Imperial College London

Department of Mathematics

European Government Bond Volume Prediction Using Dealer to Client Flow

Author: Kian Hatamieh(CID:01340679)

A thesis submitted for the degree of

MSc in Mathematics and Finance, 2020-2021

Declaration

The work contained in this thesis is my own work unless otherwise stated.

Acknowledgements

I would like to begin by thanking the EGBs e-trading team at Deutsche Bank for their consistant guidance, support and patience throughout the writing of this thesis. Particularly, thank you Sahil Chadha, Simon Wivell and Lubomir Schmidt for all your insight into the European government bond market.

Additionally, I would like to thank Dr Paul Bilokon for all his comments and suggestions in writing this thesis, his help was greatly appreciated.

To my friends George and Hugo, thank you for being able to distract me whenever a break from work was deemed necessary.

Finally, thank you to my parents for all their emotional and financial support as I close out 18 continuous years in education. I cannot imagine where I would be without you both, and I am grateful everyday for all you have done for me.

Abstract

This thesis explores how one can use machine learning to predict the volumes of certain European government bonds to be traded over long term (daily) horizons and short term horizons (hourly, 15 and 5 minute buckets). Our initial regression models contain features relating to previous volumes, special days in the bond calendar and bond pricing, while our most complicated model, a multilayer perceptron neural network, contains over 300 weights to be calibrated.

We show that both linear and non-linear models can be utilised successfully, resulting in them performing better out-of-sample than a benchmark of a moving average of previously observed volumes. Furthermore, we devise a feature selection algorithm in order to determine the key drivers behind volume prediction, as well as comparing the most useful features when modelling a long term horizon opposed to a short term horizon.

Contents

List of Figures

List of Tables

Introduction

0.1 Background

Volume forecasting for European government bonds is a topic which both traditional machine learning techniques, and more advanced ones can be applied towards. The focus of this thesis is on D2C markets (dealer to client). In these markets, various buy side institutions, such as hedge funds or pension funds, approach market makers with a 'request for quotation' (RFQ). RFQs were first introducted by Tradeweb in 1998 (Tradeweb), and serve as a platform to connect buyers and sellers in the market. An RFQ requests a price for a bid/ask of a certain ISIN (a specific bond). Buy side institutions will send multiple RFQs to different market makers, such as Deutsche Bank, and will then pick who to enter a transaction with based upon who provides the best price. Offering a competitive price, but not one that is too competitive is key, as one aims to win a trade without providing too good a price which would mean they would not be able to buy/sell on a contract for a profit. Initially we will provide a static prediction for the amount of bonds traded via RFQs for a given day, however further on in this thesis we will develop a model to provide hourly intraday updates, based upon realised volumes in D2C markets.

Bonds have tenors attached to them, indicating how long they are from maturity. The market for bonds of a 2 year maturity and the market for bonds of a 30 year maturity are vastly different, due to the respective yields associated with the bonds. As of the publication of this thesis, there is over a 100 basis point spread between the yields of Italian 2 year and 10 year bonds, and a 35 basis point spread between the yields of German 2 year and 10 year bonds (MTS). Considering this, this thesis' primary focus is being able to predict daily (long horizon) volumes for European government bond 'sectors'. A sector is defined as a combination of both bond issue country, and tenor bucket, and the sector we carry out our initial analysis on is German 7-11 year bonds. In the most simplistic model of a simple moving average, we see limited success, since while previously traded volumes can be indicative of future behaviours, there are many other features that go into volume prediction. In an increasingly competitive environment, more advanced techniques are required to study the behaviour of volumes. Whilst we will begin with examining more simplistic topics, such as feature engineering, cross-validation and OLS/Ridge/LASSO regression, by the end of this thesis we will have examined the utility provided by techniques such as random forests, multilayer perceptron neural networks as well as clustering via PCA to examine the inter-dependencies between sectors.

Additionally, via further feature engineering, we will discuss a model that uses our initial daily prediction in order to predict volumes every hour, 15 minutes and even 5 minutes. Time series modelling can be difficult, due to features such as non-stationary and temporality, and thus we explore techniques such as data normalisation and different cross-validation/train-test split techniques to accommodate for this. For example, a typical k-fold cross validation technique, which can be effective on non-time series data, cannot be used, since one would be training a model on data in the future to predict results now.

To summarise, this thesis will be split up primarily into three sections. In our initial section, we explore both typical regression based models for modelling a singular sector, German 7-11 year bonds, as well as non-linear more advanced machine learning models. We will then extend our models to explore the dependencies found within various sectors, and report the robustness of our models by seeing if they can accurately predict sectors which do not share many common attributes. Finally, we delve into intraday modelling, to see how our models fare with a data set with much greater variance.

0.2 Methodology and motivation

Throughout this thesis, we will examine the various factors and features associated with European government bonds trading volumes, ranging from the aforementioned obvious features such as the volume traded in the past, to more nuanced and specific features such as European Central Bank meetings. One potential motivation for studying volume predictions is as follows. Market making for bonds can be a very competitive environment, and knowledge of volume traded could be used for the purposes of appropriately hedging as well as allowing a trader to understand how long they may have to hold onto a position for. If one could theoretically perfectly predict future volumes, they could execute 'perfect' bond to bond hedges, meaning they would not have to enter the futures market to execute a hedge. Furthermore, holding on to a contract for a long period of time equates to a potential loss in earnings, as funds cannot be released for more trades to be made.

0.3 Data set

The data set used in this project will be all RFQs received by Deutsche Bank, between January 1st 2019 to 18th May 2021. This 2.5 year time period will ideally be sufficient to pick up any seasonal trends (for example, volumes dropping during December), as well as allowing us to explore the performance of models during more tumultuous periods, such as in March 2020. In March 2020, due to great uncertainty surrounding the COVID-19 pandemic, bond yields behaved erratically. A paper by Paule-Vianez et al. (14), studied the correlation between search terms via Google trends as a proxy for COVID-induced fear, and established that a 1 point increase in COVID-induced fear was associated with an increase in the weekly change in sovereign bond yields of around 0.0007%. Additionally, many spreads widened during this period. It is interesting to include a period such as this as a further test of the robustness of our model.

0.4 Limitations

It is worthwhile examining the limitations in this thesis, and potential expansions that could be covered in further research. Firstly, this thesis will focus on linear regression techniques, as well as non-linear methods such as random forests and neural networks. Clustering algorithms such as k-means and t-SNE are not explored in this thesis, only their more simpler predecessor PCA is. Additionally, we do not provide individual ISIN (bond) level predictions, since this data can be extremely noisy and also many ISINs have missing data due to being new issues. As previously mentioned, our data set contains all RFQs recieved by Deutsche Bank (regardless of whether the trade was won/lost), however we do not include data for RFQs sent to other financial institutions since this is private data. Finally, holidays are removed from the data set, as they often have zero, or very low, volume traded.

Chapter 1

Models : linear & non-linear

1.1 Bias-variance trade-off

Before we begin exploring the individual models, it is worthwhile taking a detour to explain the nuances behind the bias-variance trade-off found in predictive modelling. For a single day, let us assume our model takes p variables as inputs, thus belonging to $\mathbb{R}^p$ and our output (daily prediction) is a value belonging to $\mathbb{R}$. For a given day, we denote an observation as $(\boldsymbol{x}_n, y_n) \in \mathbb{R}^{p+1}$, these being observed values from some unknown joint distribution (X_t, Y_t). We assume some relationship between our Y_t and X_t of the form

$$Y_t = f(X_t) + \varepsilon_t \quad t = 1, 2, \ldots N$$

where $\mathbb{E}(\varepsilon_t) = 0$, $\text{Var}(\varepsilon_t) = \sigma^2$. We take $\mathcal{L} = \{(\boldsymbol{x}_1, y_1) \ldots (\boldsymbol{x}_n, y_n)\}$ as the set of all observed values from the joint distribution of (X_t, Y_t). We aim to find a function $f_{\mathcal{L}}$ to minimise our expected prediction error (with a loss function based on squared loss). The expected prediction error, conditional on $X_t = \boldsymbol{x_t}$ is,

$$EPE[f_{\mathcal{L}}] = \mathbb{E}\Big[\big(Y_t - f_{\mathcal{L}}(\boldsymbol{x}_t)\big)^2 | X_t = \boldsymbol{x}_t\Big]$$

If our function $f_{\mathcal{L}}$ is too far from the true value of f we may end up with a large error, known as estimation bias. However, if $f_{\mathcal{L}}$ varies wildly amongst different training sets, we construct very different $f_{\mathcal{L}}$ for each of these sets. This is known as estimation variance. We can decompose our expected prediction error in terms of these, using that $Y_t = f(X_t) + \varepsilon_t$, yielding,

$$\begin{aligned} EPE[f_{\mathcal{L}}] &= \mathbb{E}\Big[\big(Y_t - f_{\mathcal{L}}(\boldsymbol{x}_t)\big)^2 | X_t = \boldsymbol{x}_t\Big] \\ &= \mathbb{E}\Big[\big(f(\boldsymbol{x}_t) + \varepsilon_t - f_{\mathcal{L}}(\boldsymbol{x}_t)\big)^2 | X_t = \boldsymbol{x}_t\Big] \\ &= \sigma_t^2 + \mathbb{E}\Big[\big(f_{\mathcal{L}}(\boldsymbol{x}_t) - f(\boldsymbol{x}_t)\big)^2\Big] \end{aligned}$$

Here the σ^2 represents irreducible error, incurred due to noise, that we cannot model. We can however further decompose the 2nd term as follows,

$$\begin{aligned} \mathbb{E}\Big[\big(f_{\mathcal{L}}(\boldsymbol{x}_t) - f(\boldsymbol{x}_t)\big)^2\Big] &= \mathbb{E}\Big[\big(f_{\mathcal{L}}(\boldsymbol{x}_t) - \mathbb{E}[(f_{\mathcal{L}}(\boldsymbol{x}_t)] + \mathbb{E}[(f_{\mathcal{L}}(\boldsymbol{x}_t)] - f(\boldsymbol{x}_t)\big)^2\Big] \\ &= \mathbb{E}\Big[\big(f_{\mathcal{L}}(\boldsymbol{x}_t) - \mathbb{E}[f_{\mathcal{L}}(\boldsymbol{x}_t)]\big)^2\Big] + 2\mathbb{E}\Big[\big(f_{\mathcal{L}}(\boldsymbol{x}_t) - \mathbb{E}[f_{\mathcal{L}}(\boldsymbol{x}_t)]\big)\big(\mathbb{E}[f_{\mathcal{L}}(\boldsymbol{x}_t)] - f(\boldsymbol{x}_t)\big)\Big] \\ &+ \mathbb{E}\Big[\big(\mathbb{E}[f_{\mathcal{L}}(\boldsymbol{x}_t)] - f(\boldsymbol{x}_t)\big)^2\Big] \end{aligned}$$

Now note that the cross term $\mathbb{E}\Big[\big(f_{\mathcal{L}}(\boldsymbol{x}_t) - \mathbb{E}[f_{\mathcal{L}}(\boldsymbol{x}_t)]\big)\big(\mathbb{E}[f_{\mathcal{L}}(\boldsymbol{x}_t)] - f(\boldsymbol{x}_t)\big)\Big] = 0$ by linearity of expectation, and additionally we can remove the outside expectation from the third term since

$\mathbb{E}[f_{\mathcal{L}}(\boldsymbol{x}_t)] - f(\boldsymbol{x}_t)$ is a constant. This leaves us with,

$$\begin{aligned} &= \mathbb{E}\Big[\big(f_{\mathcal{L}}(\boldsymbol{x}_t) - \mathbb{E}[f_{\mathcal{L}}(\boldsymbol{x}_t)]\big)^2\Big] + \big(\mathbb{E}[f_{\mathcal{L}}(\boldsymbol{x}_t)] - f(\boldsymbol{x}_t)\big)^2 \\ &= \mathrm{Var}[f_{\mathcal{L}}(\boldsymbol{x}_t)] + \mathrm{Bias}[f_{\mathcal{L}}(\boldsymbol{x}_t)]^2 \end{aligned}$$

Thus our final expression for the expected prediction error is,

$$EPE[f_{\mathcal{L}}] = \sigma^2 + \mathrm{Var}[f_{\mathcal{L}}(\boldsymbol{x}_t)] + \mathrm{Bias}[f_{\mathcal{L}}(\boldsymbol{x}_t)]^2$$

Here we have our bias variance trade off, and these two forces normally works against one another. When we construct a model, typically adding more features will increase complexity, resulting in our bias being decreased, however our variance increases. In more typical machine learning terms, too few features and we may *underfit* the model to our data, yet too many features and we may *overfit* our model to the data, resulting in poor out-of-sample performance. It is important to strike a balance between these two in order to minimise our expected prediction error.

We can now explore our individual models. We begin by covering more traditional models, being ordinary least squares (OLS), ridge and LASSO, before covering more advanced machine learning models (random forest, multilayer perceptron (MLP)). Regarding our linear models, we assume some sort of linear relationship between our target variable, $y \in \mathbb{R}$, and a set of regressors/features, $(x_1, x_2, \dots x_p) \in \mathbb{R}^p$. The linear relationship is of the form, $y = \theta_0 + \theta_1 x_1 + \dots \theta_p x_p + \varepsilon_i$, where ε_i is some noise term. The role our models play is to determine these coefficients (or weightings), θ_n.

We begin with examining the simplest model used for this, known as ordinary least squares.

