OAS	0.038	0.032	0.026	0.010	0.005	0.004	0.000	0.002	0.003	0.003
ispread	0.019	0.017	0.020	0.020	0.012	0.010	0.006	0.006	0.006	0.006

(b) Variable importance



5.1.2 Local explanations

For completeness, we have also reported local explainability results for a randomly selected bond in our test sample. An important advantage of SHAP-based explanations is the feature additivity property (i.e. the sum of importance scores for a given instance across all features gives the predicted value of the spread changes). This property enables intuitive interpretations for the user as the magnitude and sign of each score is directly related to the effect and directional impact that the feature has on the predicted value for each bond.

Figure 5.5, shows that DeepSHAP identifies "Spread changes relative to low-level sector peers" as the most important factor in predicting spread changes for the selected bond. It has a positive contribution to the predicted output, which suggests that if the change in z-spread for the selected bond is higher (lower) than the sector peers, our model predicts an increase (decrease) of the spread over the next time step. This intuitively makes sense as it suggests that the market expectation on future spread values incorporates evidence on bonds' relative performance. A positive spread change relative to sector peers suggests an increase level of the relative risk which in turn results in higher predicted spread changes for that bond.

The waterfall plots for SHAP-based explanations display importance scores for individual predictions. The index of the randomly selected bond in our test sample is 22086. The bottom of a waterfall plot starts as the expected value of the model output, and then each row shows how the positive (red) or negative (blue) contribution of each feature moves the value from the expected model value to the predicted output for the selected bond.

Figure 5.5: Local explainability for SHAP's deep explainer for a randomly selected bond in the test sample.

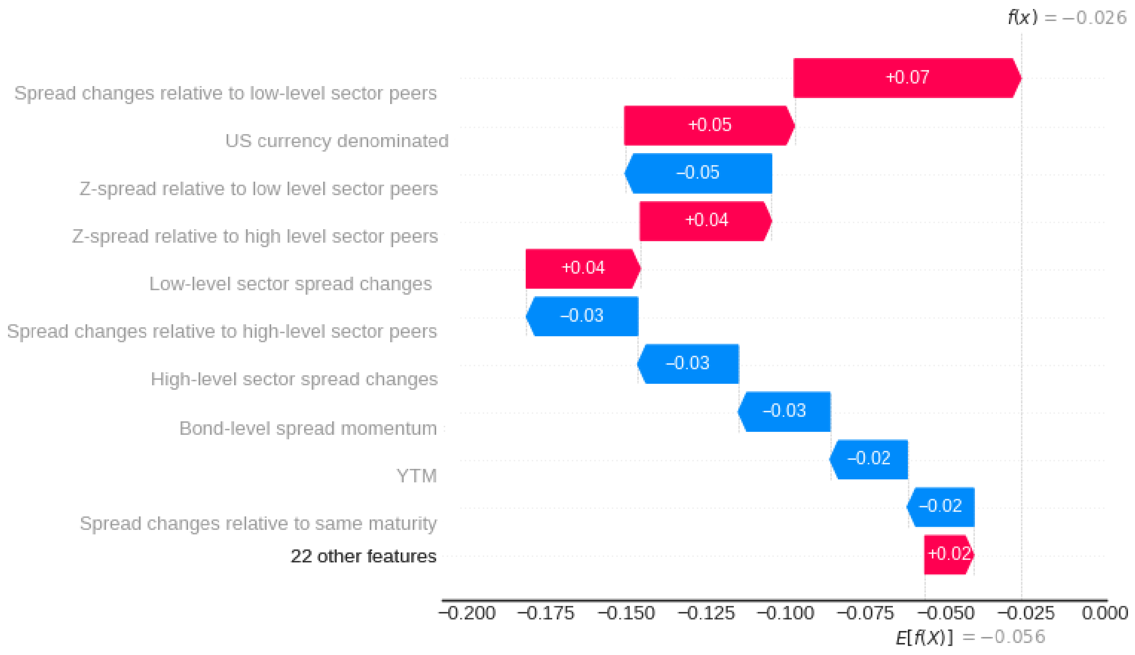
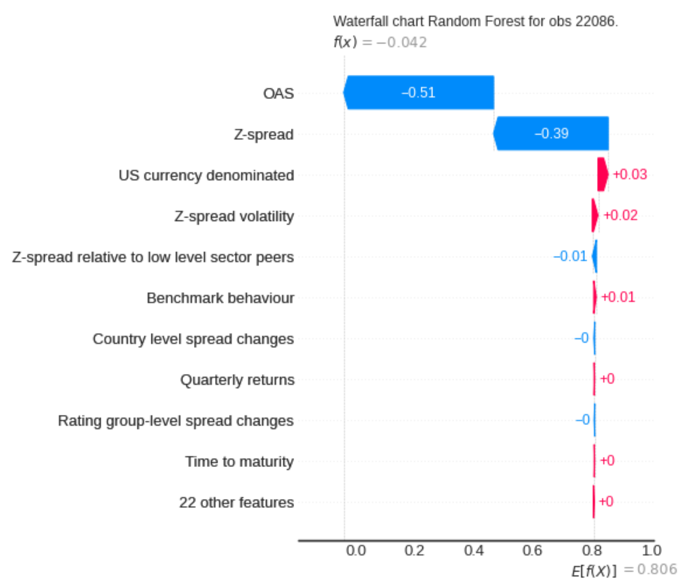


Figure 5.7 reports the local explainability results based on the knowledge distillation method. This method also inherits the feature additivity property from TreeSHAP which is used to generate local explanations. There is a high degree of consistency in feature rankings obtained from the two student models, albeit the rankings differ significantly from the DeepSHAP method. This could be due to the selected bond not being sufficiently representative of the test sample. Despite the inconsistency, the factors identified by these models make intuitive sense. For instance, the most important factor is "OAS", which

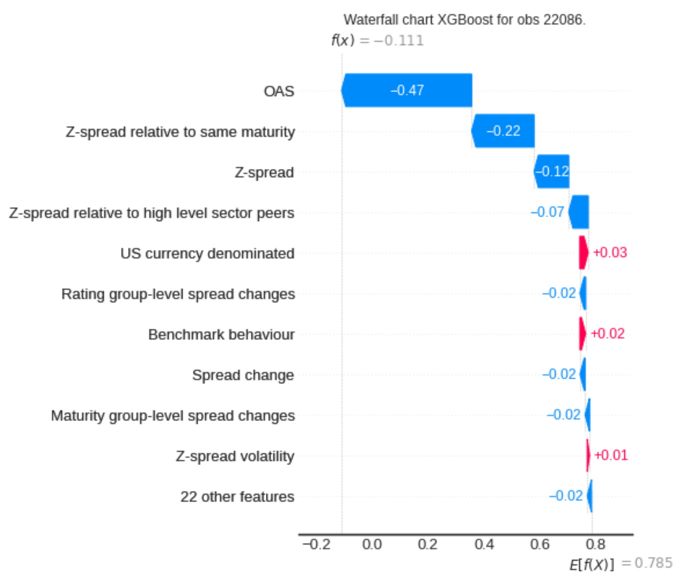
represents the option adjusted spread for the selected bond. "OAS" is shown to have a negative contribution on the predicted spread changes, implying that a high(low) "OAS", would contribute to the decrease(increase) in predicted spread changes generated by the student models. This makes intuitive sense as it supports the evidence on the mean reverting behaviour of corporate bonds spreads as reported in Hong et al. [66].[2]

Figure 5.6: Local explainability for the selected bond based on the knowledge distillation method.

(a) Random forest



(b) XGBoost

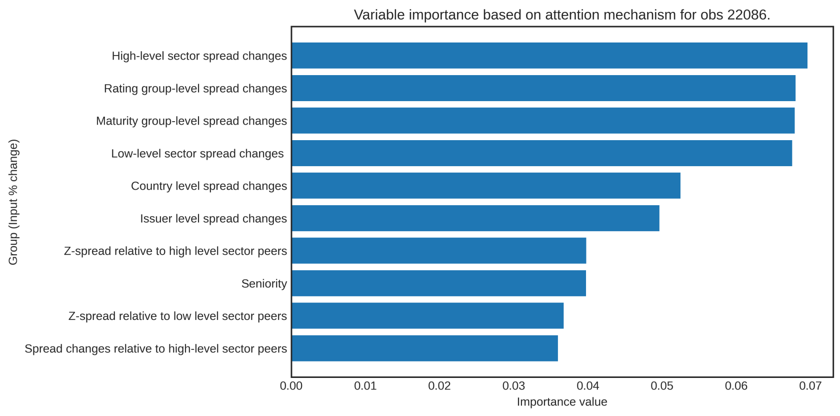


[2]Z-spread has a high degree of correlation with OAS (correlation coefficient is > 0.7). If values of OAS are higher than the long term mean, then the evidence on the mean reverting behaviour of bonds spreads would suggest a pullback to the historical mean, which implies a decrease in z-spread changes.

