UNIVERSITY COLLEGE LONDON

MASTERS THESIS FOR MRES IN WEB SCIENCE AND BIG DATA ANALYTICS

Investigating Predictability of e-FX Option RFQs Using Bayesian and Classical Machine Learning Techniques

---

Author:	Shahrooz ALIABADI-ZADEH
UCL Supervisor:	Prof. Philip TRELEAVEN
BNP Paribas Supervisor:	Dr. Graham BARRETT

September 2019

Abstract

This thesis is an investigation into the predictability of currency (FX) option derivative requests for quotes (RFQs). The source of the RFQs was a single-dealer electronic trading platform from a sell-side financial institution. The investigation researched two different philosophical approaches to machine learning, namely the Bayesian and Classical techniques, by evaluating a series of supervised and unsupervised models that straddled both philosophies. The Bayesian hidden Markov model (HMM) was used as the unsupervised learning model which was compared to the Classical and Bayesian supervised models. All models had some degree of success in predicting patterns in the data. In addition, the Bayesian hidden Markov model was able to infer the hidden underlying RFQ market regime (e.g. increasing, decreasing or level RFQ counts).

Background research suggested that the ability to predict RFQs along with knowledge of the hidden underlying state of the market regime could potentially be pivotal for sales and trading outfits of investment banks. In an industry where volumes are declining and deals are lost by the thinnest of margins, this level of insight will allow banks to better configure their offerings, price quotes more competitively and provide more targeted sales interventions. All of these will result in revenue growth either through an increase in the probability of conversion of a quote into an actual trade or through an increase in the average size of the trade.

This thesis consists of the following three experiments:

1. Predicting RFQs using supervised learning models. This first experiment investigates the ability of three different models to predict RFQs. A feed-forward neural network regressor, a ridge regression and a Bayesian ridge regression model were trained, evaluated and compared against each other.

2. Predicting RFQs using the unsupervised hidden Markov model. The objective of the second experiment was to create and fit a hidden Markov model to predict RFQs using emission distribution of the most likely hidden state.

3. Inferring RFQ hidden states using the unsupervised hidden Markov model. This final experiment explored the hidden Markov model much deeper to give insight into the underlying hidden process by using the visible data to predict the most likely hidden market regime of the RFQ data.

This research was conducted in collaboration with Dr. Graham Barrett of BNP Paribas and it presents the following contributions to research:

1. Bayesian versus Classical techniques – This study is believed to be the first academic study to predict and find patterns in electronic FX option

RFQs using both Bayesian and Classical methods. Further, the methodology could be readily applied to many other order processes with a time-series repeatable order aspect.

2. Application of the hidden Markov model to a new domain - the RFQ data was transformed into a Markovian state fit for consumption by the hidden Markov model using a 'differences' method. This unsupervised Bayesian model was used to investigate not only the predictability of RFQs but also to infer the hidden ('latent') states of the underlying visible RFQ data using the Baum-Welch expectation maximisation algorithm. As a result, at any point in time the underlying RFQ regime could be determined.

3. Provision of simulated RFQ data - due to data protection and privacy laws, RFQ data is not publicly available thus limiting research opportunities. To the best of our knowledge this is the only example of simulated FX option RFQs constructed, leveraging intimate knowledge of the FX options market. The dataset provides production quality RFQ data for EUR-USD vanilla European options centred around London trading hours which could be utilised for future machine learning research in this area. This data has been pre-processed for both supervised and unsupervised learning methods.

For accompanying data files and code see:
https://github.com/shahroozaz/MastersThesis.

Contents

1 Introduction

The aim of this chapter is to present an overview of this thesis by first presenting the motivation for this research along with a robust business justification. A brief introduction to e-FX options and platforms is provided based on intimate industry knowledge. The chapter also outlines the research objectives, the overall framework used for learning the RFQs along with an explanation of each component in the framework. The model selection component presents a two-by-two matrix used as a foundation for this research. Finally the chapter introduces the experiments conducted along with the structure of this thesis.

1.1 Motivation for this research

While machine learning is increasingly being used to support the customer purchase process in financial institutions, the use of predictive analytics is still relatively limited. For example, one of the more sophisticated applications is the robo-advisor platforms used across a number of large financial institutions such as HSBC, Fidelity, RBS and Santander. These platforms provide automated investment advice to institutional and retail customers based on their profiles. However, they do not have the predictive power to enable targeted sales actions to close the deals. This thesis explores advanced techniques for time-series predictive analytics to take these capabilities to the next level.

This work was in part inspired by the commercial opportunity in FX option sales, and in part by the philosophical debate between the Classicists and the Bayesians. This study will investigate both techniques for predicting customer quote requests as well as identifying the underlying hidden market dynamics behind it. The ability to predict the next quote request will enable banks to a) configure an offering that better meets the clients' needs at that particular time, thus increasing sales conversion rates and b) cross-sell or upsell other suitable products, thus increasing average order size. Further, the ability to predict the market regime within which the request is occurring (i.e., booming, flat or declining market demand), will help banks optimise pricing to achieve the sale at the best margins.