1.2 Linear regression

1.2.1 Ordinary least squares (OLS)

Let us shift our focus to a scenario where we have multiple observations of our target variables and regressors, that is to say our linear model is now of the form

$$y_i = \theta_0 + \theta_1 x_{i,1} + \dots \theta_p x_{i,p} + \varepsilon_i \quad i = 1, \dots, n$$

where p represents the number of parameters (features) in our training set, and n the number of observations. Let's introduce the notation,

$$X = \begin{bmatrix} 1 & x_{1,1} & \dots & x_{1,p} \\ 1 & x_{2,1} & \dots & x_{2,p} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & x_{n,1} & \dots & x_{n,p} \end{bmatrix},\ Y = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix},\ \boldsymbol{\theta} = \begin{bmatrix} \theta_1 \\ \theta_2 \\ \vdots \\ \theta_n \end{bmatrix}$$

Here we refer to X as our feature set, and it includes a row consisting of only ones, which would be the intercept. The goal of the ordinary least squares regression is to minimise the following cost function by choosing appropriate coefficients $\boldsymbol{\theta}$,

$$L(\boldsymbol{\theta}) = ||Y - X\boldsymbol{\theta}||_2^2$$

where $||\cdot||_2$ is the $\mathbf{L^2}$ norm. Via differentiating and setting the derivative to zero, the solution to this minimisation problem is given by $\hat{\boldsymbol{\theta}} = (X^\intercal X)^{-1} X^\intercal Y$. There are many pros and cons associated with this model. The results of it can be easy to interpret, especially so if the data is normalised before the regression is carried out (a topic to be explained later). A regressor having a positive (negative) coefficient indicates that it is positively (negatively) correlated with the target variable. It is also overall a simple model to fit, and thus does not require much in terms of computing power/computation time. However there are drawbacks to it. Firstly, this model is very much subject to overfitting due to the lack of a regularisation penalty (a topic we will explore later). It also does not model or taken into account non-linear interactions between our target variable and the features we supply. Any prediction outputted will be a linear combination of our feature set, plus an intercept. Additionally, high levels of colinearity within the columns of our data can be troublesome (7), and this can result in $X^\intercal X$ with one or more small eigenvalues. Computer systems may be ill-conditioned to approximate an inverse to $X^\intercal X$ (which is in the equation for $\hat{\boldsymbol{\theta}}$), and thus even slight variations in data could cause large changes in our predictions (due to magnified effects of rounding error), with coefficients changing a lot for small changes in data. Thus we could see a large distance between $\hat{\boldsymbol{\theta}}$ and $\boldsymbol{\theta}$. We now move on to looking at models that implement a regularisation paramater, to tackle the issues of overfitting and colinearity.

1.2.2 Ridge regression

Avoiding overfitting is an important task in all regression models. Recall that the mean squared error of our estimation for $\boldsymbol{\theta}$ can be written as $\text{MSE}(\hat{\boldsymbol{\theta}}) = \text{Bias}^2(\hat{\boldsymbol{\theta}}) + \text{Var}(\hat{\boldsymbol{\theta}})$. In ordinary least squares, it can be shown that $\mathbb{E}(\hat{\boldsymbol{\theta}}) = \boldsymbol{\theta}$(6) , that is to say the OLS regression has no bias. We effectively choose to set our model to have no bias, and then use this to calculate our estimates with minimum variance. The role of regularisation is to include some bias in our estimates, however due to the bias variance tradeoff, we will have lower variance and potentially a lower MSE.

The ridge regression, originally developed by Horel and Kennard (7), adds a regularisation paramater of the form $\lambda||\boldsymbol{\theta}||_2^2$ to our cost function, and it aims to address the issues of overfitting and colinearity found in OLS. Its overall goal is to minimise the following over $\boldsymbol{\theta}$,

$$L(\boldsymbol{\theta}) = ||Y - X\boldsymbol{\theta}||_2^2 + \lambda||\boldsymbol{\theta}||_2^2$$

A closed form solution to this minimization problem exists, and it is

$$\hat{\boldsymbol{\theta}}^{\text{ridge}} = (X^\intercal X + \lambda I)^{-1} X^\intercal Y$$

Already, from an intuitive standpoint, one may note that the λ term may be of use when otherwise $X^\intercal X$ would not be invertible. The λ term here is to penalise high coefficients. As $\lambda \to \infty$ the coefficients shrink towards zero (however do not reach zero, so no feature selection occurs), and for $\lambda = 0$, we recover our OLS regression. The choice of λ comes down to a bias-variance tradeoff, as a high (low) λ results in higher (lower) bias, but also lower (higher) variance.

1.2.3 LASSO regression

In LASSO regression, developed by Robert Tibshirani 2006 (16), we apply a $\mathbf{L}^1$ norm penalty, giving us the following cost function,

$$L(\boldsymbol{\theta}) = ||Y - X\boldsymbol{\theta}||_2^2 + \lambda||\boldsymbol{\theta}||_1$$

Unlike ridge, we now can see coefficients being shrunk to zero, and thus we have variable selection occurring. One must ensure they appropriately choose λ via some form of cross-validation, to avoid a scenario with too sparse parameters. The continuous shrinkage also solves the issue with bias-variance trade off. This function is no longer differentiable, so no closed form solution exists to the LASSO, however there are efficient algorithms available for computing the entire path of solutions as λ varies (6) (Chapter 3.4.4).

1.2.4 ARMA(1,1) model

Finally, since we believe the 1 day lagged volume to be an important feature, as an alternative benchmark we will create an ARMA(1,1) model. The process $\{X_t\}_{t\in\mathbb{Z}}$ is an ARMA(1,1) process if it satisfies the following,

$$X_t - \phi X_{t-1} = Z_t + \theta Z_{t-1} \text{ for all t}$$

where $Z_t \sim WN(0, \sigma^2)$. Thus we see that this model takes new values to be a linear combination of the previous observation, a noise term, and a lagged noise term. A maximum likelihood estimation method is used to determine the coefficients ϕ and θ to fit the model to the data. Autoregressive models will be explored further during the intraday modelling section.

1.3 Non-linear machine learning models

1.3.1 Random forest

Before we explore the usage of random forests for regression, we must begin with the more fundamental building block of decision/regression trees.

Let us begin by considering the usage of a decision tree in a classification problem, following the approach of G. Louppe (2015), (10) In machine learning, classification refers to a predictive modeling problem where a class label is predicted for a given example of input data. For example, one could transform our regression problem into a classification problem by change our prediction goal to defining whether a day is to be a low, medium or high volume traded day. Assume we have a set of classes, $\{a_1, a_2, \dots, a_n\}$. Our target variable (true labels), Y defines a partition over the universe Ω. This partition can be denoted as such,

$$\Omega = \Omega_{a_1} \cup \Omega_{a_2} \cup \dots \Omega_{a_n}$$

Here $\Omega_{a\,i}$ are subsets of the entire space whose actual label is a_i. We can also denote by f a function which classifies objects, that defines a partition on input space X and provides us with an approximation $\hat{Y}$ of the true label space Y. However, this partition can only be defined on our input space X, and we denote it as such

$$X = X^f_{a_1} \cup X^f_{a_2} \cup \dots X^f_{a_n}$$

where now X_{a_i} are subsets $\boldsymbol{x} \in X$ such that $f(\boldsymbol{x}) = a_i$. Our goal is to learn a partition of $\mathcal{L}$ which matches (by some sort of loss/error criteria) the best possible partition, which is formed by the Bayes model f_b,

$$\mathcal{L} = \mathcal{L}^{f_b}_{a_1} \cup \mathcal{L}^{f_b}_{a_2} \cup \dots \mathcal{L}^{f_b}_{a_n}$$

The Bayes model minimises the expected prediction error. With this in mind, we now can see the methodology and principle of a tree structured model. We can define it as a model $f : X \to Y$, that approximates partitions of the Bayes model by partitioning the space X (feature space) into subspaces. We aim to recursively partition X until we can assign a constant prediction value $\hat{y} \in Y$ to all $\boldsymbol{x}$ in these subspaces.

Focusing now on regression problems, we would have $\hat{y} \in \mathbb{R}$, and we assign a constant value within each partition we create. Let us briefly explain some terminology used in decision trees, which will allow us to gain more intuition for the procedure. Our goal is to predict daily volumes, and thus we have root nodes, where our feature set gets initially split. A root node then gets split into two or more sub nodes, which then further get split into decision nodes. Finally, we reach a terminal node, or 'leaf', which contains a final volume prediction. Let us address how one comes up with this model f and suitable partitions for a regression tree. In classification problems, there are numerous criteria one can base it upon, such as information gain (12), and Gini impurity (18). However, the focus of this thesis is on regression problems, and thus we outline the approach found in chapter 9.2.2 of the Elements of Statistical Learning (6), based upon minimisation of sum of squares . Our goal here is to come up with an algorithm that is able to automatically split our tree up, with suitable splitting variables/nodes. Let us assume we have a partition M into regions $R_1, R_2, \dots R_m$. We model the target variable to take a constant value within each of these regions, which we denote as c_m. Thus our model function f can be defined as,

$$f_{dec}(\boldsymbol{x}) = \sum_{m=1}^{M} c_m \mathbb{1}_{\{\boldsymbol{x} \in R_m\}}$$

where $\boldsymbol{x}$ are our observed features. What value of c_m do we choose? If our goal is to minimise squared loss, the average of all observed points y_i in the region R_m (i.e. $\hat{c}_m = \text{ave}(y_i|\boldsymbol{x}_i \in R_m)$). However, finding the optimal binary partition numerically is not computationally feasible, thus a greedy algorithm must be implemented instead. Giving a brief overview of this, the general idea is setting up a splitting variable j and a split point s, and considering the half planes,

$$R_1(j,s) = \{X|X_j \leq s\} \text{ and } R_2(j,s) = \{X|X_j \geq s\}$$

We split X into two disjoint subspaces, however we must optimise over s and j for this. We choose to solve the following minimisation problem, that focuses on the sum of squared distances for both planes,

$$\min_{j,s} \left[\min_{c_1} \sum_{\boldsymbol{x}_i \in R_1(j,s)} (y_i - c_1)^2 + \min_{c_2} \sum_{\boldsymbol{x}_i \in R_2(j,s)} (y_i - c_2)^2 \right]$$

As we mentioned before, for a sum of squares minimisation, the optimal solution is an average of all observed points in a region. More formally, we can differentiate and take first order conditions over the inner minimisation, yielding

$$-2 \sum_{\boldsymbol{x}_i \in R_1} (y_i - c_1) = 0 \implies \sum_{\boldsymbol{x}_i \in R_1} (y_i - c_1) = 0 \implies \sum_{\boldsymbol{x}_i \in R_1} y_i = c_1|R_1| \implies \hat{c}_1 = \frac{1}{|R_1|} \sum_{\boldsymbol{x}_i \in R_1} y_i$$

This process continues iteratively, as we continue splitting the data into further optimal splits on resulting split regions. A hyperparamater here to tune is how deep the tree should go (i.e. the number of splits we do). Again here, we have the usual crux of a bias variance tradeoff, as a tree which goes too deep (shallow) can overfit (underfit) data, resulting in high (low) variance and low (high) bias. There are various 'stopping criteria', which can be enforced to avoid this, however we simply enforce that trees stop when all leaves are pure (i.e. contain a single value only). As a result, decision trees can be prone to high variance, and a slight difference in training set can result in largely different results. Thus, the approaches of bagging and finally a random forest are used to address this. Let us first explore bagging.

We follow the approach outlined by L.Brieman (3). We take $\mathcal{L} = \{(\boldsymbol{x}_1, y_1) \dots (\boldsymbol{x}_n, y_n)\}$ as the set of observed values (i.e. a training set) from the joint distribution of (X_t, Y_t). Assume we have a method (i.e. a regression) for forming a predictor, $f_{\mathcal{L}}(\boldsymbol{x})$ for y when we observe $\boldsymbol{x}$. In order to reduce the variance of our model, we would ideally have a sequence of identically distributed training sets, $\{\mathcal{L}_k\}$, with n independent observations each. Assuming we have N training sets, we could take an average over our training sets to form a prediction. However, we often do not have multiple training sets, thus one can take repeated bootstrapped sample $\{\mathcal{L}^{(B)}\}$, forming predictors $\{f_{\mathcal{L}}^{(B)}(\boldsymbol{x})\}$ (curly brackets distinguish sequence from individual item in sequence). Each of these bootstrapped training sets, $\mathcal{L}^{(B)}$ consists of N samples, drawn randomly from $\mathcal{L}$, with replacement. The probability of a sample being in a bootstrapped sample $\mathcal{L}^{(B)}$ is

$$\mathbb{P}(\boldsymbol{x} \in \mathcal{L}^{(B)}) = 1 - \mathbb{P}(\boldsymbol{x} \notin \mathcal{L}^{(B)}) = 1 - (1 - \frac{1}{N})^N \to 1 - e^{-1} \text{ as } N \to \infty$$

Then, assuming we have M total bootstrapped samples, one can set $f_{\mathcal{L}}^{(\text{bag})}(\boldsymbol{x}) = \frac{1}{M}\sum_{m=1}^{M} f_{\mathcal{L}}^{m}(\boldsymbol{x})$ (i.e. take average over all of our bootstraped samples). In Briemann's original paper, the usage of bagging regression trees resulted in a reduction in test set mean squared error between 21% to 46%.

Finally we can address the random forest algorithm, which further reduces the variance of our estimates. This algorithm works by averaging out the predictions taken from B trees, where bootstrapped samples are used to build each tree using the aforementioned bagging approach. Considering a sequence of trees $\{T_b\}_{i=1}^{B}$, our model function is a slight variant of the one for our decision tree, and is given by

$$f_{rf}^{B}(\boldsymbol{x}) = \frac{1}{B} \sum_{b=1}^{B} T_b(\boldsymbol{x})$$

This approach of averaging over numerous unbiased (albeit noisy) trees is key in reducing variance of our model. If we consider each of our trees to be identically distributed with variance σ^2, but not necessarily independent, we can calculate the variance of the average. Taking B trees, we have,

$$\mathrm{Var}\left(\sum_{b=1}^{B} T_b(\boldsymbol{x})\right) = \mathrm{Cov}\left(\sum_{b=1}^{B} T_b(\boldsymbol{x}), \sum_{b=1}^{B} T_b(\boldsymbol{x})\right) = \sum_{b=1}^{B} \mathrm{Var}(T_b(\boldsymbol{x})) + \sum_{b\neq c} \mathrm{Cov}(T_b(\boldsymbol{x}), T_c(\boldsymbol{x}))$$

The first term is just the sum of the variances of our B trees, and is simply $B\sigma^2$. For the second term, we are summing covariance where indexes are not equal, with $B \times B$ total terms, minus B repeated, so $B(B-1)$ occurrences , with each term being $\rho\sigma^2$ (the covariance between trees). Putting all this together, the right hand side in the equation above can be re-written as,

$$B\sigma^2 + B(B-1)\rho\sigma^2$$

If we consider taking the average, i.e. we divide our initial sum of variances by $\frac{1}{B}$, we realise we must multiply the above by $\frac{1}{B^2}$, yielding that the variance of the averages is,

$$\rho\sigma^2 + \frac{1-\rho}{B}\sigma^2$$

We see, as the number of trees B increases, the 2nd term tends to zero. The purpose of a random forest is improving the variance reduction of bagging via reducing the correlation between individual trees, without increasing total variance by a large amount.