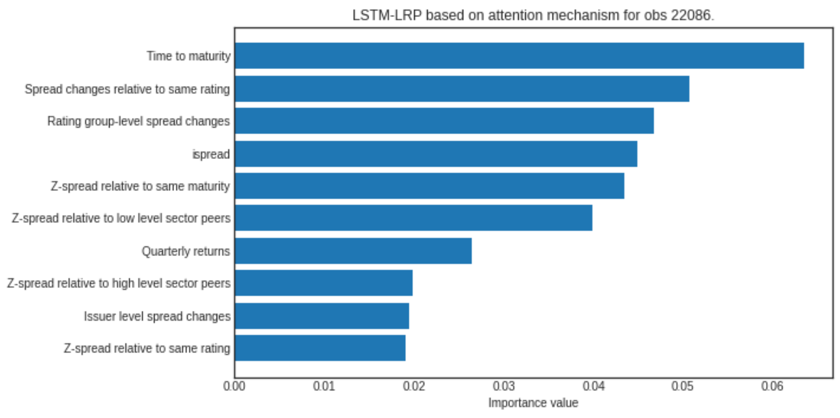
Contrary to SHAP-based explanations, model-specific methods are not compliant with the feature-additivity property, thus adding another layer of complexity in interpreting their results. For IMV-LSTM, the attention mechanism can only generate positive importance scores. This makes it harder for the user to understand the directional impact of each feature in the predicted spread. This is not the case for LSTM-LRP since there are positive and negative importance scores. There is no obvious similarity between the feature rankings obtained from the IMV-LSTM and LSTM-LRP methods. We observe that based on the IMV-LSTM method, "High-level sector spread changes" is recognised as the most important variable. The interpretation of this factor is intuitive for the user and it is in line with the logic applied for the DeepSHAP method above. On the other hand, LSTM-LRP identifies "Time to maturity" as the most important factor. However, there is no clear expectation on the relationship between time to maturity and predicted spread changes. In general, LSTM-LRP generates less intuitive explanations for the unique bonds in our test sample.

Figure 5.7: Local explainability for the selected bond based on the model-specific methods.

(a) IMV-LSTM



(b) LSTM-LRP



5.2 Interpretability

There is no prescribed framework in the literature to evaluate the interpretability of XAI methods, so the process of identifying which methods provide the most interpretable results is inherently subjective. We have attempted to systematically assess the interpretability of each method by checking whether the ranking of the most important features is aligned with: i) the literature on credit spread spread determinants and ii) the the domain knowledge of industry players. The analysis presented in this section is based on the global explainability results.

There is a general view in the literature that the underlying drivers of z-spread changes include macroeconomic factors affecting the bond issuer as well as the bond itself. These factors consist of the credit risk, liquidity, taxation, risk-free risk etc. See Appendix 7.4, for further information. We have summarised in Table 5.2, the extent to which the explanations obtained from each method are aligned with the general consensus.

Table 5.2: Important factors in determining z-spread changes based on the academic literature and market knowledge (not ranked in the order of importance).

Factor	Deep SHAP	Knowledge distillation (Random Forest)	Knowledge distillation (XGBoost)	IMV-LSTM	LSTM-LRP
Risk-free rate	-	✓	✓	-	-
Slope of default-free term structure	-	-	-	-	-
Asset price	-	-	-	-	-
Asset volatility	✓	✓	✓	-	-
Liquidity	✓	✓	✓	✓	✓
Taxation	✓	✓	✓	✓	-
Credit rating	✓	✓	✓	✓	✓
Maturity	✓	✓	✓	✓	✓
Market risk	✓	✓	✓	✓	✓
Bond characteristics	✓	✓	✓	-	✓

We observe that explanations obtained from DeepSHAP and knowledge distillation are largely in line with the theoretical and market expectations. For instance knowledge distillation based on the XGBoost student model identifies “Benchmark behaviour” as the most important factor. This factor implicitly accounts for the impact of the “Risk-free rate”, which is recognised as one of the most important determinants of credit-spreads in the literature. See Appendix 7.4. Moreover, the “US currency denominated” variable (ranked 2nd in the order of importance), captures the impact of taxation and liquidity, given that a dollar-denominated bond issuance is subject to the US taxation law and it is generally more liquid compared to other currencies.

“Z-spread volatility” (ranked 3rd in the order of importance), represents the rolling standard deviation of the z-spread for a given bond. It implicitly captures the impact of the "asset volatility" factor which is recognised in the literature. "Issuer-level spread changes" which shows changes in z-spread averaged across all bonds emitted by a given issuer, is another variable directly related to "asset volatility", as higher spread-changes at the issuer level would suggest higher asset volatility.

Moreover, the "z-spread relative to low level sector peers", compares the spread a given bond relative to bonds of the same sector. This can be perceived as a proxy of bonds' credit risk relative to market risk. The effect of "credit rating", "maturity" and "market risk", are all embedded in this variable. Bonds with higher credit rating and lower maturity have generally lower z-spread compared to their sector peers. Other variables such as: "YTM", "quarterly returns", "present variation per one bps", "monthly returns" and "ispread" are all related to bond-specific characteristics, which are important determinants in predicting changes in the z-spread for a given bond in our sample. A similar logic to determine the consistency of explanations with the academic and market expectations, is applied to other methods.

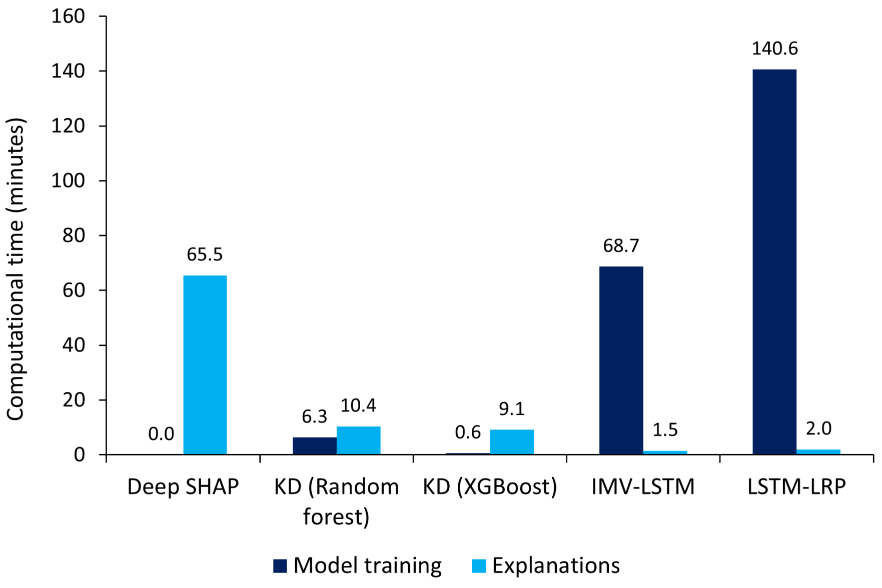
5.3 Computational efficiency

In this section we evaluate the computational efficiency of each method based on the time it takes to train the models and to generate local and global explanations. The purpose of this study is to create an interface between the spread prediction model and the interpretability analysis. Considering the magnitude of the data (i.e. a total of 4.57 million observations), a very important consideration is the efficiency of each method. It would not be feasible to rely on methods that require a considerable computational time no matter how accurate or interpretable they are.

XGBoost is the fastest model to train (in 36.4 seconds), whereas LSTM-LRP is the slowest model to train (in 2h 20min 38s). In terms of generating local and global explanations for the bonds in our sample, IMV-LSTM requires the least amount of time (1.5 min) compared to DeepSHAP which requires approximately 1h 5min 29s (DeepSHAP does not require training as it is directly applied to the predictions of the original LSTM model).

Overall, knowledge distillation based on the XGBoost student model is the fastest method to train **and** obtain explanations as it requires only 9min 43s, compared to LSTM-LRP which is the slowest method requiring a total of 2h 22m 38s. Figure 5.8 summarises the computational time required to train the model and to obtain explanations, for all the XAI methods under consideration.

Figure 5.8: Computational time for model training and explainability results.



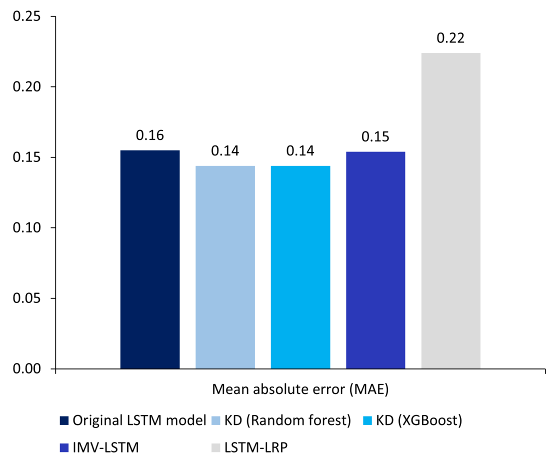
5.4 Accuracy

SHAP's deep explainer uses the original spread prediction model to obtain explanations therefore it achieves both accuracy and fidelity. We have measured the accuracy of XAI methods using MSE and MAE performance metrics. Knowledge distillation with the tree ensemble methods as student models, achieve higher accuracy than the other explainability methods. Knowledge distillation based on the random forest model achieves even better accuracy than the original LSTM model, as evidenced by the lower MSE and MAE values.