1.3.2 Multilayer perceptrons (MLPs)

Neural networks are thought to have been originally developed by Walter Pitts and Warren McCulloch in 1943 (11), outlined in a research paper on the neural networks of the human brain. They are credited with a computational model, based upon an algorithm known as threshold logic in order to mimic the thought process of the human brain. Since then, neural networks have been a highly investigated topic in science, for both the purposes of modelling human biological processes in the brain, and for artificial intelligence, the latter of which is our focus here. Multilayer perceptrons (MLPs) are one of the most popular neural networks, which can be used to model non-linear dependencies between our dependent and independent variables. It is useful to consider a simple neuron like unit, known as a perceptron. Consider the following diagram below.

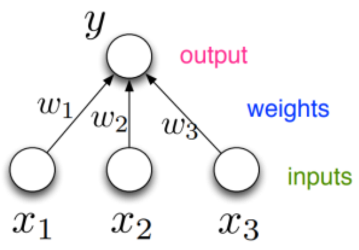

Figure 1.1: A simple perceptron model

The circles in this diagram are known as neurons, with an input layer consisting of the neurons $\{x_1, x_2, x_3\}$, and these represent the feature set in our regression problem. We can see that the output neuron is connected to the input neurons, which has weights passed into it from the input neurons, which can broadly be thought of as similar to the coefficients for features. The weights are summed up and a bias is added, resulting in the pre-activation sum, defined as

$$PreActiviationSum = \sum_k w_k x_k + b$$

This scalar here b represents the bias. The pre-activation sum is then passed through an activation function, which places the final value into a bounded range. Examples of activation functions include the sigmoid function ($f(x) = \frac{1}{1+e^{-x}}$), tanh ($f(x) = \frac{e^{x}-e^{-x}}{e^{x}+e^{-x}}$) and ReLu ($f(x) = max(0, x)$). The neural network needs to be trained in order to correctly determine these weights. However now that we have explored the general intuition surrounding perceptrons and neural networks, let us expand this by looking at MLPs.

An MLP consists of multiple different perceptrons (networks that we described above), all connected to one another. Our input is as above, a set of features X, and we set our target as Y, the volume traded in a given day.

We will create a network consisting of 3 layers, an input layer, a hidden layer, and an output layer that will output a single figure, being the estimate for volume.

(i) **Input Layer** - As explained above, this contains a set of neurons $\{x_1^{(1)}, x_2^{(1)}, \dots x_m^{(1)}\}$, which represent the m different features we are inputting in the model. The superscript 1 represents us being in the first layer. Each of these features are then passed onto each of the neurons in the hidden layer.

(ii) **Hidden Layer** - The hidden layer captures any non-linear dependencies in the data. Initially, random weights are assigned to these features in each neuron, and as in the single perceptron case, these are combined into a preactivation sum and passed into a non-linear activation function. Then each of these neurons inputs this value to the single neuron in the output layer.

(iii) **Output Layer** In the output layer, once again a weighted linear sum of the inputted signals is taken, including the bias. No activation function is used here, due to this being a regression problem.

What we have just described is known as the *forward activation* of the neural network. For the neural network used in this thesis, we have a total of 21 features, thus our input layer contains 21 neurons. The hidden layer is chosen to have 15 neurons, after testing for it as a hyperparamater. For a single observation (i.e. one day), we can express the values at each layer as follows. In our initial layer, we would have $X^1 = \{x_1^{(1)}, x_2^{(1)} \cdots, x_{21}^{(1)}\}$, in our second layer $X^2 = \{x_1^{(2)}, x_2^{(2)} \cdots, x_{15}^{(2)}\}$ and in our last layer $x_1^{(3)}$. Find below a diagram of the overall structure of our network.

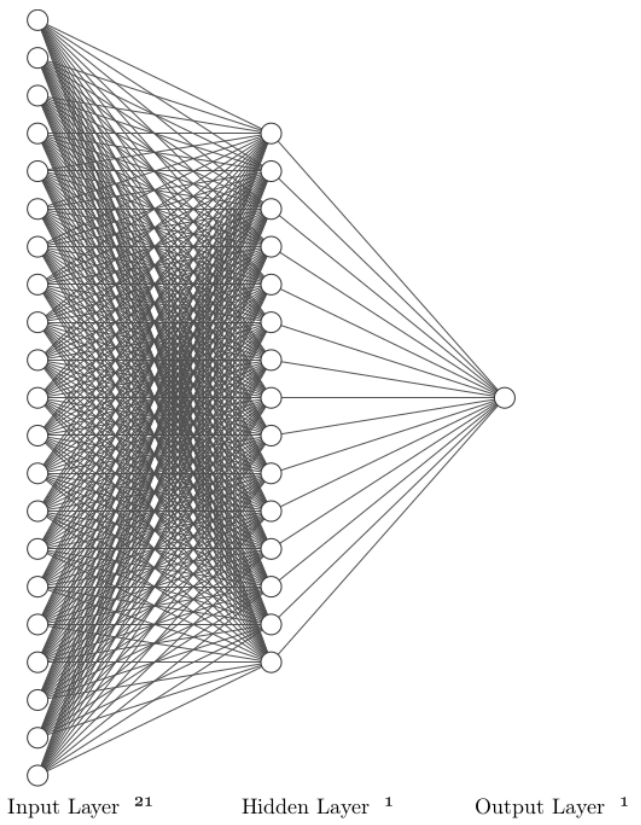

Figure 1.2: The architecture of our network - with 21 neurons in initial layer, 15 in hidden and 1 in output.

We can express the weights between neurons mathematically. Between our input and hidden layer, we can have a 21×15 matrix, and for our hidden layer to output layer a 15×1 matrix represented as follows,

$$W^1 = \begin{bmatrix} w_{1,1}^{(1)} & w_{1,2}^{(1)} & \dots & w_{1,15}^{(1)} \\ w_{2,1}^{(1)} & w_{2,2}^{(1)} & \dots & w_{2,15}^{(1)} \\ \vdots & \vdots & \ddots & \vdots \\ w_{21,1}^{(1)} & w_{21,2}^{(1)} & \dots & w_{21,15}^{(1)} \end{bmatrix}, \; W^2 = \begin{bmatrix} w_{1,1}^{(2)} \\ w_{2,1}^{(2)} \\ \vdots \\ w_{15,1}^{(2)} \end{bmatrix}$$

Likewise, each layer also has its own bias. With all this considered, we can express the networks computations as follows,

$$x_i^{(2)} = \sigma^2\left(\sum_j w_{ij}^{(2)} x_j^{(1)} + b_i^{(2)}\right)$$
$$x_i^{(3)} = \sigma^3\left(\sum_j w_{ij}^{(3)} x_j^{(2)} + b_i^{(3)}\right)$$

with b_i^k representing the bias in i^{th} neuron in the k^{th} layer and σ^k representing our activation function for each layer (i.e. in the output layer, it is just the identity). Here the first equation represents the output of the hidden layer, and the second equation represents the output of the output layer (i.e. our prediction). In vector form, we can write the component equation as follows. Here l represents the layer,

$$x^{(l)} = \sigma^{(l)}(w^{(l)}x^{(l-1)} + b^{(l)})$$

As mentioned above, the weights initially used are randomised, as well as the bias. The model uses this feed-forward loop for training, as this will produce a prediction result that can be measured against the target value. Then, backward propagation can be applied in order to adjust these weights and biases to further train the model. One total cycle of forward and backward propagation is known as an epoch, and the number of epochs is a hyperparamater for the model that needs to be trained. Select too few epochs and the model may underfit the data, too many and it will overfit.

Once our initial prediction is made, for each of our observations, we input this into a loss function in order to determine the goodness of fit of our weights. A typical loss function used (and the one used in this thesis), is the quadratic cost function, which can be expressed using the notation from above as

$$C = \frac{1}{2N}\sum_i ||y_i - x_i^{(3)}||^2$$

Note the superscript in $x_i^{(3)}$ is 3, indicating this is the output of our 3rd layer in our network, i.e. the prediction we have for the i^{th} sample in our data set. There are some requirements our cost function should meet for certain gradient descent algorithms, which are of common use. Firstly, they must be able to be decomposed into an average over the respective samples. That is to say, $C = \frac{1}{N}\sum_i C_i$. In this thesis, the gradient descent algorithm which is used is known as Adam, however we will start from just general gradient descent, and also explain stochastic gradient descent. Gradient descent is an interative optimisation algorithm, that we use to find the local minimum of the cost function. Intuitively, the idea behind gradient descent is we approximate the gradient function, trying to find the minimum of the cost function, and repeatedly adjust and move our position on the gradient curve in order to reach the 'bottom' (i.e. the minimum). In general gradient descent (GD), these updates for the weights and bias take the following form.

$$w_{k+1} = w_k - \eta\nabla_w C(w_K)$$
$$b_{k+1} = b_k - \eta\nabla_b C(b_k)$$

where $\nabla_w C_t$ is gradient of our cost function with respect to time step t. From now on for ease, instead of specifying updates for bias and weights, we just refer to paramaters θ. The initial values w_0 and b_0 are initialised randomly. The value η here represents the 'learning rate'. A low value could result in the algorithm taking a very long time, while if it is too small the algorithm may not be able to find the minimum, due to it overshooting. Since in GD every instance in the training set requires a prediction, the algorithm can take very long to run. Thus a variant of it known as stochastic gradient descent (SGD) can be employed instead. Due to high computational costs, when one carries out back propagation via a gradient descent method, one does not usually use the entire training sample for it. Instead, it is typical to take a subset $M \subseteq \{1, 2, \cdots n\}$, of size m of the training set. Thus, the gradient of the cost function can be approximated using a subsample of data as follows,

$$\frac{\sum_{i\in M}\nabla C_i}{m} \approx \frac{\sum_{i=1}^{n}\nabla C_i}{n} \approx \nabla C$$

This size m becomes another hyperparamater one can tune, known as the batch-size used in gradient descent. SGD uses these batches, instead of the entire sample size. As outlined by Keskar et al (8) a low batch size is associated more so flat minimizers of the cost function, while a larger batch size is associated with more sharp minimisers. These sharp minimisers tend to generalise more poorly to non-training data, due to large sensitivity of the training function at a sharp minimizer. Find below a diagram expressing the generic shape of sharp/flat minimisers.

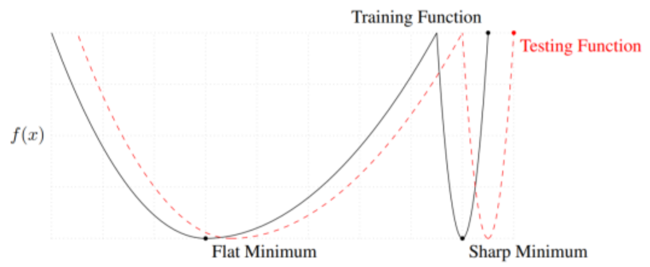

Figure 1.3: Visualisation of sharp vs flat minimisers, with the y-axis representing the value of a loss function and x-axis the variables.

Mathematically, considering a stochastic cost function C with respect to some paramaters θ, SGD can be summarised as follows,

$$\theta_{t+1} = \theta_t - \alpha_t \left(\frac{1}{m} \sum_{i \in B_k} \nabla_\theta C_i(\theta_t) \right)$$

where m is our batch size, and B_k is the k^{th} batch, and the subscript i in C_i is just used to clarify it is the loss function for data point i.

As previously mentioned, the descent method we use throughout our thesis is known as Adam gradient descent (9). On a high level, Adam works via calculating individual adaptive learning rates for different parameters by using estimates of first and second moments of the gradients, the first moment being the mean and the second the uncentered variance. Adam's update rule can be summarised as follows,

$$\begin{aligned} m_t &= \beta_1 m_{t-1} + (1-\beta_1)\nabla_\theta C(\theta_t) \\ \hat{m}_t &= \frac{m_t}{1-\beta_1^t} \\ v_t &= \beta_2 v_{t-1} + (1-\beta_2)(\nabla_\theta C(\theta_t))^2 \\ \hat{v}_t &= \frac{v_t}{1-\beta_2^t} \\ \theta_t &= \theta_{t-1} - \alpha \cdot \hat{m}_t / (\sqrt{\hat{v}_t} + \varepsilon) \end{aligned}$$

The first and third update rules correspond to updating biased first and second moment estimates, while the second and fourth update rules apply a bias correction. Finally, the last rule updates the paramaters. This algorithm is simply taking an exponential moving average of the first and second moments of the gradient, with parameters β_1 and β_2 controlling the exponential decay. The bias correction is needed since the moving averages are initialized as a vector 0s, thus the initial moment estimates are biased towards 0.

In terms of the activation function used, a ReLu function, defined as $f(x) = max(0, x)$ is opted to be used, which is computationally efficient and also does not suffer from the vanishing gradient problem found in the tanh and sigmoid activation functions.

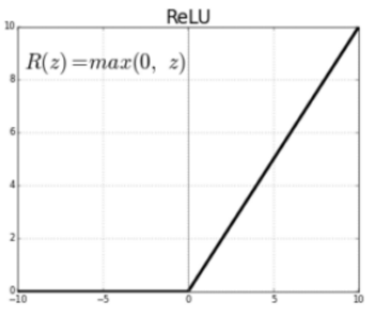

Figure 1.4: ReLu function graph

1.3.3 Principle component analysis (PCA)

PCA is an unsupervised machine learning technique, originally developed in 1901 by Karl Pearson (15). Despite the process being developed quite a few years ago, it can still provide some insight. The purpose of PCA is to perform a dimensionality reduction on high-dimensional data. Given a set of variables, PCA creates principal components, which are linear combinations of our original variables/columns, in order to maximise the variance in our sample.

We carry out our original PCA on the 5 bonds closest to 2,5,10 and 30 years to maturity in France, Germany and Italy. For the rest of this subsection, we will refer to these categories of bonds as sectors, and these effectively act as the features in our PCA. The reason for this is that a specific bond today will not act similarly in 3 years time say, due to it being at a different point in its individual life cycle, thus by taking bonds at static points in the curve we are getting a more time invariant view of bonds. The PCA is carried out on the weekly deltas of these bonds, as we take linear combinations of these for different bonds for the purpose of dimensionality reduction. Thus with 2 principal components, we aim to view our data in a 2-dimensional space, whilst still hopefully maintaining enough variance to make this dimensionality reduction still meaningful.

The purpose of this PCA is we can view which of these sectors get clustered together in a loading graph. The loadings are the coefficients with respect to the first and second principal component for each column/sector.

1.4 Training and test data split

A key concept in machine learning algorithms is the idea of a test/training split. If one does not provide said split, the model may be overfitted to the data, meaning when our model is confronted with data it has not seen before it performs very poorly. By keeping our training and test data separate we can avoid this issue, and we will be evaluating all performance of our model on data which it has not been trained upon. This is known as out-of-sample performance. For all the algorithms below, we will slightly modify them when implementing our methods to account for an extra feature selection step, as well as hyperparamater optimisation, however for now we present them in their most general form.

1.4.1 K-fold

One commonly used train/test split is a k-fold split. A visual representation of this split is as follows.

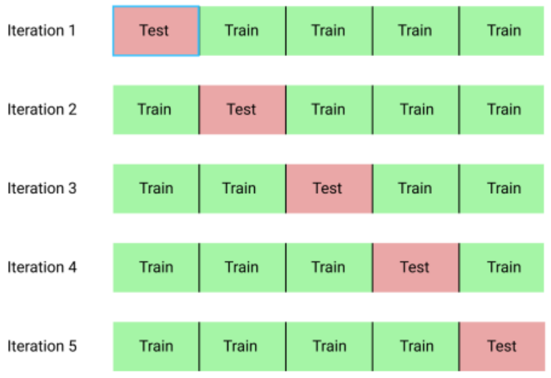

Figure 1.5: K-fold train/test split

Its algorithm can be summarised as below,

(i) Divide entire data set into k equal sized 'folds'.

(ii) Set fold $n = 1$ as our test fold, and the other $k - 1$ folds as our training folds.

(iii) Fit data to our training folds (i.e. determine weightings for our features).

(iv) Use these weightings on the test set to predict results on that set. Store these results

(v) Shift the training set, increasing n by 1 and set the the other folds as training. Repeat steps 3 and 4. Continue increasing n by 1 until we reach the end of our data set.

(vi) Report performance via a performance metric, seeing how our predictions compare to the actual results.

The main issue with this, making it ill fitted for time series, is that it naturally can disregard the temporal nature of the data, since if our test fold is not either at the start or end of the data set, our training data will not be a continues series of time points. Furthermore, this method could not be used on a daily basis to update predictions, since it requires data from the future in order to predict data today. We present a further test/training splits to generate predictions from a set of data, without directly training on the data you wish to predict.

1.4.2 Nested

A nested split is an alternative to a k-fold split, which does not suffer from the problems specified above. A visual representation of the split is as follows.

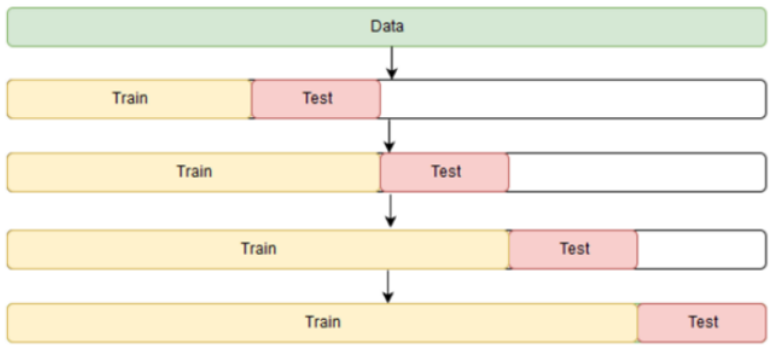

Figure 1.6: Nested Train/Test Split

The procedure is as follows,

(i) Specify an initial training set. In our analysis, we take this to be 8 months.

(ii) Fit data to training set (i.e. determine weightings for our features).

(iii) Test/predict on 5 days data.

(iv) Add these 5 days to the new training set, and predict the next 5 days. Continue this process until reaching the end of the data set.

Using this train/test split ensure we manage to get a large amount of out-of-sample predictions, without only being able to train on a small sample.

Chapter 2

Feature creation and data cleaning

In this section we outline various features in the model that were created, as well as various actions we must take on our data before we can use it for predictions.

2.1 Feature creation

There are several caveats we must address before commencing with this section.

(i) Avoid using features that are linear combinations of each other. An example of this would be including 5 days of volume look back and also including a 5 day volume average. The reasons for this were discussed above, in terms of the colinearity issues ran into by OLS. Furthermore, issues could arise in feature selection (covered in a later section).

(ii) Avoid all ‘look forward’ or ‘peeking’ features. For example, including today’s volume in a feature such as monthly volume average or normalising data points by including future data, which can implicitly leak information and give artificially high performance. Note from now on, terms such as n days average, means the average of the *previous* n days (i.e. not including today).

(iii) Avoid ordinality if it is not originally intended. For example, do not use days of the weeks as features, with each day represented by an integer $1, 2 \dots 7$, as this may result in the model treating certain days with unnecessary importance compared to others. Instead, for something such as days of the week, a ‘one-hot-encoding’ approach should be used (explained later in this section).

We now outline our features by splitting them into similarity groups.

2.1.1 Volume based features

Volume traded on previous days can be indicative of the volume to be traded today, and these features aim to capture this relationship.

- Volume traded at a $1, 2, \dots, 5$ day lag.
- 2 week, 3 week and 1 month average volume traded (not including today’s volume).

- Yesterday's flow inbalance. This is more formally defined as, $|x - 0.5|$ where

$$x = \frac{BuyOrderQuantity}{BuyOrderQuantity + SellOrderQuantity}$$

  This feature is included to explore whether there is a link between areas of high/low volume and an order inbalance between buy/sell orders. The transformation with the modulus is applied since we are interested in how far our flow deviates from a 50:50 split between buy and sell orders.

2.1.2 (Special) days

As well as what day of the week it is, there are also various other dates which are of particular importance in the EGB market.

- Day of the week. Done using a one-hot-encoding approach. In one hot encoding, a column such as 'day of the week' which contains 5 unique values (the weekdays), is transformed to 5 different columns which operate as indicator functions, having a 1 if it is that day of the week and 0 otherwise.

  Observe the following graph below, which plots the volume traded for each day of the week with confidence intervals for all German sectors,

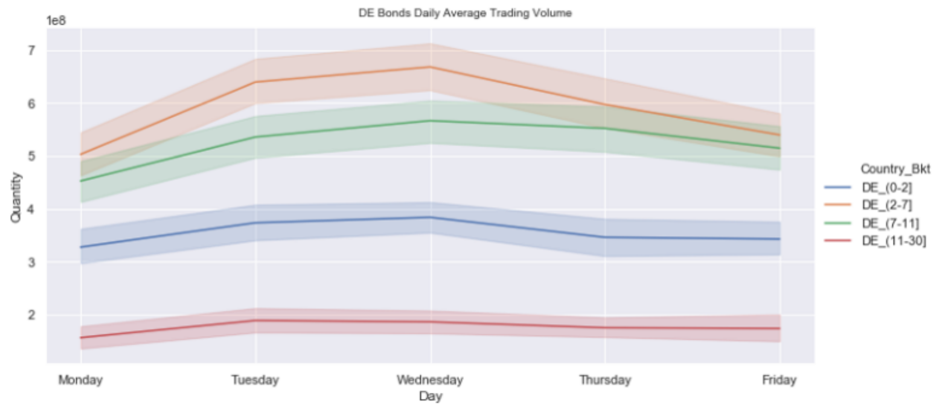

Figure 2.1: German sector daily volumes

  For the more liquid German sectors $(0-2, 2-7)$ we note that generally a day such as Monday is traded lower than a Wednesday, and we can see why it is important to include this as a feature.

  However, now by circumventing any issue of false ordinality via one-hot encoding, we have created a new issue, being multicolinearity. On any given day we have that the sum of all 'Day of week' columns is 1, and each feature/day can be expressed as a linear combination of the other 4. We arbitrarily choose to drop 'Day of week Thursday', from our features (we could choose any day). This does not actually remove any information from the model, since this column was just a linear combination of the other day of the week columns, and it solves the issues of multicolinearity.

- Last day of the month (a boolean). The bond market can see an uptake of activity on the final days of a month, due to fund managers who aim to track various indexes buying/selling certain bonds at months end to re-weigh their portfolio correctly. Observe the following graph comparing final day of month volumes vs average in the rest of the month in 2020 for German $7-11$ year bonds.

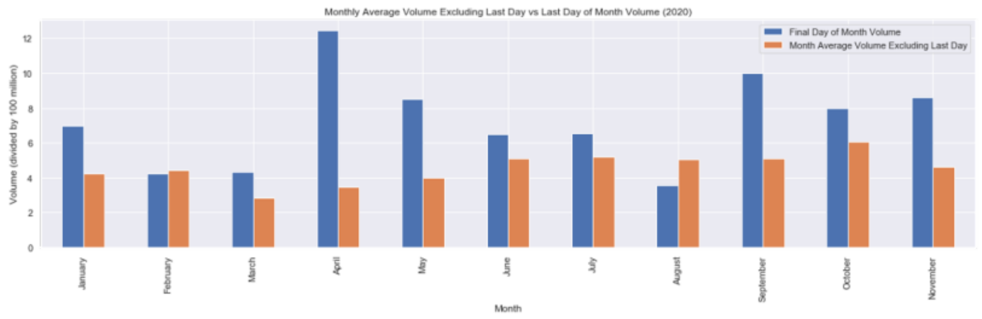

Figure 2.2: End of month volume comparisons

- European Central Bank (ECB) meeting dates. There could be a change in activity when an ECB meeting is approaching.
- Rollover dates for futures contracts. Futures contracts are used to set up hedges in the bond market. When these contracts are close to expiring, we enter a roll over period, where many of these futures hedges need to be 'rolled over' to the next available contract.

2.1.3 Bond pricing

The spreads and logarithmic returns of bonds within a given sector could be indicative of the volumes at which they get traded.

- Exponentially weighted moving average (EWMA) of volume weighted spread. For a given day t, and a set of spreads for N bonds in our given sector, the volume weighted spread is the following,

  $$VolumeWeightedSpread(t) = \sum_{i=1}^{N} \frac{Spread(t,i) \times Volume(t,i)}{TotalBucketVolume(t)}$$

  For $\alpha \in (0,1)$, and time $t \in \{1,\dots,T\}$ the EWMA $\mathbb{N} \times (0,1) \to \mathbb{R}$

  $$\begin{aligned} EWMA(0,\alpha) &= VolumeWeightedSpread(0) \\ EWMA(t,\alpha) &= \alpha \times VolumeWeightedSpread(t) + (1-\alpha) \times EWMA(t-1,\alpha) \end{aligned}$$

  where the α paramater controls the 'memory' of the time series. A higher α places more weight on more recent data points.
- Yesterday's median logarithmic returns for bonds in bucket.
- EWMA of daily median logarithmic returns for bonds in bucket.

2.2 Data normalisation

Given the features we have listed above, it is clear that these have very different ranges. Thus, we must normalise our data to a common scale, without distorting the differences in ranges of values. When it comes to comparing the coefficients assigned by our models, this normalisation will allow us to understand better the effects each feature has on our target variable. As previously mentioned, it is essential to avoid normalising the training and test sets in one go, instead doing them separately, as otherwise there is an implicit information leak into the test set. There are various normalisation techniques to speak of, including

- **Mean-Standard Deviation Normalisation**. Here we use a rolling window of size N to normalise our data, subtracting the mean and dividing by the standard deviation. The formula is the following,
$$\widetilde{X_t} = \frac{X_t - \frac{1}{N}\sum_{i=1}^{N} X_{t-i}}{\sqrt{\frac{1}{N}\sum_{i=1}^{N}(X_{t-i} - \bar{X})^2}}$$
This works well on data which does not originally resemble a normal distribution.
- **Min-Max Scaling** This translates each feature individually, using a rolling max/min, such that it is now in the closed interval $[0, 1]$. It applies the following formula,
$$\widetilde{X_t} = \frac{X_t - \min_{i \in [0,t-1]} X_i}{\max_{i \in [0,t-1]} X_i - \min_{i \in [0,t-1]} X_i}$$

Throughout this thesis, we apply the first normalisation.