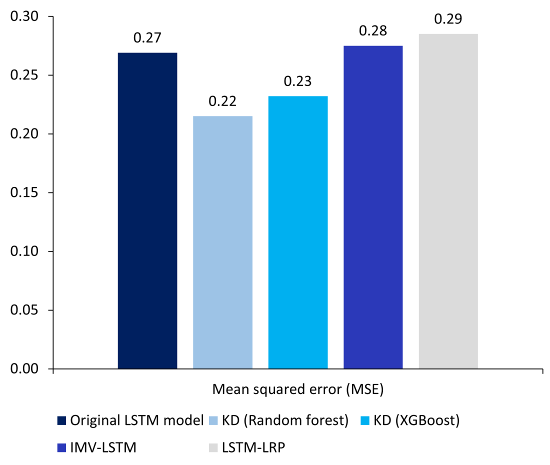
The MAE for both student models is the same but MSE suggests that random forest is performing better than XGBoost in terms of predictive accuracy. The higher accuracy of the random forest model compared to XGBoost, is expected since there are a greater number of hyperparameters available for optimisation. Moreover, only a subsample of the full dataset is used to generate these results, and random forest performs better in data with high noise. However, this effect is only marginal and it may be dependent on the sample size. If the entire sample is used XGBoost may achieve higher accuracy.

Contrary to the belief that model-specific methods would achieve higher accuracy, IMV-LSTM and LSTM-LRP have higher MAE and MSE compared to other methods. In particular, LSTM-LRP has the worst predictive performance out of the four explainability methods. See Figure5.9 for a comparison of MSE and MAE values across the different XAI methods.

Figure 5.9: Accuracy of XAI methods based on the MAE and MSE metrics.



(a) Mean absolute error



(b) Mean squared error

5.5 Fidelity

Fidelity is concerned with how well the model explanations approximate the predictions of the black box model. For the knowledge distillation approach and model-specific methods we are not using the original LSTM model to obtain spread predictions, therefore it is important to measure how well these methods approximate the predictions of the original LSTM model. As a measurement of fidelity we have computed the mean absolute deviations between the predictions generated from XAI methods and the LSTM predictions

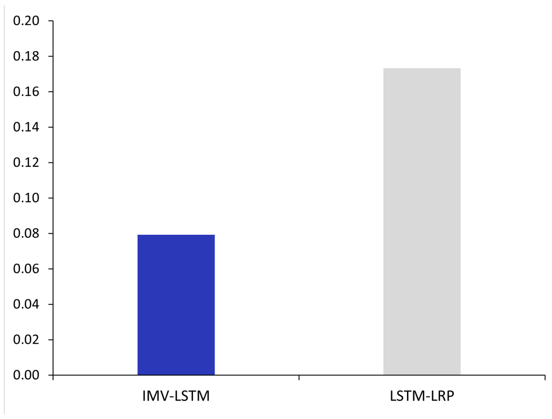
$$\mathrm{MAD} = \frac{1}{N}\sum_{i} |\hat{y}_i - \tilde{y}_i|, \tag{5.1}$$

where $\hat{y}_i$ is the predicted spread change obtained from the original LSTM model on observation i of the test set, and $\tilde{y}_i$ is the prediction obtained from the XAI method.

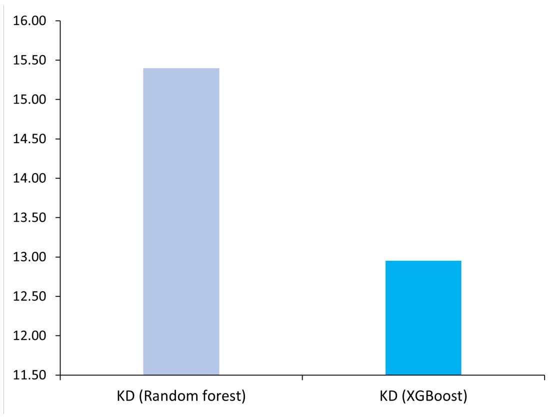
A low MAD score, implies a high level of fidelity because it suggests that the predictions of the XAI method are on average close approximates to the predictions of the original spread model. As expected, model-specific methods have higher fidelity compared to the knowledge distillation models because their network architecture has a greater resemblance to the original LSTM model. Out of the two model-specific methods IMV-LSTM has the lowest MAD score (i.e. highest fidelity).

When comparing the two knowledge distillation models, we observe that XGBoost has a lower MAD score, thus ensuring a higher level of fidelity to the original LSTM model. These results are summarised in Figure 5.10.

Figure 5.10: Fidelity of XAI methods measured as the mean absolute deviations between the predictions generated from XAI methods and the LSTM predictions.



(a) model-specific methods



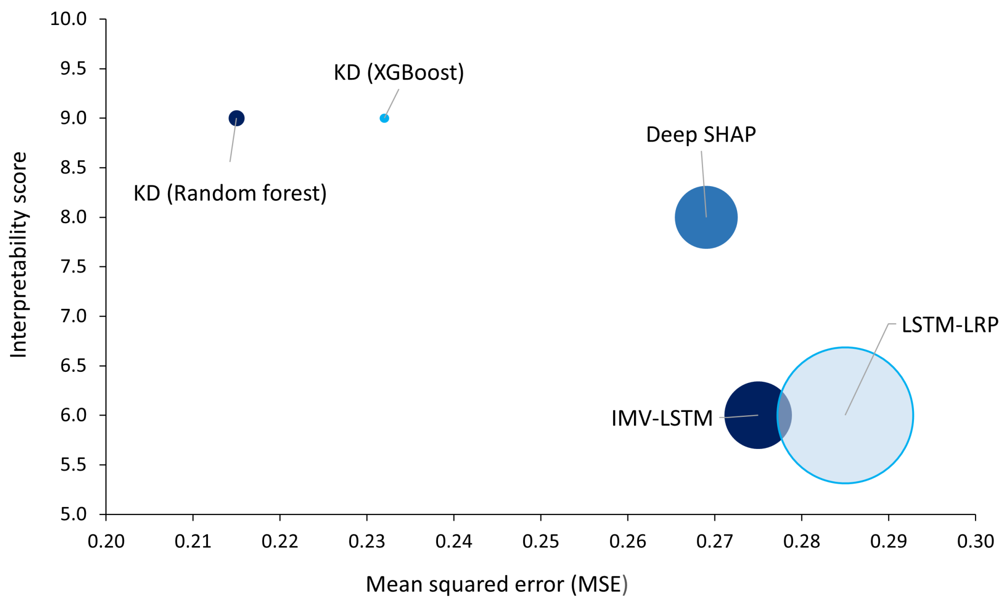
(b) knowledge distillation methods

5.6 Summary of results

In this chapter we have shown that we can achieve reasonable interpretations of the LSTM predictions using various explainability methods. One of the main objectives of this thesis was to recommend one of these methods to Intellibonds. The recommended method will be then implemented and integrated to their spread forecasting platform. As explained in Section 4.5, there are a number of desirable properties that the recommended XAI method should have. Most importantly, it should be fast and able to provide intuitive explanations.

Knowledge distillation based on the XGBoost student model outperforms the other methods reviewed in Chapter 4, as it provides intuitive explanations with a reasonable level of accuracy and at a higher computational speed. Knowledge distillation based on random forest has a higher accuracy than XGBoost, however the difference is marginal and it could be attributed to the selected sample. Figure 5.11 provides a summary of results for the different XAI methods under consideration along the dimensions of interpretability, accuracy and computational time.

Figure 5.11: Comparing XAI methods based on interpretability, accuracy and computational efficiency. The size of the bubbles represents the computational time required to train the model and to obtain explanations.



Note: A higher interpretability score suggests that the method under consideration is able to provide more intuitive explanations. Accuracy is measured in terms of the MSE score in order to be consistent with the loss function used in the training of the original LSTM model. A higher MSE score indicates lower accuracy. The size of the bubbles represents the computational time required to train the model and to obtain explanations.

Chapter 6

Conclusion and Further work

6.1 Conclusion

The scope of this study is to investigate explainable artificial intelligence (XAI) methods that can enhance the interpretability of the LSTM model that is used to generate predictions of spread changes. There are two main research questions in this thesis.

1. Are there any model-agnostic and/or model-specific methods that can effectively interpret the predictions of the LSTM model?