2.3 Stationarity

Machine learning models tend to perform better with data that is stationary (constant mean and variance in time, and time invariant covariance that can however depend on lag length), and normally distributed. Thus, we must test whether this is the case for our data. Below, find a diagram of the ACF plot for the daily volume traded in German 7-11 year bonds

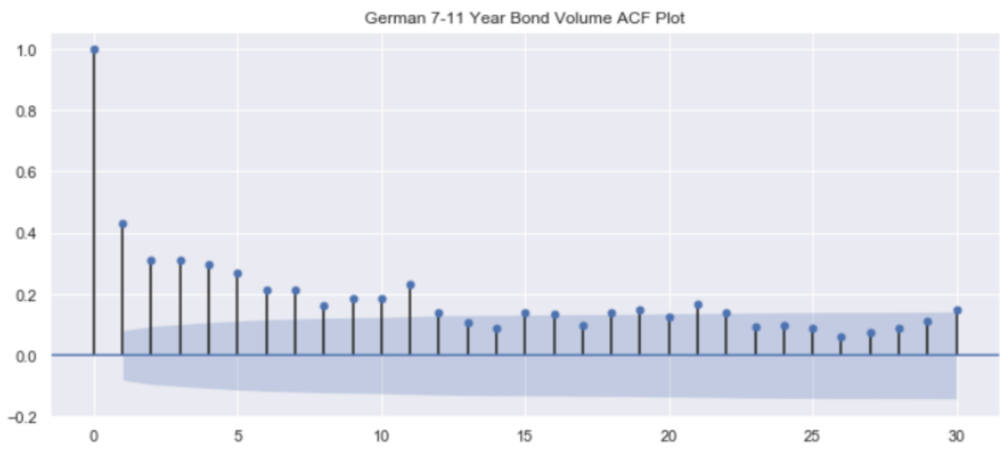

Figure 2.3: German $7-11$ year bond ACF Plot

While this graph does seem to suggest that our time series may be non-stationary, since lags are correlated, we investigate this further with a more quantitative method opposed to a qualitative one, known as a Dickey-Fuller test.

2.3.1 Dickey-Fuller test

If we consider an AR(1) model of the form

$$y_t = \phi y_{t-1} + \varepsilon_t$$

where ε_t is white noise term, we know that a unit root is present if $\phi = 1$, and in this case our process is non-stationary. Subtracting y_{t-1} from both sides, and we now get,

$$\Delta y_t = (\phi - 1)y_{t-1} + \varepsilon_t = \gamma y_{t-1} + \varepsilon_t \quad \text{with} \quad \gamma := \phi - 1$$

A Dickey-Fuller test now carries out a hypothesis test, with the null hypothesis being $\gamma = 0$, that is that the process is non-stationary. In terms of our time series, the model used in a regular Dickey-Fuller test would be of the form $y_t = \phi y_{t-1} + \alpha + \beta t \varepsilon_t$, with this constant α term accounting for any potential drift, and the βt term accounting for trend. Additionally, the Augmented Dickey-Fuller test, which we implement, would include high-order regression processes by including Δy_{t-p} in this model. The test statistic achieved by carrying this test out on German 7-11 year bonds was -6.216084, corresponding to a p-value of 0.000001, thus we reject the null hypothesis that the data set is non stationary. As a result, we do not carry out any differencing/fractional differentiation on our data set before carrying our regression out.

2.4 Feature selection

To avoid overfitting our data to a test set, we devise an algorithm to filter out features which do not provide a unique signal. Below we outline a feature selection algorithm that gets used on 6 month intervals. Given a data set, we use a 70:30 split to separate it into training and testing data, and perform the following algorithm. All removed features are removed for the next 6 months, until the algorithm is re-ran.

Algorithm 1 Feature selection

```
input: InputtedData = X-train, y-train, X-test, y-test datasets. Let $F = [f_1, f_2, \dots f_n]$ be a list of
       all features of the training set.
for $i$ in range(1000) do
    $MSE_{benchmark} = 10000000$
    F = RandomlyShuffle($F$)
    RegressionFeatures = {}
    for feature in $F$ do
        RegressionFeatures.append(feature)
        MSE$_{new}$ = $MSE(LinearRegression(InputtedData, RegressionFeatures))$
        if $MSE_{new} < 0.99 * MSE_{benchmark}$ then
            $MSE_{benchmark} = MSE_{new}$
        else
            RegressionFeatures.remove(feature)
    return RegressionFeatures
Count number of times each feature picked after 1000 shuffles. Remove all picked < 100 times.
```

Below we report the results of running this feature selection algorithm on the German 7-11 year bond sector.

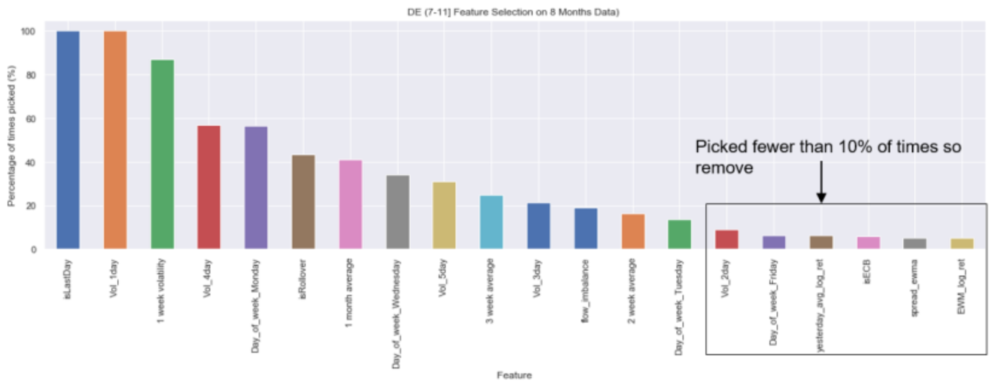

Figure 2.4: German 7-11 year bonds feature selection on initial training period

We see that the 1 day lagged volume and the final day of the month are very important features, while the pricing features generally do not increase our predictive power much and do not get picked by the feature selection algorithm.

2.4.1 Hyperparamater cross-validation

Additionally, we run inner loops for hyperparamater cross validation. In our ridge/LASSO models, this corresponds to the aforementioned λ penalty, that shrinks coefficients. For our random forest, we use a grid search method to optimise various hyperparamaters, such as the number of features to be considered in each tree, as well as the number of samples required to be in a leaf node. For the MLP, we again use a grid search method to tune the number of epochs, the batch size and the learning rate of Adam gradient descent. We present below a graph of mean square error with respect to the size of the λ paramater in a ridge.

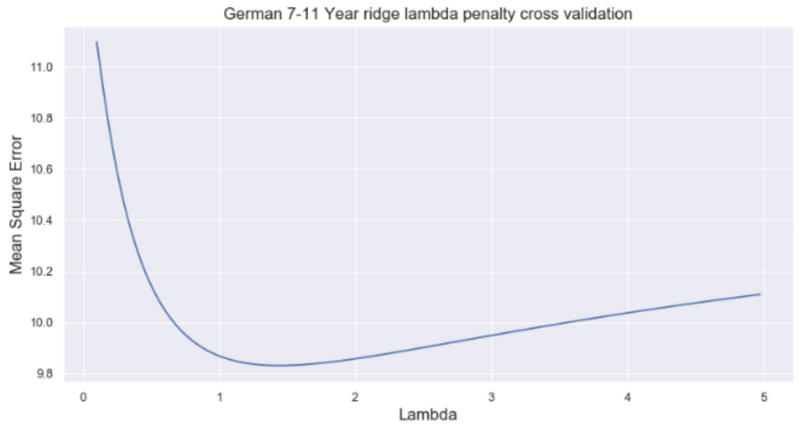

Figure 2.5: Model loss for training and test set with respect to epoch

We see the graph reaches a minimum and thus we would hope for our cross validation to select this point as our λ paramater for our regression.

Additionally, we present below a graph of the loss (mean square error) on both the training and test set for our MLP.

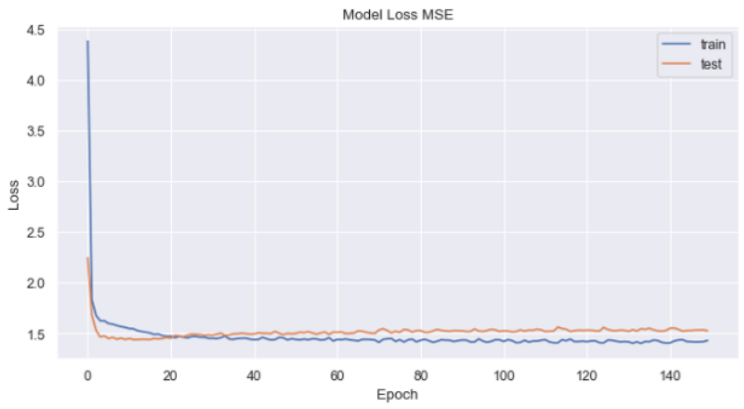

Figure 2.6: Model loss for training and test set with respect to epoch

Here we observe that increasing epochs seems to continuously improve performance on the training set, however it only does this to a certain point on the test set, before it begins overfitting and our MSE begins to increase.

Chapter 3

Performance metrics & results

3.1 R-Squared

The coefficient of determination, R^2, for N observations of data, is defined as follows,

$$R^2 = 1 - \frac{\sum_{i=1}^{N}(y_i - \hat{y_i})^2}{\sum_{i=1}^{N}(y_i - \bar{y})^2}$$

where y_i are observations of our target variable, $\hat{y}_i$ is the models output (predictions) and $\bar{y}$ is the average of the observations of the target variable.

Qualitatively this can be thought of as what proportion of the variance seen in results our model is able to explain. R^2 lies between $[-\infty, 1]$. An R^2 of 1 implies that the model perfectly describes the relationship, since $\Sigma(\hat{y}_i - y_i) = 0$ (i.e we have no difference between our predictions and the target variable). An R^2 of 0 implies the model only does as well as simply taking the average of the entire target variable and finally a negative R^2 implies it does worse than the average. This is one advantage of R^2, that since it is compared to a baseline it is normalised, and thus easy to interpret. It does however has its shortcomings. Firstly it is unable to show whether estimations are biased. Furthermore, if a dataset is non-stationary (i.e. let's assume there is a large upwards drift), the R^2 over the entire dataset may be 'artificially' high, however if one was to take a subset of this data to examine, where this drift is less evident, the R^2 would be different. For these reasons listed above, we use further performance metrics.

3.2 Mean absolute error

Mean average error (MAE) is defined as follows,

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

This metric measures the average distance of our predictions from their actual values. Evidently, a lower MAE is desirable. One benefit is that it has the same unit as the variable we are trying to predict, giving it greater interpretability. However, if we are comparing model performance over two different sectors for example, which have vastly different volumes traded, it can be difficult to determine which sector the model performs better on since this metric is not normalised. Furthermore, the function is non-differentiable.

3.3 Results

Below we find the out of sample results for our models, based on R^2, for the German 7-11 year bond sector.

Table 3.1: German 7-11 year bonds results summary

Model	R^2 **Score**	**MAE**
5 Day SMA	0.172	1.50
OLS (with feature selection algorithm)	0.238	1.48
Ridge (with feature selection algorithm)	0.248	1.47
LASSO	0.250	1.47
Random Forest	0.184	1.51
Multilayer Perceptron	0.270	1.44
ARMA(1,1)	0.235	1.48

All the models beat the benchmark in terms of R^2 and MAE, which is a success. The random forest models appears to perform the worst out of all our models, perhaps due to the instability of hyperparamater optimisation. During testing it was discovered that even slight variations of the training set could result in different optimal hyperparamater values, meaning any sort of optimisation would in fact be somewhat redundant. The linear regression models all perform well, and there is not very much to separate them. When we expand the model to look at other sectors, we will be able to compare this easier. The best performing model is our neural network. While it having the highest R^2 and lowest MAE is good reasoning to use it, the neural network does have to make many sacrifices for this compared to our regression models, namely in the spaces of computational time, as well as interpretability. Regressions output coefficients which make interpreting the correlations between our dependent and independent variables easier. Let us briefly draw attention to a couple plots of our models outputs.

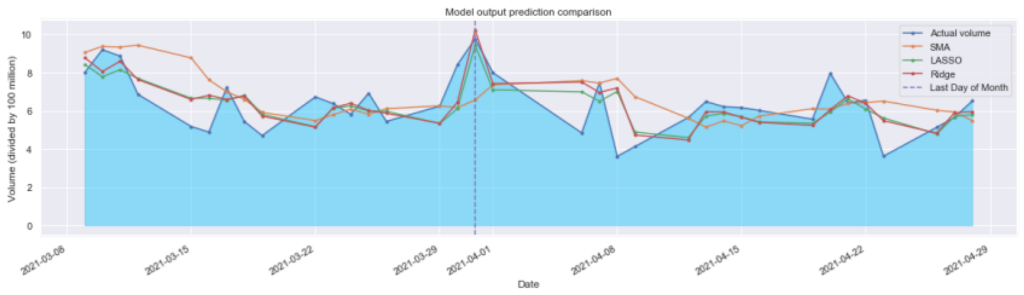

Figure 3.1: Model output comparison 1 - last Day of Month

The first of these here demonstrates how our model is able to pick up on increases in volume on the final day of the month (highlighted by the purple lines). The simple moving average has no mechanism to predict this.

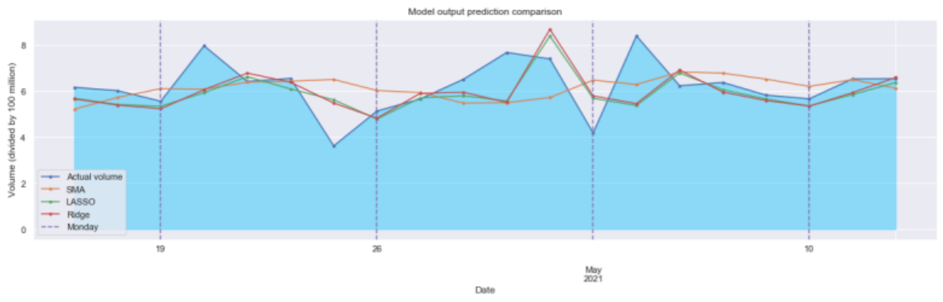

Figure 3.2: Model output comparison 1 - days of Week

The second here is slightly more subtle. As shown before (Figure 2.1), Mondays are generally lower traded than Wednesdays, and as a result the model created will very rarely predict Mondays to be higher volume than Wednesdays. As expected however, the moving average has no such mechanism, thus suffering in its predictions compared to our models.