2. Are the identified methods able to generate computationally efficient and intuitive interpretations with a reasonable level of accuracy?

Four explanation methods are considered appropriate to enhance the interpretability of the LSTM model: i) DeepSHAP; ii) knowledge distillation with random forest and XGBoost as student models; iii) IMV-LSTM and iv) LSTM-LRP. It is expected that DeepSHAP would generate the most reliable importance scores as it is directly applied to the predictions of the original LSTM model. Knowledge distillation is expected to be the fastest method as SHAP's tree explainer is well integrated with the random forest and XGBoost models. However, this comes at the expense of accuracy and fidelity. On the other hand, model-specific methods such as LSTM-LRP and IMV-LSTM are expected to perform better in terms of accuracy and fidelity because their network architecture has a closer resemblance to the original LSTM model.

The results presented in Chapter 5 are generally in line with our expectations, except for the LSTM-LRP and IMV-LSTM which were the least accurate models. This could be related to the sample size and the validation set being underrepresented relative to the training set, hence the hyperparameters found may not be the best ones to use for these models. One can overcome this limitation by using a larger sample size and a more representative parameter grid to achieve better hyperparameter optimisation. Models that rely on SHAP-based explanations (i.e. DeepSHAP and knowledge distillation) provide the most intuitive explanations because summing up the individual contributions results in the predicted output and the sign of importance scores shows the directional impact of each feature on the predicted spread changes.

We have completed the research objectives of this study as we were able to show that one can obtain reasonable interpretations of the LSTM model using various explainability methods. Moreover, we conducted a comparative assessment of the XAI methods listed above, based on the framework explained in Section 4.5 and we were able to recommend knowledge distillation based on the XGBoost student model as the best method for implementation in Intellibonds platform. This method outperforms the others, as it provides intuitive explanations with a reasonable level of accuracy and at a higher computational speed.

6.2 Further work

This study currently focuses only on investment-grade corporate bonds. Further investigation is required to analyse whether the XAI methods explored so far are suitable for high-yield corporate bonds. We are currently summarising the interpretability results at the group level, where the importance score for each group is computed as the summation of the importance scores of underlying features. We have aggregated the 58 pre-processed features into 32 groups using our domain knowledge and feedback from Intellibonds. This process can be further refined by employing a clustering technique or a principal component analysis for feature groupings as opposed to their arbitrary assignment. In addition to the features obtained from Intellibonds data, one may consider including other features that are widely recognised in the literature as important spread determinants. For instance, explicitly accounting for bond liquidity, equity market returns at the issuer level, asset volatility, slope of default-free term structure etc. can further increase the predictive accuracy of the original LSTM model and lead to more intuitive explanations for the end-user.

It would also be interesting to employ a clustering algorithm to group similar bonds together [57], and implement different explainability methods for each group based on the criteria described in Section 4.5.

There are several improvements that can be extended to the explainability methods described in Chapter 4. For DeepSHAP, one can consider using a larger sample size or averaging the importance scores for each feature across several small sample iterations. For knowledge distillation, one can try different searching techniques such as: Bayesian optimisation, tree-structured Parzen estimators and population-based training [62], for the hyperparameter tuning of the student models. Moreover, an alternative knowledge distillation technique can be used where a deep neural network is employed as the student model complemented with the DeepSHAP method for model interpretability. For IMV-LSTM and LSTM-LRP one can try using a larger sample size, considering a more representative parameter grid for tuning and/or employing different hyperparameter optimisation techniques to select the best models.

Finally, the original LSTM spread prediction model can also be further improved in order to achieve higher predictive accuracy and more intuitive explanations. For instance, one can try pre-training the weights through random restarts or an alternative weight initialisation method such as Xavier instead of using a random seed. This helps the model achieve faster convergence and more stable training. Moreover, one can consider a wider parameter grid for model tuning and different hyperparameter searching techniques in order to identify the LSTM model with the best out-of-sample predictive accuracy.

Bibliography

[1] Y. LeCun, Y. Bengio, and G. Hinton, “Deep learning,” *Nature*, vol. 521, no. 7553, pp. 436–444, 2015. DOI: 10.1038/nature14539.

[2] G. Cybenko, “Approximation by superpositions of a sigmoidal function,” *Mathematics of Control, Signals and Systems*, vol. 2, no. 4, pp. 303–314, 1989. DOI: 10.1016/j.fcij.2018.10.003.

[3] A. Parmezan, V. M. A. Souza, and G. Batista, “Evaluation of statistical and machine learning models for time series prediction: Identifying the state-of-the-art and the best conditions for the use of each model,” *Neural Networks*, vol. 484, no. 4, pp. 278–293, 2019. DOI: 10.1016/j.ejor.2015.11.027.

[4] S. Namini, N. Tavakoli, and A. Namin, “A comparison of arima and lstm in forecasting time series,” *in Proceedings of the 17th IEEE International Conference on Machine Learning and Applications*, 2018. DOI: 10.1109/ICMLA.2018.00227.

[5] S. Hochreiter and J. Schmidhuber, “Long short-term memory,” *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, 1997. DOI: 10.1162/neco.1997.9.8.1735.

[6] P. E. Utgoff and D. J. Stracuzzi, “Many-layered learning,” *Neural Computation*, vol. 14, no. 10, pp. 2497–2529, 2002. DOI: 10.1162/08997660260293319.

[7] F. Black and M. Scholes, “The pricing of options and corporate liabilities,” *Journal of Political Economy*, vol. 81, no. 3, pp. 637–654, 1973. [Online]. Available: http://www.jstor.org/stable/1831029?origin=JSTOR-pdf.

[8] F. J. Fabozzi, A. K. Bhattacharya, and W. S. Berliner, *Mortgage-Backed Securities: Products, Structuring, and Analytical Techniques*, 2nd. Hoboken, New Jersey: Wiley, 2007.

[9] A. V. Landschoot, “The determinants of credit spreads,” *Financial Stability Review*, vol. 2, no. 1, pp. 135–155, 2004. [Online]. Available: https://ssrn.com/abstract=587264.

[10] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. London: The MIT Press, 2015.

[11] Z. Lipton, “The mythos of model interpretability: In machine learning, the concept of interpretability is both important and slippery,” *Association for Computing Machinery*, vol. 16, no. 3, pp. 31–57, 2018. DOI: 10.1145/3236386.3241340.

[12] S. M. Lundberg and S. I. Lee, “A unified approach to interpreting model predictions,” *Advances in Neural Information Processing Systems 30*, pp. 4765–4774, 2017. arXiv: 1705.07874.

[13] X. Liu, X. Wang, and S. Matwin, “Improving the interpretability of deep neural networks with knowledge distillation,” *in Proceedings of the 2018 IEEE International Conference on Data Mining Workshops*, 2018. DOI: 10.1109/ICDMW.2018.00132.

[14] T. Guo and T. Lin, “Exploring the interpretability of lstm neural networks over multi-variable data,” *in Proceedings of the 36th International Conference on Machine Learning*, vol. 97, pp. 2494–2504, 2019. DOI: 10.1016/j.neucom.2018.01.007.

[15] L. Arras, J. A. Medina, M. Widrich, G. Montavon, M. Gillhofer, K. R. Müller, S. Hochreiter, and W. Samek, “Explaining and interpreting lstms,” *Explainable AI: Interpreting, Explaining and Visualizing Deep Learning*, vol. 97, pp. 211–238, 2019. DOI: 10.1007/978-3-030-28954-6_11.

[16] G. Box, G. Jenkins, G. Reinsel, and G. Ljung, *Time Series Analysis: Forecasting and Control*, 5th. Hoboken, New Jersey: Wiley, 2016.

[17] A. Ariyo, A. Adewumi, and C. Ayo, “Stock price prediction using the arima model,” *in Proceedings of the International Conference on Computer Modelling and Simulation, Accession Number: 14934124*, 2014. DOI: 10.1109/UKSim.2014.67.

[18] A. Pankratz, *Forecasting with Univariate Box-Jenkins Models: Concepts and Cases.* Hoboken, New Jersey: Wiley, 1983.

[19] F. Chollet, *Deep-learning with python*, 7th. Greenwich, United States: Manning Publications, 2017.

[20] G. Zhang, B. E. Patuwo, and M. Y. Hu., “Forecasting with artificial neural networks, the state of art,” *International Journal of Forecasting*, vol. 14, no. 1, pp. 35–62, 1998. DOI: 10.1016/j.fcij.2018.10.003.

[21] K. Hornik, M. Stinchcombe, and H. White, “Multilayer feedforward networks are universal approximators,” *Neural Networks*, vol. 2, no. 5, pp. 359–366, 1989. DOI: 10.1016/0893-6080(89)90020-8.

[22] X. Glorot and Y. Bengio, “Understanding the difficulty of training deep feedforward neural networks,” *in Proceedings of the 13th International Conference on Artificial Intelligence and Statistics*, vol. 9, pp. 249–256, 2010. [Online]. Available: http://proceedings.mlr.press/v9/glorot10a.html.

[23] K. He, X. Zh, S.Ren, and J. Sun, “Delving deep into rectifiers: Surpassing human-level performance on imagenet classification,” *Proceedings of the 2015 IEEE International Conference on Computer Vision (ICCV)*, 2015. DOI: `10.1109/ICCV.2015.123`.

[24] J. B. D. P. Kingma, “Adam: A method for stochastic optimization,” *Proceedings of the 3rd International Conference for Learning Representations*, 2015. arXiv: `1412.6980`.

[25] P. Zhang, E. Patuwo, and M. Y. Hu, “A simulation study of artificial neural networks for nonlinear time-series forecasting,” *Computers & Operations Research*, vol. 28, no. 4, pp. 381–396, 2001. DOI: `10.1016/S0305-0548(99)00123-9`.

[26] A. L. Maas, Q. V. Lee, T. M. O’Neil, O. Vinayls, P. Nguyen, and A. Y. Ng, “Recurrent neural networks for noise reduction in robust asr,” *in Proceedings of the 2012 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2012. DOI: `10.1109/ICASSP.2012.6288816`.

[27] I. Sutskever, J. Martens, and G. Hinton, “Generating text with recurrent neural networks,” *Proceedings of the 28th International Conference on International Conference on Machine Learning*, vol. 57, no. 6, pp. 1017–1024, 2011. DOI: `10.1109/MCOM.2019.1800155`.

[28] R. Pascanu, T. Mikolov, and Y. Bengio, “On the difficulty of training recurrent neural networks,” *Proceedings of the 30th International Conference on International Conference on Machine Learning*, vol. 28, no. 3, pp. 1310–1318, 2013. arXiv: `1211.5063`.

[29] A. Zhang, Z. C. Lipton, M. Li, and A. J. Smola, *Dive into Deep Learning: Tools for Engagement*, 1st. California, USA: Corwin, 2019.

[30] R. M. Schmidt, “Recurrent neural networks: A gentle introduction and overview,” *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, 2019. arXiv: `1912.05911`.

[31] K. Sagheer and M. Kotb, “Time series forecasting of petroleum production using deep lstm recurrent networks,” *Neurocomputing*, vol. 323, no. 5, pp. 203–213, 2019. DOI: `10.1016/j.neucom.2018.09.082`.

[32] S. Namini, N. Tavakoli, and A. Namin, “The performance of lstm and bilstm in forecasting time series,” *in Proceedings of the 2019 IEEE International Conference on Big Data (Big Data)*, 2019. DOI: `10.1109/BigData47090.2019.9005997`.

[33] W. Yuting, Y. Mei, D. Shaopeng, L. Li, and L. Yingqi, “Remaining useful life estimation of engineered systems using vanilla lstm neural networks,” *Neurocomputing*, vol. 275, no. 3, pp. 167–179, 2018. DOI: `doi.org/10.1016/j.neucom.2017.05.063`.

[34] K. Greff, R. K. Srivastava, J. Koutník, B. R. Steunebrink, and J. Schmidhuber, "Lstm: A search space odyssey," *Transactions on Neural Networks and Learning Systems*, vol. 28, no. 10, pp. 2222–2232, 2017. DOI: 10.1109/TNNLS.2016.2582924.

[35] H. Abbasimehr, M. Shabani, and M. Yousefi, "An optimized model using lstm network for demand forecasting," *Computers & Industrial Engineering*, vol. 143, no. 4, pp. 106–435, 2020. DOI: 10.1016/j.cie.2020.106435.

[36] M. Längkvist, L. Karlsson, and A. Loutfi, "A review of unsupervised feature learning and deep learning for time-series modeling," *Pattern Recognition Letters*, vol. 42, no. 1, pp. 11–24, 2014. DOI: 10.1016/j.patrec.2014.01.008.

[37] S. Spiegel, J. Gaebler, A. Lommatzsch, E. D. Luca, and S. Albayrak, "Pattern recognition and classification for multivariate time series," *in Proceedings of the Fifth International Workshop on Knowledge Discovery from Sensor Data*, pp. 34–42, 2011. DOI: 10.1145/2003653.2003657.

[38] C. Krishnan, P. H. Ritchken, and J. B.Thomson, "Predicting credit spreads," *Journal of Financial Intermediation*, vol. 19, no. 4, pp. 529–563, 2010. DOI: 10.1016/j.jfi.2009.02.004.

[39] S. Gu, B. Kelly, and D. Xiu, "Empirical asset pricing via machine learning," *Review of Financial Studies*, vol. 33, no. 5, pp. 2223–2273, 2020. DOI: 10.1093/rfs/hhaa009.

[40] D. Bianchi, M. Büchner, A. Tamoni, and S. V. Nieuwerburgh, "Bond risk premiums with machine learning," *Review of Financial Studies*, vol. 34, no. 2, pp. 1046–1089, 2021. [Online]. Available: https://ideas.repec.org/a/oup/rfinst/v34y2021i2p1046-1089..html.

[41] R. Xiong, H. Cai, I. Diego-Guerra, Y. Lu, X. Xu, and Y. Yin, "Forecasting credit spreads: A machine learning approach," *in Proceedings of the International Association for Quantitative Finance*, 2019. [Online]. Available: https://www.iaqf.org/UMich.

[42] M. Telgarsky, "Benefits of depth in neural networks," *Proceedings of Machine Learning Research*, vol. 49, no. 5, pp. 1517–1539, 2016. arXiv: 1602.04485.

[43] C. Molnar, *Interpretable Machine Learning*, 1st. Hoboken, New Jersey: Wiley, 2019. arXiv: 2010.09337.

[44] T. Miller, "Explanation in artificial intelligence: Insights from the social sciences," *Artificial Intelligence*, vol. 267, no. 5, pp. 1–38, 2017. DOI: 10.1016/j.artint.2018.07.007.

[45] B. Kim, R. Khanna, and O. O. Koyejo, "Examples are not enough, learn to criticize," *Advances in Neural Information Processing Systems 29*, pp. 2288–2296, 2016. DOI: 10.5555/3157096.3157352.

[46] L. H. Gilpin, D. Bau, B. Z. Yuan, A. Bajwa, M. Specter, and L. Kagal, "Explaining explanations: An overview of interpretability of machine learning," *in Proceedings of the 2018 IEEE 5th International Conference on Data Science and Advanced Analytics*, 2018. DOI: 10.1109/DSAA.2018.00018.

[47] M. T. Ribeiro, S. Singh, and C. Guestrin, "Why Should I Trust You?: explaining the predictions of any classifier," *in Proceedings of the 22nd International Conference on Knowledge Discovery and Data Mining*, pp. 1135–1144, 2016. DOI: 10.1145/2939672.2939778.

[48] M. M. Sundararajan, A. Taly, and Q. Yan, "Axiomatic attribution for deep networks," in *in Proceedings of the International Conference on Machine Learning (ICML)*, 2017. arXiv: 1703.01365.

[49] L. Arras, G. Montavon, K. R. Müller, and W. Samek, "Explaining recurrent neural network predictions in sentiment analysis," *in Proceedings of the EMNLP'17 Workshop on Computational Approaches to Subjectivity, Sentiment & Social Media Analysis*, 2017. DOI: 10.18653/v1/W17-5221.

[50] A. Shrikumar, P. Greenside, and A. Kundaje, "Learning important features through propagating activation differences," *in Proceedings of the International Conference on Machine Learning (ICML)*, pp. 3145–3153, 2017. arXiv: 1704.02685.

[51] M. Sundararajan, A. Taly, and Q. Yan, "Gradients of counterfactuals," *in Proceedings of the EMNLP'17 Workshop on Computational Approaches to Subjectivity, Sentiment & Social Media Analysis*, 2016. arXiv: 1611.02639.

[52] M. Ancona, E. Ceolini, and C. Öztireli, "Towards better understanding of gradient-based attribution methods for deep neural networks," *in Proceedings of the 6th International Conference on Learning Representations*, 2018. DOI: 10.3929/ethz-b-000249929.

[53] S. Bach, A. Binder, G. Montavon, F. Klauschen, K. R. Müller, and W. Samek, "On pixel-wise explanations for non-linear classifier decisions by layer-wise relevance propagation," *Public Library of Science*, vol. 10, no. 7, pp. 130–140, 2015. DOI: 10.1371/journal.pone.0130140.

[54] Y. Xu, S. Biswal, S. R. Deshpande, K. O. Maher, and J. Sun, "RAIM: Recurrent attentive and intensive model of multimodal patient monitoring data," *in Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 2565–2573, 2018. DOI: 10.1145/3219819.3220051.