We can additionally observe some of the main outputted coefficients from our ridge and our LASSO models below,

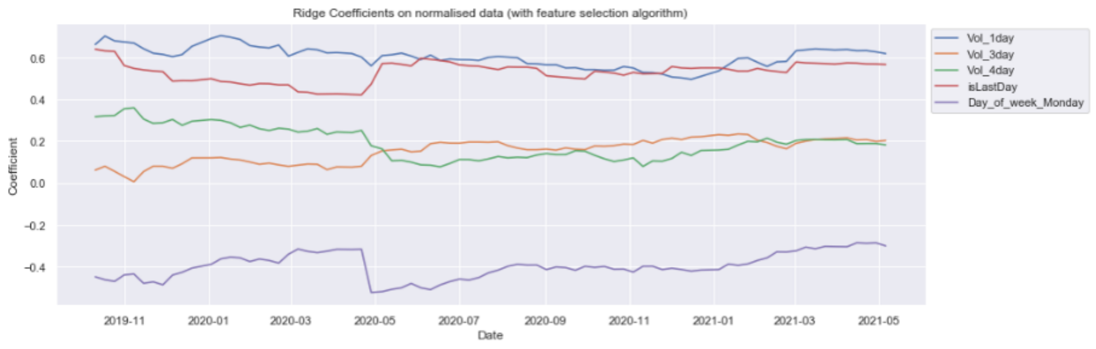

Figure 3.3: Ridge coefficients on German 7-11 Bonds

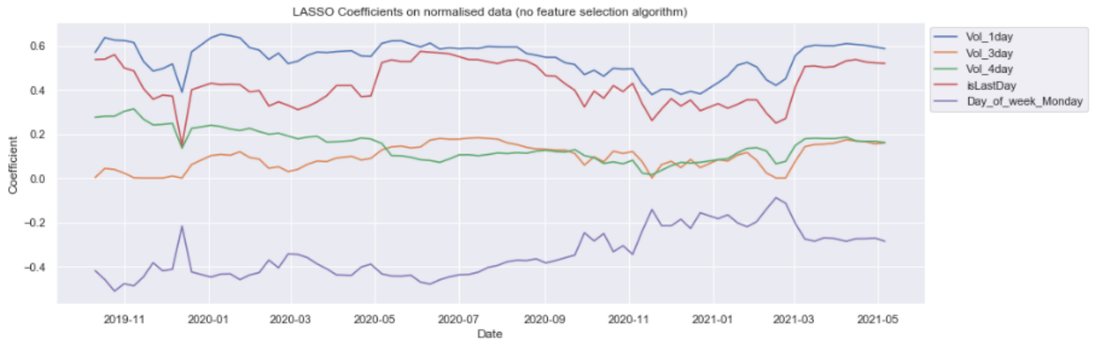

Figure 3.4: LASSO coefficients on German 7-11 Bonds

These coefficients are normalised so one can broadly interpret their magnitude as being related to their importance. As we would expect, the 1 day lagged volume and isLastDay have large magnitude positive coefficients, reaffirming their importance as suggested by the feature selection algorithm. Likewise, we note how Mondays have a large negative coefficient, due to volumes generally being lower on Mondays compared to other days of the week (Figure 2.1). Finally, we note how in the LASSO model, during the Christmas periods where volumes tend to crash, we see the coefficients shrink towards zero.

Chapter 4

Expanding analysis to larger bond universe

This section will focus on taking the work we have carried out on the German 7-11 Year Bond sector, and applying it to additional sectors. We will examine the similarities and difference between the results of our analysis for different sectors, as well as commenting on potential reasons for this. Furthermore, we will perform clustering analysis via PCA in order to determine what factors may cause certain bonds to act similar to one another. The features used in this section are the same as the ones used in the previous section, although once again we run a feature selection algorithm to remove unnecessary features.

4.1 PCA analysis results

As previously mentioned, bonds behave differently depending on what tenor they are in. For running a PCA, it makes more sense if we were able to construct a portfolio of bonds that are in a sense invariant in time. Thus, we group bonds, by taking the 5 bonds closest to 2, 5, 10 and 30 years to maturity, points that are associated with high liquidity. We aggregate each of these 5 bonds for every one of these tenors, running a PCA on the weekly volume delta ($\Delta V_t = V_t - V_{t-1}$). The results of this PCA are found below.

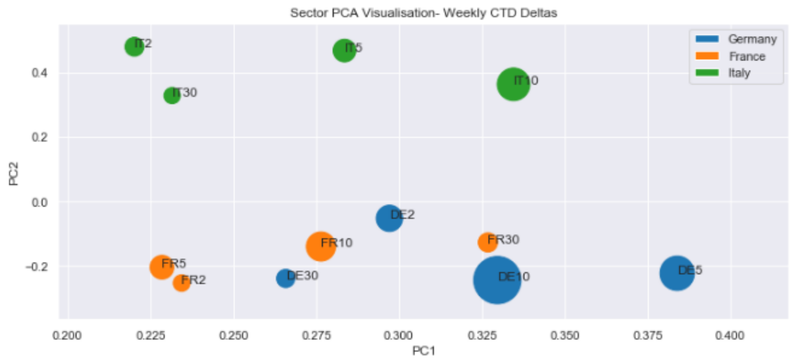

Figure 4.1: PCA results - weekly deltas

We can see that the loading coefficients for Italian bonds, which are colored green, are much higher on the 2nd principal component, compared to that of French and German bonds. This clustering is most likely due to the split we see between core countries in the soveriegn bond market (i.e. Germany/France), and periphery countries such as Italy. While there is only (as of the publication of this thesis), around a 35 basis point spread between 10 year French and German bonds, there is a 105 basis point spread between Italian and German bonds according to worldgovernmentbonds.com. This can be attributed to the lack of investor confidence toward Italian bonds, and this spread can be thought of as a price investors are willing to pay for a safer investment that reduces the risk of sovereign default.

4.2 Results

Below, we present the results in terms of R^2 for all of the sectors in our data set. We do not include MAE here, due to its lack of interpret ability between different sectors.

Table 4.1: All sectors out of sample R^2 - benchmark, OLS, ridge and LASSO

Sector	**5 Day SMA**	**OLS**	**Ridge**	**LASSO**
DE(0-2]	-0.001	0.099	0.142	0.119
DE(2-7]	0.019	0.227	0.207	0.227
DE(7-11]	0.172	0.238	0.248	0.250
DE(11-30]	0.428	0.539	0.553	0.550
FR(0-2]	-0.022	-0.040	0.010	0.082
FR(2-7]	-0.153	0.159	0.139	0.181
FR(7-11]	-0.123	0.109	0.111	0.082
FR(11-30]	0.250	0.264	0.300	0.274
FR(30-100]	-0.030	0.101	0.105	0.097
IT(0-2]	0.205	0.302	0.322	0.316
IT(2-7]	0.430	0.427	0.483	0.481
IT(7-11]	0.105	0.102	0.125	0.118
IT(11-30]	0.275	0.362	0.370	0.370
IT(30-100]	0.326	0.328	0.340	0.339

Table 4.2: All sectors out-of-sample R^2 - random forest, ARMA(1,1) and MLP

Sector	**Random Forest**	**ARMA(1,1)**	**MLP**
DE(0-2]	0.065	0.040	0.134
DE(2-7]	0.165	0.059	0.217
DE(7-11]	0.184	0.235	0.270
DE(11-30]	0.510	0.469	0.556
FR(0-2]	-0.093	0.032	0.069
FR(2-7]	0.090	0.051	0.151
FR(7-11]	0.049	0.030	0.120
FR(11-30]	0.238	0.274	0.256
FR(30-100]	0.073	0.095	0.073
IT(0-2]	0.260	0.268	0.327
IT(2-7]	0.378	0.420	0.440
IT(7-11]	0.090	0.125	0.135
IT(11-30]	0.333	0.370	0.372
IT(30-100]	0.298	0.353	0.249

To help us visualise these tables better, we can plot the R^2 amongst different models, averaged out across all sectors in a country.

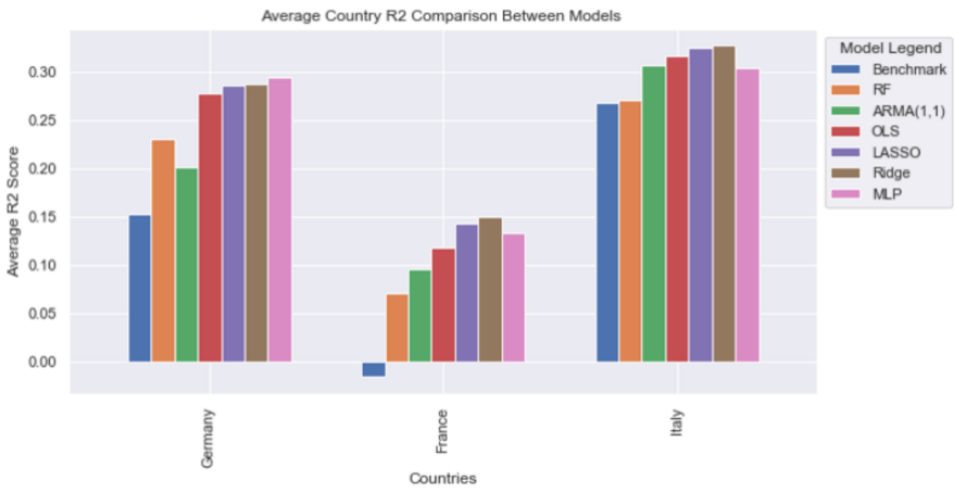

Figure 4.2: Average country R^2 comparison amongst all models

We note what sectors our models seem to do best on, and also what models perform the best. As with when we looked at only German 7-11 year bonds, we see that our models for the most part beat the benchmark in terms of R^2. The ARMA(1,1) model can be thought of as an alternative benchmark, that notes that the 1 day lagged volume is the most useful feature, and creates a model based upon this. We see the ARMA(1,1) model beating the benchmark on most occasions. If the ARMA(1,1) model were to outperform our other models too, we would have to conclude that our other features apart from 1 day lagged volume were not providing much utility. This is not the case however, as we see the regression models as well as the MLP beating it on most occasions. The ridge appears to be the best performing regression model, outperforming the OLS and LASSO in 10/12 sectors, however these improvements in performance are not particularly large. The MLP does do very well in some sectors, such as $DE(7-11]$, however there are certain sectors such as $IT(30-100]$ where it performs very poorly. This combined with the lack of interpratatability associated with the output of the MLP model, as well as the increased computational time, leads us to conclude that while the MLP model does give some improvements over the traditional regression models, the regression models are still better fit for the problem we have at hand.

Regarding the sectors our models perform best on, the main quantity worth examining is how much they are able to improve upon the benchmark. There is a trend here, with our models beating the benchmarks for French/German sectors, whilst struggling more for Italian sectors. As previously mentioned in this thesis, Italy is a periphery country, which means its market can be a lot more volatile and being able to beat a benchmark is a much harder exercise. This leads us to believe that perhaps for the Italian markets further research can be done to determine potential features that could be included, such as bond auction dates. While in terms of R^2, the results for French markets are not as good, our results do show that our models are able to beat the benchmark. The lower R^2 is most likely due to the number of data points of French bonds simply being lower than Germany and Italy, meaning there is more variance in our datasets.

Chapter 5

Intraday updates & modelling

5.1 Updating total daily prediction

This section explores using intraday modelling in two ways. Our first exercise is to create a regression with the same target as before, total volume traded in a given day, but observe how by including a couple hours of information, we can greatly improve our predictive power. That is to say, after 2 hours of trading, we should be able to accurately determine whether we are having a high volume, medium volume or low volume day.

To do this, we once more run a linear regression. We take a ridge regression, and use the same features as beforehand, however now we include an additional feature, volume traded up to hour n in a day. As before we used a nested train/test split, however before we would use our weights to predict 5 days ahead. Now we choose to only predict 1 day ahead. As we increase the value of n our predictions generally trend towards the true value for the volume traded in a given day.

The graph below demonstrates the utility that can be provided by an additional 2 hours of information.

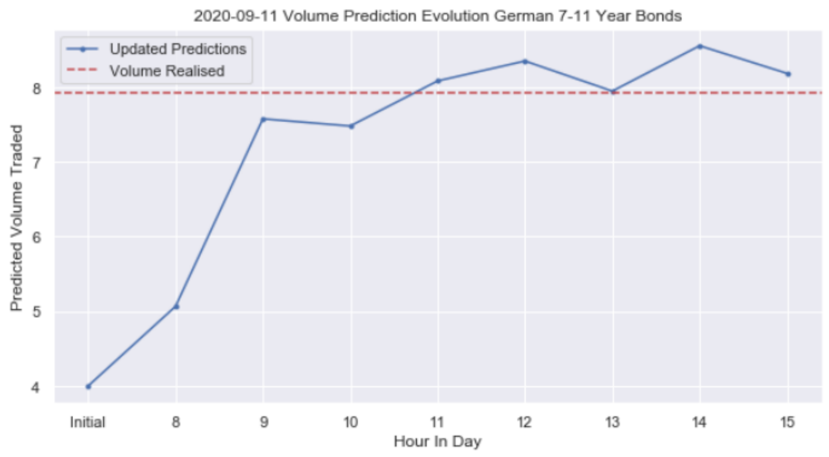

Figure 5.1: $2020-09-11$ prediction including volume traded up to n, with $n = 8, 9, \ldots 15$

We can see for this day $2020-09-11$ our initial volume prediction missed out on the spike in demand for that day. However, with 2 additional hours of information, the model is able to pick up on the fact that it is going to be a high volume day, and thus accordingly adjusts its prediction.