[55] H. Choi, K. Cho, and Y. Bengio, "Fine-grained attention mechanism for neural machine translation," *Neurocomputing*, vol. 284, no. 6, pp. 171–176, 2018. DOI: 10.1016/j.neucom.2018.01.007.

[56] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *in Proceedings of the 3rd International Conference for Learning Representations*, vol. 13, no. 10, 281–305, 2017. arXiv: 1008.2849v1.

[57] T. Hastie, R. Ribshirani, and J. Friedman, *The Elements of Statistical Learning*, 2nd ed. Springer, 2008.

[58] G. James, D. Witten, T. Hastie, and R. Tibshirani, *An Introduction to Statistical Learning*. Springer, 2013, pp. 316–321.

[59] T. K. Ho, "Random decision forests," in *Proceedings of the 3rd International Conference on Document Analysis and Recognition*, vol. 1, 1995, pp. 278–282. DOI: 10.1109/ICDAR.1995.598994.

[60] T. Chen and C. Guestrin, "Xgboost: A scalable tree boosting system," *in Proceedings of the 22nd ACM International Conference on Knowledge Discovery and Data Mining*, vol. 16, pp. 785–794, 2016. DOI: 10.1145/2939672.2939785.

[61] S. M. Lundberg, G. G. Erion, and S. I. Lee, "Consistent individualized feature attribution for tree ensembles," *Computing Research Repository*, 2018. arXiv: 1802.03888.

[62] J. Bergstra and Y. Bengio, "Random search for hyper-parameter optimization," *Journal of Machine Learning*, vol. 13, no. 10, 281–305, 2012. DOI: 10.5555/2188385.2188395.

[63] L. Zhang, C. Agrawal, and G. Qi, "Stock price prediction via discovering multi-frequency trading patterns," *in Proceedings of the 23rd International Conference on Knowledge Discovery and Data Mining*, pp. 2141–2149, 2017. DOI: 10.1145/3097983.3098117.

[64] D. Bahdanau, K. Cho, and Y. Bengio, *Neural machine translation by jointly learning to align and translate*, 2016. arXiv: 1409.0473.

[65] A. Warnecke, D. Arp, C. Wressnegger, and K. Rieck, *Evaluating Explanation Methods for Deep Learning in Security*, 2020. arXiv: 1906.02108.

[66] Y. Hong, H. Lin, and C. Wu, "Are corporate bond market returns predictable?" *Journal of Banking & Finance*, vol. 36, no. 8, pp. 2216–2232, 2012. DOI: 10.1016/j.jbankfin.2012.04.001.

[67] M. Khashei and M. Bijari, "A novel hybridization of artificial neural networks and arima models for time series forecasting," *Applied Soft Computing*, vol. 11, no. 2, pp. 2664–2675, 2011. DOI: 10.1016/j.asoc.2010.10.015.

[68] S. Haykin, *Neural networks and learning machines*, 3rd. London: Pearson, 2009.

[69] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, “Learning representations by back-propagating errors,” *Nature*, vol. 323, no. 6088, pp. 533–536, 1986. DOI: 10.1038/323533a0.

[70] O. Sezer, M. Gudelek, and A. Ozbayoglu, “Financial time series forecasting with deep learning : A systematic literature review: 2005-2019,” *Applied Soft Computing*, vol. 90, no. 13288, pp. 106–181, 2020. DOI: 10.1016/j.asoc.2020.106181.

[71] Y. Hua, Z. Zhao, R. Li, X. Chen, Z. Liu, and H. Zhang, “Deep learning with long short-term memory for time series prediction,” *in Proceedings of the IEEE Communications Magazine*, vol. 57, no. 6, 2019. DOI: 10.1109/MCOM.2019.1800155.

[72] J. Donahue, L. A. Hendricks, and S. Guadarrama, “Long-term recurrent convolutional networks for visual recognition and description,” *in Proceedings of the 2015 IEEE Conference on Computer Vision and Pattern Recognition*, 2015. DOI: 10.1109/CVPR.2015.7298878.

[73] M. Choudhry, *Analysing and Interpreting the Yield Curve*, 2nd. Hoboken, New Jersey: Wiley, 2019.

[74] M.Choudhry, “Revisiting the credit default swap basis: Further analysis of the cash and synthetic credit market differential,” *The Journal of Structured Finance*, vol. 11, no. 4, pp. 21–32, 2006. DOI: 10.3905/jsf.2006.614079.

[75] R. Merton, “On the pricing of corporate debt : The risk structure of interest rates,” *Journal of Political Economy*, vol. 29, no. 2, pp. 449–470, 1974. DOI: 10.1111/j.1540-6261.1974.tb03058.x.

[76] M. Boss and M. Scheicher, “The determinants of credit spread changes in the euro area,” *Market functioning and central bank policy*, vol. 12, no. 3, pp. 181–199, 2002. [Online]. Available: https://www.bis.org/publ/bppdf/bispap12j.pdf.

Chapter 7

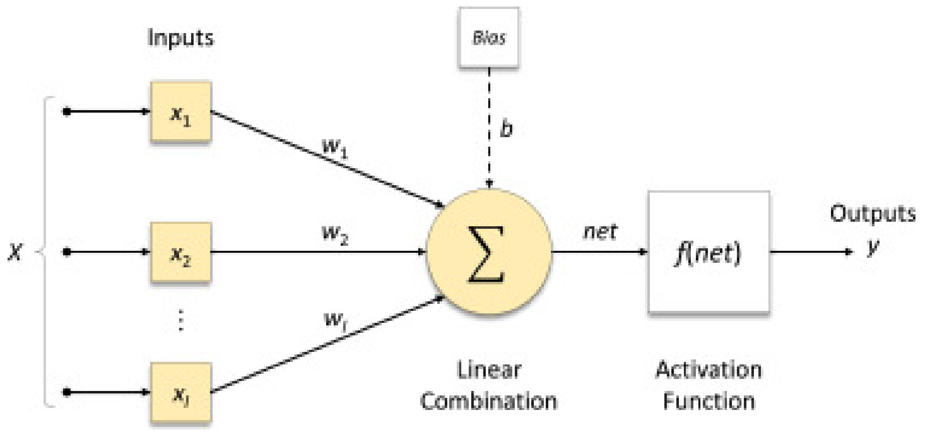
Appendix

7.1 A1. Artificial Neural Networks (ANNs)

ANNs are computational non-parametric models that capture complex nonlinear relationships between the response variable and its predictors. [67].

The simplest networks contain no hidden layers and are equivalent to linear regressions. The network depicted in Figure 7.1 is the simplest form of the ANN. It is called a perceptron. A perceptron is a simple network used for binary classifications and it does not have any hidden layers. The coefficients attached to the predictors are called "weights". The forecasts are obtained as a linear combination of the inputs [35].

Figure 7.1: The structure of a perceptron [3].



The single neuron takes in an input vector $\mathbf{x} = (x_1, x_2, ...x_m)^\intercal$, where m is the number of inputs. Each element $x_i \in \mathbf{x}$ where $i = 1, 2, .., m$, is associated with a weight w_i. This weight can assume a negative or positive value that reflects the importance of the input i. The value of the perceptron, also referred to as the "net value" in the literature, is computed by performing a linear combination of the weighted inputs and adding a bias term $b \in \mathbb{R}$ [3]:

$$z = \sum_{i=1}^{m} w_i x_i + b. \tag{7.1}$$

This net value z, is used as an input value to an activation function $f(z)$ that computes the output y of the neuron. For a single layer perceptron the activation function that was originally used was the Heaviside step function $H(z)$ [68]:

$$H(z) = \begin{cases} 1 & \text{if } \ z > 0, \\ 0 & \text{otherwise.} \end{cases} \tag{7.2}$$

The weights and the bias term are the parameters of the perceptron, and they are adjusted by performing a learning process with a finite number of iterations. The learning is conducted via the perceptron convergence algorithm which searches for a weight vector $\mathbf{w}$ and a bias term b, such that the two equalities of the Heaviside step function defined in Equation (7.2) are satisfied [68].

Once we add an intermediate layer with hidden neurons, the neural network becomes non-linear. The emergence of the backpropagation learning algorithm enabled the popularity of ANNs with more than two layers. One type of ANN with multiple layers, is the multilayer perceptrons (MLP). An MLP is composed of at least three layers of nodes: an input layer, a hidden layer and an output layer [19].

MLPs learning process is implemented through backpropagation, which is essentially a practical application of the chain rule for derivatives [69].

Figure 7.2 below, shows the structure of a three-layered MLP.