While this information is useful in evaluating the impact of observing values up to a given point of the day, we can carry out further quantitative and qualitative analysis. A kernal density estimation (KDE) can be used to estimate the probability desnity function of a random variable in a non-parametric way. By observing the KDE for the R^2 scores of our sectors at different times in the day, we can observe the impact of increase the value of n. Mathematically, we let $(x_1, x_2, \ldots, x_n)$ be identically distributed independent samples drawn from a univariate distribution with unknown density f at a given point x. One can estimate the shape of this density function, via a kernel density estimator, which is defined as follows,

$$\hat{f}_h(x) = \frac{1}{n}\sum_{i=1}^{n} K_h(x - x_i) = \frac{1}{nh}\sum_{i=1}^{n} K\left(\frac{x - x_i}{h}\right)$$

Here K is some non-negative function, and $h > 0$ is a paramater called the bandwith which controls the smoothness of the estimation. The kernal that is used throughout this section is the Gaussian kernal, defined as

$$K(x) = \frac{1}{\sqrt{2\pi}} e^{\frac{-x^2}{2\sigma^2}}$$

The results for this are given below.

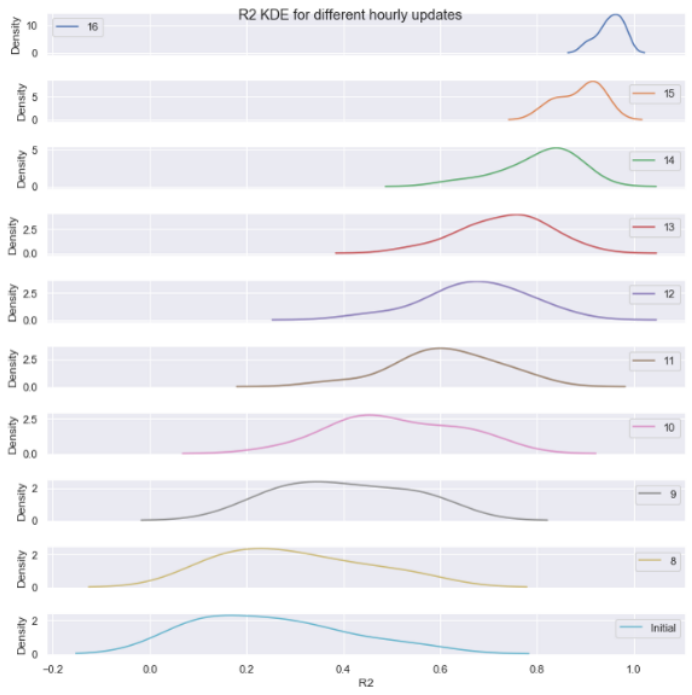

Figure 5.2: Hourly plot KDE evolution

There is a shift towards the right in terms of the values in the KDE (i.e. R^2 increases as we increase n). Additionally, the standard deviations of the R^2 values reduces, as the scores become closer to one another. By the time we are a $16:00$, the model is essentially just placing all of the weightings coefficient wise on the volume traded up to $16:00$, since this is very close to the total volume traded in the day. We note a large shift towards the right with the R^2 increasing when one increases the volume traded up to 9am, signifying that this hour long period between 8 and 9 is crucial.

We also observe the coefficients in our regression models as we change the value of n. For $n = 8$, we can see below that the volume traded up to 8 is clearly an important feature, as it has the highest magnitude coefficient, and clearly correlates positively with total volume traded on that day. If we observe the coefficients with $n = 12$, we see by this point the volume traded up to 12 dominates the regression, and the other coefficients are shrunk.

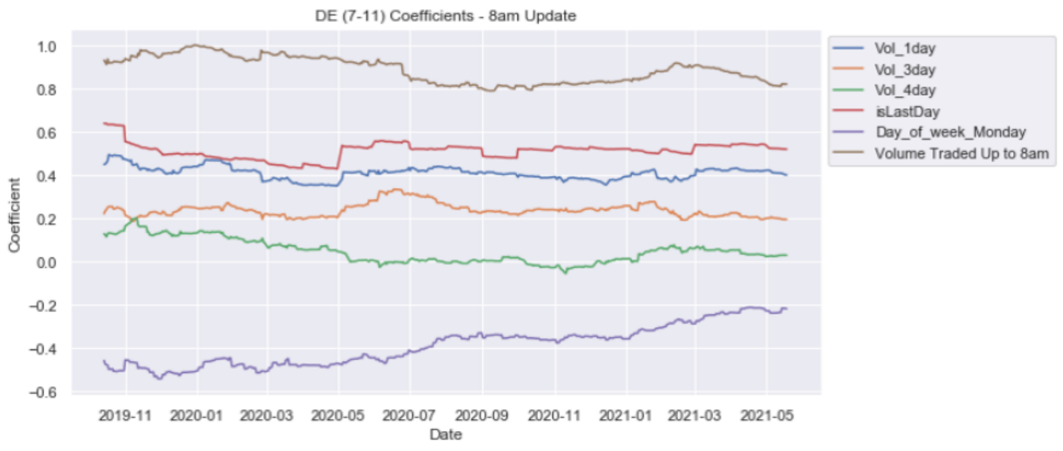

Figure 5.3: Coefficients for Germany 7-11 Bonds at 8am

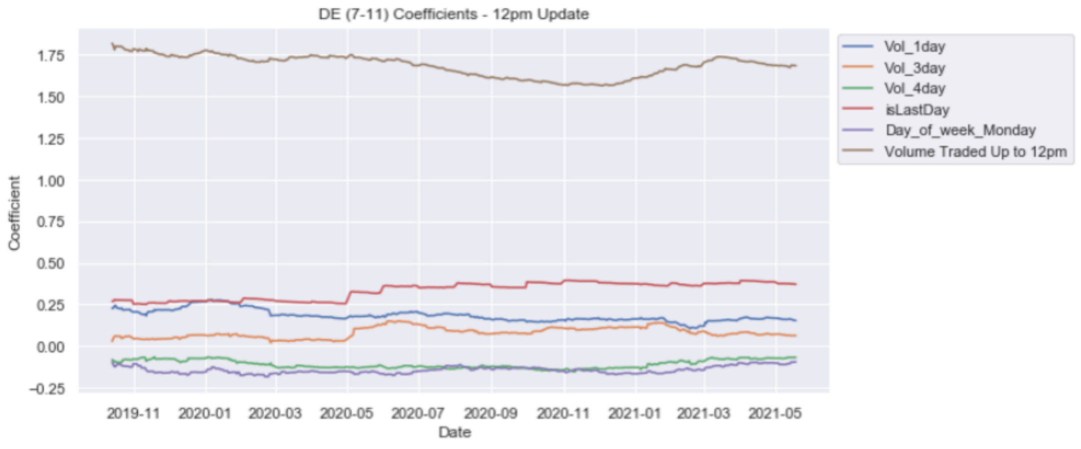

Figure 5.4: Coefficients for Germany 7-11 Bonds at 12pm

The main conclusion that can be reached from this task, from a business perspective, is that a couple hours of trading information can be useful in being able to determine the total volume to be traded on that day. What could be more useful is actually being able to predict the volume to be traded in the next hour or even the next 15/5 minutes, and this is the focus of our next section.

5.2 Intraday modelling

The purpose of this section is as follows. Given we have a prediction for the total volume to be traded on a given day, how can this be used to predict the volume to be traded in hourly, 15 minute or even 5 minutes buckets? One could use total volume predicted as a feature in a regression, but there are other techniques, which we will explore further in this section. Let us begin by defining a benchmark.

5.2.1 Benchmark model

Observe the following graph below, which shows the volume traded in given hours of the day for all German sectors, with confidence intervals.

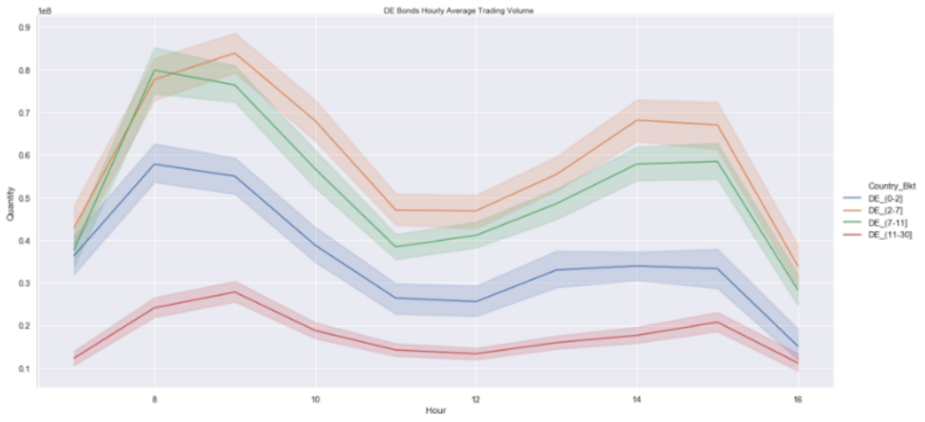

Figure 5.5: Hourly volume traded German sectors

We note that the volume starts off low generally, however trading peaks between 8 and 10 (note that Europe is an hour ahead). Then there is generally a decrease in volume towards lunchtime, before another smaller peak at around 1pm/2pm when the U.S. markets go live. Finally, as we approach the end of the trading day, volumes decrease. Given that there are these very clear patterns, these must be utilised in order to create a successful model. As a benchmark for the volume traded in the j^{th} time bucket of the day on day t, we define the following benchmark

$$BucketVolume_t^{(j)} = \frac{1}{22}\sum_{i=1}^{22} BucketVolume_{t-i}^{(j)}$$

This is a 1 month moving average of the volumes traded in the same time bucket.

5.2.2 AR(p) process on residuals

This model aims to utilise our total daily predicted volumes fully. Given that for a given day we have a daily volume prediction, we carry out the following,

(i) Take 22 days data (a month), and find the average proportion of daily volume to fall into each time bucket

(ii) Fit the aforementioned daily prediction model, and predict the total volume to be traded on a set of days for training data.

(iii) Multiply this figure by the average proportion expected to fall into each bucket.

(iv) Calculate the residual errors between the realised volume falling into each bucket and this initial estimate.

(v) Fit an $AR(p)$ model to this data and forecast the residuals for a given day.

(vi) Add these residuals onto the initial estimates to get an updated prediction of volumes to fall into each bucket.

The $AR(p)$ process is included since if we only used our initial estimate, we are relying on 1) our predicted total daily volume being accurate and 2) the intraday volume distribution being accurate. Our first step is to determine what order p we must choose for the autoregression process. We can do this via a partial autocorrelation (PACF) plot. If we were dealing with a $MA(q)$ process an ACF plot would suffice, since all lags greater than q are zero, this is not the case for an $AR(p)$ process. Due to the iterative nature of the model there is a chain of dependence. For example, considering an $AR(1)$ model defined as $X_t = \phi X_{t-1} + Z_t$, where $Z_t \sim WN(0, \sigma^2)$, we note,

$$\begin{aligned}
\gamma(2) &= Cov(X_t, X_{t-2}) \\
&= Cov(\phi X_{t-1} + Z_t, X_{t-2}) \\
&= Cov(\phi^2 X_{t-2} + \phi Z_{t-1} + Z_t, X_{t-2}) \\
&= \mathbb{E}[\phi^2 X_{t-2}^2 + \phi Z_{t-1} X_{t-2} + Z_t X_{t-2}] - \mathbb{E}[X_{t-2}]\mathbb{E}(\phi^2 X_{t-2} + \phi Z_{t-1} + Z_t)] \\
&= \phi^2 \mathbb{E}[X_{t-2}^2] \\
&= \phi^2 \gamma(0)
\end{aligned}$$

We see that there is a non-zero autocorrelation between 2 lags, since X_t depends on X_{t-2} through X_{t-1}. The core idea of a PACF function is to remove this dependence. More formally, one can prove, denoting ϕ_{kj} as the j^{th} coefficient in an $AR(k)$ process that the coefficients satisfy the following set of equations, (2).

$$\rho_j = \phi_{k1}\rho_{j-1} + \cdots + \phi_{k(k-1)}\rho_{j-k+1} + \phi_{kk}\rho_{j-k} \quad j = 1, 2, \ldots, k$$

which leads to the Yule-Walker equations, which can be written as,

$$\begin{bmatrix} 1 & \rho_1 & \rho_2 & \cdots & \rho_{k-1} \\ \rho_1 & 1 & \rho_1 & \cdots & \rho_{k-2} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \rho_{k-1} & \rho_{k-2} & \rho_{k-3} & \cdots & 1 \end{bmatrix} \begin{bmatrix} \phi_{k1} \\ \phi_{k2} \\ \vdots \\ \phi_{kk} \end{bmatrix} = \begin{bmatrix} \rho_1 \\ \rho_2 \\ \vdots \\ \rho_k \end{bmatrix}$$

One can solve these equations for $k = 1, 2, 3 \ldots$. The quantity ϕ_{kk} is referred to as the partial autocorrelation function, and is zero for $k > p$. It is the partial autocorrelation of the process X_t at lag k, since it is the partial correlation between X_t and X_{t-k} adjusted for the variables between, being $X_{t-1}, X_{t-2}, \ldots X_{t-k+1}$. We can now use the partial autocorrelation function plot on our time series to determine the order p. Observe as an example the PACF plot for German bond volumes residuals in 15 minute buckets.

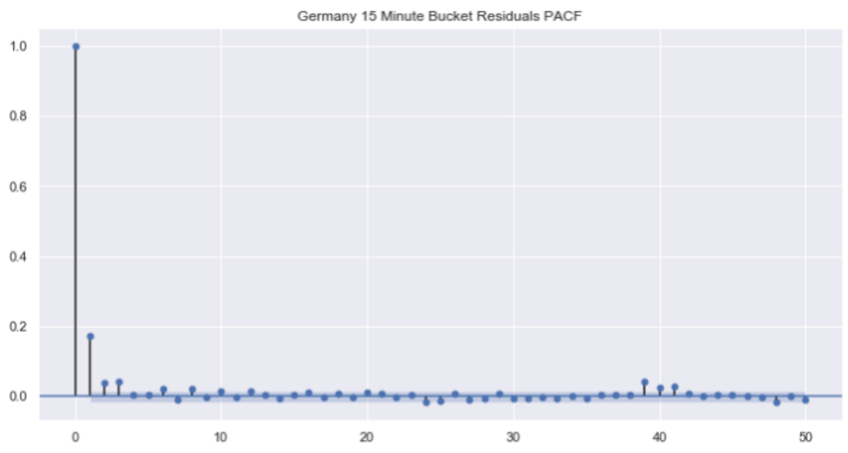

Figure 5.6: PACF plot Germany 15 minute buckets

After 3 lags, the lags stop being statistically significant, and thus an $AR(3)$ model appears to be appropriate for the time series. Different orders p were used for different time bucket sizes. We note a spike in the autocorrelations at around lag 40. Since we are looking here at 15 minute buckets, and we are considering 10 trading hours in a day, 40 lags from a given bucket in a day would correspond to the same bucket in the previous day. So, 1 day lagged bucket residual is in fact correlated, however an $AR(3)$ model would not be able to account for this. Therefore, we choose to consider one more model that would be able to pick up on these dependencies.

5.2.3 Regression Model

Our final model in the intraday modelling section is a regression model. In this model, due to the lack of utility provided by pricing features in interday modelling, we opt to omit them from the model. Thus, we can split our features up now into two broad categories, being volume based features and special days/times/booleans.

Volume Based Features

- Volume of previous bucket.
- Yesterday's bucket volume.
- 1 week and 1 month bucket volume average.
- Daily volume prediction

(Special) Days/Booleans

- Hour of the day
- Last day of month
- ECB meeting
- Futures rollover

The dataset for 5/15 minute predictions is heavily skewed, so in order to get normally distributed residuals, a log transformation is applied to all volume related quantities (4). Observe the coefficients from this model,

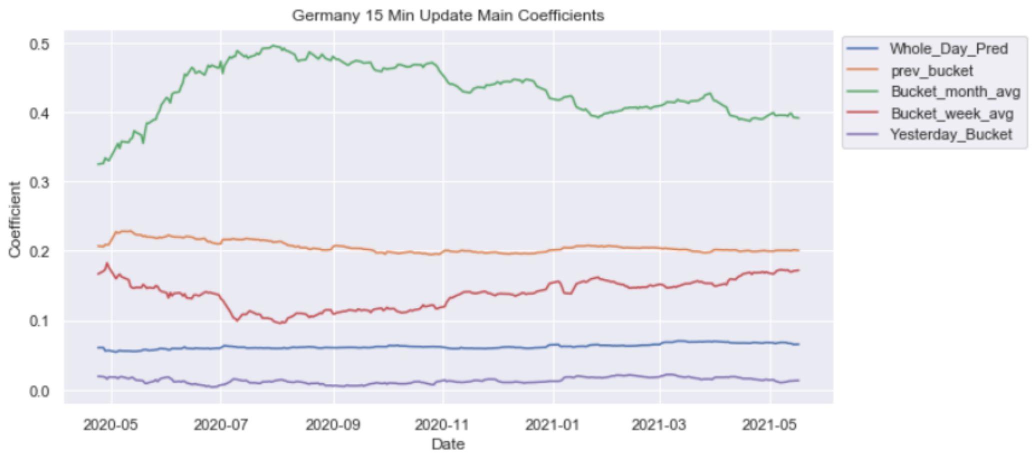

Figure 5.7: Germany 15 minute update non-hour coefficients

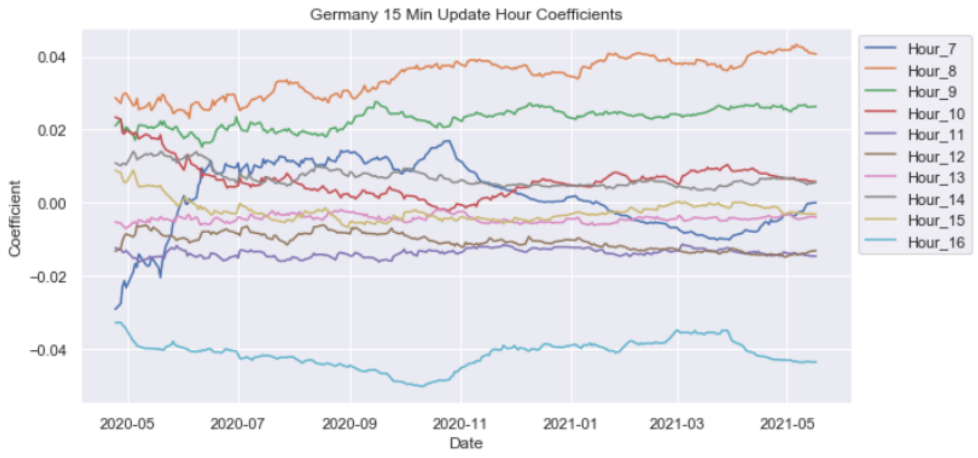

Figure 5.8: Germany 15 minute update hour coefficients

Given the high variance of volumes traded in buckets, we see that the bucket monthly average has a higher coefficient than yesterday's bucket value (they are on similar scales so comparable). This is interesting since in the daily case, the 1 day lagged volume was the most important feature, however the equivalent in intraday modelling is too noisy to be particularly useful. In the hour coefficients, we see a perfect match almost between it and Figure 5.5, the graph of hourly trading volumes in German sectors. Hours of the day where the volume is high have a high hour coefficient, while hours of the day where the volume is low have a low hour coefficient.

5.3 Intraday Results

We now summarise the overall performance of all of our models below.

Table 5.1: Sector hourly modelling R^2 results

Sector	**Benchmark**	**AR(p)**	**Ridge**
DE(0-2]	0.131	0.107	0.160
DE(2-7]	0.138	0.154	0.193
DE(7-11]	0.134	0.161	0.182
DE(11-30]	0.122	0.149	0.171
FR(0-2]	-0.030	-0.028	-0.012
FR(2-7]	0.008	0.043	0.047
FR(7-11]	0.018	0.034	0.048
FR(11-30]	0.045	0.065	0.104
FR(30-100]	0.021	0.054	0.066
IT(0-2]	0.042	0.064	0.071
IT(2-7]	0.142	0.162	0.195
IT(7-11]	0.080	0.091	0.112
IT(11-30]	0.067	0.089	0.111
IT(30-100]	0.038	0.074	0.102

Table 5.2: Country 15 minute bucket R^2 results

Country	**Benchmark**	**AR(p)**	**Ridge**
Germany	0.126	0.141	0.184
France	0.038	0.067	0.087
Italy	0.124	0.140	0.195

Table 5.3: Country 5 minute bucket R^2 results

Country	**Benchmark**	**AR(p)**	**Ridge**
Germany	0.047	0.055	0.134
France	-0.004	0.008	0.027
Italy	0.037	0.045	0.086

Due to higher variance and noise, our predictive power falls as we go from 15 minute predictions to 5 minute predictions. We cannot fairly compare the drop in R^2 between hourly and minutely buckets however, since we shift our target from sector volumes to country volumes. We note both our autoregression model and our ridge beat the benchmark, however it appears the ridge offers the best out-of-sample performance. This could be attributed to the fact that the autoregressive model would not account for similar residual sizes for the same buckets at different sizes of the day, given that p was consistently smaller than the number of buckets in a day. When it comes to liquidity and amount of RFQs, France has a far smaller amount than Germany and Italy, which leads to its data set having a higher variance and being somewhat more difficult to predict, resulting in it having the lower out-of-sample R^2. The models perform best on Germany, perhaps due to it lacking the aforementioned liquidity problem that France has, yet being a far less volatile market than Italy.

Chapter 6

Conclusion

In this section, we summarise the main conclusions that can be drawn from our thesis, and reiterate the reasoning behind this research project to evaluate whether the models created can be put into production to provide genuine use to Deutsche Bank.

Our original goal was to model daily total volumes amongst German, French and Italian bond sectors. We saw successful results doing this, with an average R^2 increase from our best models compared to the benchmark of around 13% for Germany, 17% for France and 9% for Italy. As this is all out-of-sample, we can conclude this to be a successful statistic. This model would be useful to traders to give them an idea of how long they may have to hold onto certain positions for.

In terms of which model performed the best, the ridge, LASSO and MLP all appeared to perform well. The ridge model did have the highest R^2 on average, but not by a statistically significant amount compared to the LASSO. This increase in R^2 may be due to the ridge having our aforementioned feature selection algorithm applied to it, while our LASSO did not. When it comes to putting a model into production, features which did not seem to provide any sort of clear signal, such as our 3 pricing features, would not be included, so there is not too much separating these two models. The MLP, while performing very well overall, does have two major shortcomings, being the very large computation time associated with creating and running a neural network, and additionally the lack of interpretability that comes with it.

In terms of the most important features associated with predicting daily volumes, it appears that the most important one, according to our feature selection algorithm as well as (normalised) coefficient size is 1 day lagged volume. This makes sense, as high activity from one party generally would lead to a counter-reaction from another party, and this effect could last over multiple days. As previously mentioned, pricing features did not improve out-of-sample R^2 for any sectors, and it appears that the most successful linear model consists of taking linear combinations of previously observed volumes, along with some long term averages, and finally including weighted indicator functions for certain market events (last day of month, rollover etc...)

Regarding intraday modelling, again we saw in general a success, with both our autoregressive model and our regression model beating our benchmark. While we are able to predict sectors on an hourly level, we must switch to predicting entire countries at a 5/15 minute level, due to the high variance of the dataset. As previously mentioned, here it appears that compared to modelling daily volumes, long term lagged averages provide a better indicator of volumes to be traded in a short time bucket, rather than just simply take the previously observed bucket value as a feature.

There are still further areas of research in volume prediction that one could potentially look into. As mentioned at the start of the thesis, predicting individual ISINs was beyond the scope of this project, however with further advancements in machine learning constantly on the horizon, perhaps this could eventually be done successfully. Also, while our target here was total volume traded, further research could be on changing the target to signed volume by modelling both the volume of buy orders and sell orders, as this would give more practical usage to traders. We note that our random forest model did not perform well, however an adaptation of tree models could still potentially be used. In L. Arous' paper on predicting corporate bond volumes (1), a multiple additive regression trees (MART) model was developed, as well as a support vector regression model, both of which performed better than an OLS. As a bagging approach did not perform well in this thesis, the boosting approach in MART could be an area of further research (5).

To conclude, our most successful model appears to be a ridge model, that comfortably beats its benchmark in both an interday and intraday setting. This model would provide utility to traders, and also would be computationally efficient, meaning it could be deployed in a high frequency space.

Bibliography

[1] Arous, L. B. (2012). Predicting the daily liquidity of corporate bonds - Stanford university.

[2] Box, G. and Jenkins, G. M. (1976). *Time Series Analysis: Forecasting and Control*, page 64. Holden-Day.

[3] Breiman, L. (1996). Bagging predictors. *Machine Learning*, 24(2):123–140.

[4] Curran-Everett, D. (2018). Explorations in statistics: the log transformation. *Adv Physiol Educ*, 42(2):343–347.

[5] Friedman, J. H. (2001). Greedy function approximation: A gradient boosting machine. *The Annals of Statistics*, 29(5):1189 – 1232.

[6] Hastie, T., Tibshirani, R., and Friedman, J. (2001). *The Elements of Statistical Learning*. Springer Series in Statistics. Springer New York Inc., New York, NY, USA.

[7] Hoerl, A. E. and Kennard, R. W. (1970). Ridge regression: Biased estimation for nonorthogonal problems. *Technometrics*, 12(1):55–67.

[8] Keskar, N. S., Mudigere, D., Nocedal, J., Smelyanskiy, M., and Tang, P. T. P. (2016). On large-batch training for deep learning: Generalization gap and sharp minima. *CoRR*, abs/1609.04836.

[9] Kingma, D. P. and Ba, J. (2017). Adam: A method for stochastic optimization.

[10] Louppe, G. (2015). Understanding random forests: From theory to practice. pages 26–27.

[11] Mcculloch, W. and Pitts, W. (1943). A logical calculus of ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics*, 5:127–147.

[12] Moore, A. W. (2001). Decision trees - school of computer science carnegie mellon university.

[MTS] MTS. Yield spreads. https://www.mtsmarkets.com/european-bond-spreads.

[14] Paule-Vianez, J., Orden-Cruz, C., and Escamilla-Solano, S. (2021). Influence of covid-induced fear on sovereign bond yield. *Economic Research-Ekonomska Istraživanja*, 0(0):1–18.

[15] Pearson, K. (1901). Liii. on lines and planes of closest fit to systems of points in space. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, 2(11):559–572.

[16] Tibshirani, R. (1996). Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society (Series B)*, 58:267–288.

[Tradeweb] Tradeweb. Government bonds. https://www.tradeweb.com/our-markets/institutional/rates/government-bonds/.

[18] Xie, P. X. (2019). CS 273P machine learning and data mining.

HATAMIEH_KIAN_01340679

GRADEMARK REPORT

FINAL GRADE

/0

GENERAL COMMENTS

Instructor

PAGE 1

PAGE 2

PAGE 3

PAGE 4

PAGE 5

PAGE 6

PAGE 7

PAGE 8

PAGE 9

PAGE 10

PAGE 11

PAGE 12

PAGE 13

PAGE 14

PAGE 15

PAGE 16

PAGE 17

PAGE 18

PAGE 19

PAGE 20

PAGE 21

PAGE 22

PAGE 23

PAGE 24

PAGE 25

PAGE 26

PAGE 27

PAGE 28

PAGE 29

PAGE 30

PAGE 31

PAGE 32

PAGE 33

PAGE 34

PAGE 35

PAGE 36

PAGE 37

PAGE 38

PAGE 39

PAGE 40

PAGE 41

PAGE 42

PAGE 43

PAGE 44

PAGE 45

PAGE 46

PAGE 47

PAGE 48

PAGE 49