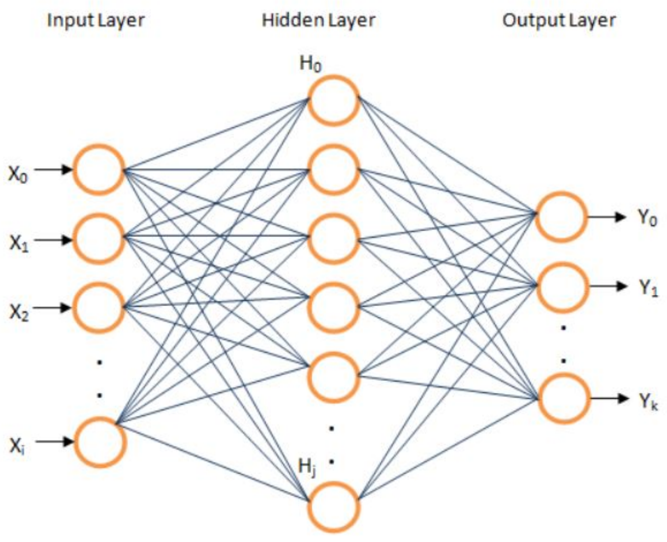
The output of the model in Figure 7.2 assuming a single neuron in the output layer is [3]:

$$y = g\left(\sum_{j=1}^{n} w_j \mathbf{f}(\sum_{k=1}^{m} w_{kj} x_k + b_j) + b \right) \tag{7.3}$$

where $\mathbf{f}(\cdot)$ and $g(\cdot)$ are activation functions of the input layer and the hidden layer respectively, w_{kj} is the weight associated with input k and neuron j in the hidden layer, w_j is the weight associated with neuron j in the hidden layer, b_j is the bias term associated with neuron j in the hidden layer and b is the bias term of the neuron in the output layer.

The advent of new activation functions has enabled a more efficient learning for MLPs, because they have addressed a major limitation of the backprogpagation process when us-

Figure 7.2: The structure of a three-layered MLP [69].



ing the Heaviside function, related to its non-differentiability at $x = 0$ and the derivative being 0 elsewhere [19].

The most commonly used nonlinear activation functions in MLPs are [70]:

- sigmoid function: $\sigma(z) = \frac{1}{1+e^{-z}}$;
- hyperbolic tangent: $\tanh(z) = \frac{e^{z}-e^{-z}}{e^{z}+e^{-z}}$;
- rectified linear unit (ReLU) also known as the ramp function: $\mathrm{R}(z)=\max(0, z)$;
- softmax: $\mathrm{S}(z) = \frac{e^{z_i}}{\sum_{j=1}^{N} e^{z_j}}$ for $i = 1, 2..., N$ and $\mathbf{z} = (z_1, z_2, ..., z_N)^\intercal$. The element z_i represents the net value computed by neuron i of the previous layer.

A common characteristic of MLP models is that they are memory-less, and they use the feed forward architecture, in which the neuron-to-neuron signals flow only in one direction: from input to output. This is a limitation when handling sequential data with temporal dependencies [25].

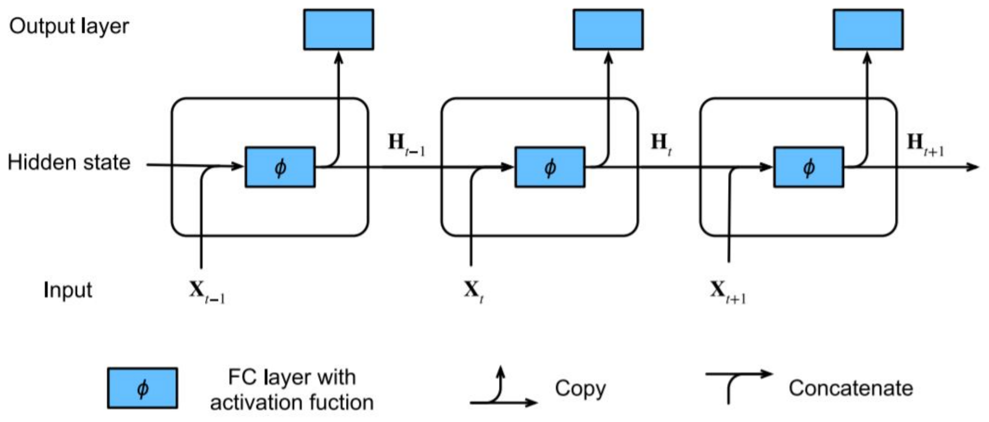
For the purpose of time series modelling, we refer to another class of deep-learning models called RNNs, which have the ability to store previous inputs and leverage sequential information [3].

7.2 A2. Recurrent neural networks (RNNs)

RNNs have cycles that help them store information from previous inputs. This special memory is called recurrent hidden states and gives RNNs the ability to predict what input is coming next in the sequence of input data [1].

An illustration of a RNN unit unfolded in time is depicted in Figure 7.3:

Figure 7.3: The architecture of RNNs unfolded in time [71].



We can describe this process of passing information in the RNN units using the mathematical notation proposed in Zhang et al.[29]. We denote the hidden state and the input at time step t, as $\mathbf{H}_t \in \mathbb{R}^n \times \mathbb{R}^u$ and $\mathbf{X}_t \in \mathbb{R}^n \times \mathbb{R}^m$ respectively, where n is the number of samples, m is the number of input variables and u is the number of hidden units. In other words, each row i of the matrix $\mathbf{X}_t$ and $\mathbf{H}_t$ for $i = 1, 2, ...n$, corresponds to an input vector $\mathbf{x}_t \in \mathbb{R}^m$ and a hidden states vector $\mathbf{h}_t \in \mathbb{R}^u$ associated with instance i.

Further, we use an input weight matrix $\mathbf{W}_{\text{xh}} \in \mathbb{R}^u \times \mathbb{R}^m$, a hidden-to-hidden state matrix $\mathbf{W}_{\text{hh}} \in \mathbb{R}^u \times \mathbb{R}^u$, an output weight matrix $\mathbf{W}_{\text{ho}} \in \mathbb{R}^o \times \mathbb{R}^u$ where o is the number of units in the output layer, and bias parameters $\mathbf{b}_{\text{h}} \in \mathbb{R}^u$ and $\mathbf{b}_{\text{o}} \in \mathbb{R}^o$ for the hidden layer and the output layer respectively.

RNNs transfer the information gathered by the current input and the previous hidden state to an activation function ϕ which is usually a logistic sigmoid or tanh function, in order to update the hidden variable $\mathbf{h_t}$ and to obtain the output variable $\mathbf{o_t}$:

$$\mathbf{h}_t = \phi_{\text{h}}(\mathbf{W}_{\text{xh}}\mathbf{x}_t + \mathbf{W}_{\text{hh}}\mathbf{h}_{t-1} + \mathbf{b}_{\text{h}}). \tag{7.4}$$

The hidden variables $\mathbf{h}_t$ and $\mathbf{h}_{t-1}$ of two adjacent time steps, capture and retain the historical information of the sequence up to their current time step. The hidden state is updated recurrently for each time step. Hence, neural networks with hidden states based on recurrent computation are named recurrent neural networks [29].

For time step t, the output of the final layer is computed below:

$$\mathbf{o}_t = \phi_{\text{o}}(\mathbf{W}_{\text{ho}}\mathbf{h}_t + \mathbf{b}_{\text{o}}). \tag{7.5}$$

Though RNNs have proven successful on tasks such as speech recognition [26], and text generation [27], it can be difficult to train them on long term sequences. This limitation is associated with the the vanishing and exploding gradient problem, which results from backpropagating the gradients through multiple layers of the network, each corresponding to a particular time step [5]. To grasp the vanishing gradient problem it is important to understand how an RNN network learns its parameters through a process called back-propagation through time (BPTT) [29].

When we forward pass our input $\mathbf{x}_t$ through the network we compute the hidden state $\mathbf{h}_t$ and the output state $\mathbf{o}_t$ sequentially (i.e., one step at a time). The difference between models' output $\mathbf{o}_t$ and the target label y_t at time step t, is then evaluated by an objective function across all the discrete T time steps:

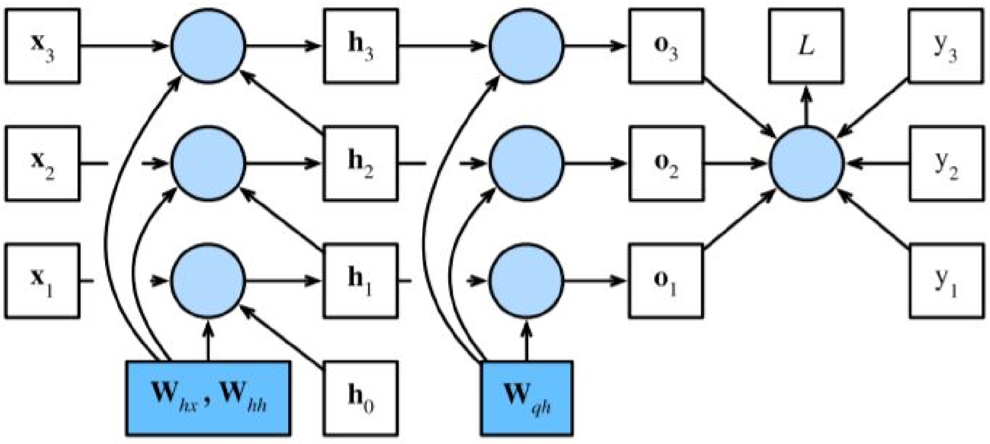
$$\mathcal{L} = \sum_{t=1}^{T} \ell_t(\mathbf{o}_t, y_t), \tag{7.6}$$

where $\ell(\mathbf{o}_t, y_t)$ denotes the loss at time step t, which can be a specific function such as: mean squared error, hinge loss, cross entropy loss, etc. [30]. The objective function that we want to minimise is $\mathcal{L}$ which represents the loss over T time steps, from the beginning of the sequence $t = 1$ [5], [29]. In order to do this, we use BPTT which calculates the gradients of the objective function with respect to model's parameters across different layers, one step at a time [5].

In order to visualize the dependencies in the computation of BPTT, we can draw a computational graph for the RNN model, as shown in Figure 7.4:
Based on the dependencies in Figure 7.4, we can traverse in the opposite direction of the arrows to calculate and store the gradients. Applying the chain rule recursively in the BPPT process, introduces matrix multiplications over long sequences which results in potentially large powers of the hidden-to-hidden state matrix $\mathbf{W}_{\text{hh}}^{T}$. This may generate eigenvalues smaller than 1, and eigenvalues larger than 1, which become numerically unstable for large matrix powers. This causes the vanishing and exploding gradients for long sequences in RNNs. See Zhang et al. for further details [29].

The vanishing/exploding gradient problem, causes the parameters' learning process to stop, thus posing a major limitation in using RNNs for modelling long-term dependencies

Figure 7.4: Backpropagation through time in RNNs [29].



[5]. This problem motivated the introduction of LSTMs, which employ more complex structures called "gates", that can process information and model long-term dependencies more effectively. In the next section we will describe in detail the architecture of LSTMs [72].

7.3 A3. Fundamental concepts in bond pricing

Bonds are debt securities issued by public or private entities to investors, in order to raise capital. The bond issuer is obliged to pay to the holder interest (i.e. the coupon) in consecutive time intervals of generally six months. Bonds instruments have a maturity date in which the principal or the face value is paid back to the bond owner. The coupon rate is represented as a percentage of the principal and the bond value is mathematically estimated as the discounted value of the periodic coupon payments and the principal payment at maturity [73].

The discount rate that enables the sum of all future cash flows to be equal to the current price of the bond is called yield-to-maturity (YTM) and it is calculated as:

$$P = \sum_{i=1}^{Nm} \frac{\frac{c}{m}}{(1+\frac{r}{m})^i} + \frac{M}{(1+\frac{r}{m})^{Nm}}, \tag{7.7}$$

where P is the market price of the bond, c is the annual coupon payment, m is the frequency of coupon payments, r is the discount rate, N is the number of years to maturity and M is the face value or the principal payment. For more details see M. Choudhry [73].

Bond spreads

The conventional approach in calculating the spread has been to use YTM. In this case, the spread would represent the difference between a corporate bond and a government bond with the same maturity measured at a particular point of the Treasury spot rate curve. This has the limitation of fixing the spread calculation to a particular point of the yield curve [74].

Therefore, market practitioners prefer to use the z-spread instead, which is the value that has to be added added to the implied spot yield curve such that the discounted cash flows of the bond are equal to its present value which should in turn equal its current market price [74]

$$P = \sum_{i=1}^{Nm} \frac{\frac{c}{m}}{(1 + \frac{(s_i + z)}{m})^i} + \frac{M}{(1 + \frac{(s_i + z)}{m})^{Nm}} = MV, \tag{7.8}$$

where c is the annual coupon payment, m is the frequency of coupon payments, z is the Z-spread, s_i is the yield on the Treasury security, N is the number of years to maturity, M is the principal at maturity, P is the theoretical price of the bond and MV is the market value of the bond [73]. The z-spread is calculated iteratively based on Equation (7.8).

7.4 A4. Determinants of spread changes

In this section we will first identify the determinants of credit spread implied by early structural credit risk models, developed by Black and Scholes [7] and Merton [75]. However, numerous empirical studies have shown that there is a gap between observed credit spreads and theoretical spreads [9]. Therefore, we will further rely on these studies to identify additional determinants of bonds' spreads that are not captured by structural models.

The early structural models state that the main determinants of credit spreads are financial leverage, asset volatility and the the term structure of risk-free interest rates [38]. In particular the following relationships are identified:

- **Risk-free rate.** We expect a negative relation between the nominal risk-free rate and the credit spread. In structural models, the drift of the risk-neutral process of the value of the assets which is the expected growth of the firms' value, is assumed equal to the risk-free rate [75]. It is shown that an increase in the risk-free rate results in higher firm value and lower credit spread.

- **Slope of default-free term structure.** An increase in the slope implies an increase in the expected short-term interest rates, which in turn is expected to lower a firms' credit spread due to its reduced default risk [9].
- **Asset price.** An increase in the firm's asset value V (for a given debt value), reduces the leverage ratio and consequently the risk to default thus resulting in a lower credit spread [9].
- **Asset volatility.** High asset volatility leads to a high probability that the firm's asset value will fall below the value of its debt [9].

Several studies have empirically investigated the significance of credit spread determinants implied by structural models. These studies have also identified additional factors that may explain the difference between the observed spreads and theoretical spreads. We are summarising the key factors below.

- **Liquidity.** Investors require a premium for investing in less liquid assets. Van Landschoot(2004) [9] find that liquidity effect becomes stronger for lower rated bonds.
- **Taxation.** is another factor that is not accounted by structural models. If taxation differences exist between corporate and government bonds, bond yield spreads are likely to reflect these differences indicating a positive relationship between taxation and credit spreads [9].
- **Credit rating.** Van Landschoot [9] show that credit spreads increase as the creditworthiness of the issuer decreases. Furthermore, credit spread volatility is higher for bonds with lower ratings.
- **Maturity.** Van Landschoot [9] and Boss and Scheicher [76], provide evidence that credit spreads are higher for bonds with longer maturities.
- **Market volatility.** Boss and Scheicher [76], find that there is a significant positive relation between credit spread changes and changes in the volatility of the market. It is observed that the effect of the volatility is asymmetric, i.e. positive changes in the volatility have a much larger impact than negative changes.

The above studies have confirmed the merits of structural models as mechanisms for understanding credit spreads. However,for the most part, these models do not address spreads' predictability which is our main concern. In the next section we will look into recent machine learning models that have proved to be highly accurate in predicting spread changes [9], [38].

7.5 A5. Original input features

In Table 7.1 we provide a comprehensive description of the original features sourced by IHS Markit.

Table 7.1: Original features with descriptions from IHS Markit.

Original Feature	**Description from IHS Markit**
Date	The date of the price.
ISIN	The ISIN (International Securities Identification Number) to which the price data applies.
Ticker	The Markit ticker for the organization.
Issue_Currency	Instrument currency.
Tier	The code representing the bond seniority type.
PV01	The present variation per one basis point change in yield, based on the mid price.
Mid_YTM	The yield to maturity of the mid price of the instrument.
iSpread_Mid	The I spread of the bond, based on the mid price. I spread is the difference between the yield to maturity of the bond and the linearly interpolated yield to the same maturity on an appropriate reference curve.
Mid_Convexity	Measures the change of duration with the change in yield, based on the mid price.
Mid_Z_Spread	The mid Z spread of the instrument, based on the mid price. Z spread is the constant spread that will make the price of a security equal to the present value of its cash flows, when added to the yield at each point on the spot rate treasury curve, where a cash flow is received.
Mid_OAS	The Option Adjusted Spread (OAS), based on the mid price. OAS is the flat spread which has to be added to the treasury yield curve in a pricing model (that accounts for embedded options) to discount a security payment to match its market price.
Mid_Effective_Duration	A duration calculation for bonds with embedded options, based on the mid price. Effective duration takes into account that expected cash flows fluctuate as interest rates change.
Dirty_Mid_Price	The price of the bond plus the interest that accrued between coupon payments, based on the mid price.
Accrued_Interest	The interest that is owed, but not yet paid, added to the price of the bond.

Table 7.1: Original features with descriptions from IHS Markit.

Original Feature	**Description from IHS Markit**
Issue_Size	
Spread_vs_Benchmark_Mid	The spread of the instrument bid yield against the assigned benchmark bid yield.
Country	Country of the coupon currency.
Level_3	These field display the Rules Based Point fuel ranked evel 3 based on latest Final Data Quality Score derived at the ISIN level.
Level_5	These field display the Rules Based Point fuel ranked level 5 based on latest Final Data Quality Score derived at the ISIN level.
ISSR_RATG_TXT	Rating at the ISIN level.
Maturity	The date on which the principal amount of the security becomes due and payable, as stated in the terms of the security.