The business justification has been validated through discussions with professionals in the industry both on the sales and trading sides. Also, it should be noted that the techniques discussed here are not constrained to the banking industry and will apply in businesses in general where there is a time-series repeatable customer order aspect. Whilst these techniques have been utilised to a greater degree in the business-to-consumer retail space (e.g., Amazon, Booking.com, eBay), there have been fewer applications in the business-to-business space. The proof-of-concept in Finance could open up doors to many other B2B markets.

1.2 Brief introduction to e-FX

1.2.1 e-FX options

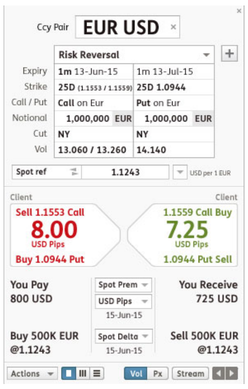
Electronic foreign exchange (e-FX) options are a type of financial instrument known as derivatives. They are not fundamentally different to FX options (also called currency options) except that they are priced and traded electronically via digital platforms rather than by voice. Other commonly traded e-FX products are FX spots, forwards, swaps as well as non-deliverable instruments. FX Options derivatives give the owner (purchaser) the right but not the obligation to buy (‘call option‘) or sell (’put option’) a specified amount of one currency in exchange for another (for example EUR-USD) at a pre-agreed exchange rate or price (known as the strike price) on a specific date. The FX market is one of the largest and most liquid of options with trading volumes in trillions, which is why it was chosen for this research as a source of data. There are many different flavours of FX Options, which along with their pricing and risk management are beyond the scope of this thesis.

1.2.2 e-FX trading platforms and RFQs

The foreign exchange market and in particular the FX option instrument type in the dealer-to-client market are traded electronically on single-dealer platforms (SDPs) or multi-dealer platforms (MDPs), both of which emerged in the late 1990s. SDPs are proprietary, customisable trading platforms offered by a single dealer (e.g. BNP Paribas) to its clients. These are often integrated with other internal systems (for example, derivatives pricing and credit checking, wealth management and regulatory reporting). MDPs, on the other hand, provide a platform that allows market investors to request quotes to trade from a number of dealers simultaneously [7]. Both SDPs and

MDPs rely on 'request for quote' (RFQ) trading protocols. An RFQ is a protocol by which the users of SDPs and MDPs (investors) send in the specifications of an order to request a quote. Typically a 2-way price (i.e. an option to BUY or SELL the instrument) is made available to the user in response to their request based on available liquidity, risk limits and credit worthiness etc.

SDPs such as Neo at UBS (Fig 1.1a) and Cortex FX at BNP Paribas (Fig 1.1b) are sophisticated enough to offer seamless no-obligation 2-way pricing (a quotation) as well as click-and-trade ability for FX option vanillas, exotics and multi-leg structures.



Ccy Pair EUR USD

Risk Reversal

Expiry	1m 13-Jun-15	1m 13-Jul-15
Strike	25D (1.1553 / 1.1559)	25D 1.0944
Call / Put	Call on Eur	Put on Eur
Notional	1,000,000 EUR	1,000,000 EUR
Cut	NY	NY
Vol	13.060 / 13.260	14.140

Spot ref 1.1243 USD per 1 EUR

Client	Client
Sell 1.1553 Call	1.1559 Call Buy
8.00	7.25
USD Pips	USD Pips
Buy 1.0944 Put	1.0944 Put Sell

You Pay 800 USD

Spot Prem · USD Pips · 15-Jun-15

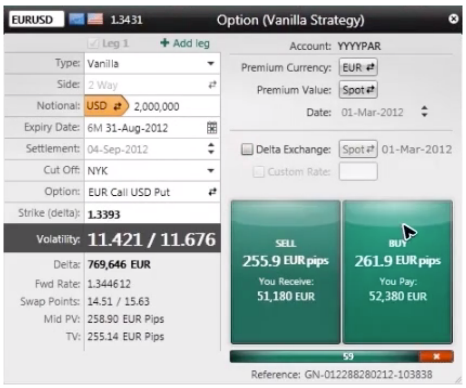
You Receive 725 USD

Buy 500K EUR @1.1243

Spot Delta · 15-Jun-15

Sell 500K EUR @1.1243

(a) Neo, UBS [37]



EURUSD 1.3431 Option (Vanilla Strategy)

Type:	Vanilla
Side:	2 Way
Notional:	USD 2,000,000
Expiry Date:	6M 31-Aug-2012
Settlement:	04-Sep-2012
Cut Off:	NYK
Option:	EUR Call USD Put
Strike (delta):	1.3393
Volatility:	11.421 / 11.676
Delta:	769,646 EUR
Fwd Rate:	1.344612
Swap Points:	14.51 / 15.63
Mid PV:	258.90 EUR Pips
TV:	255.14 EUR Pips

Account:	YYYYPAR
Premium Currency:	EUR
Premium Value:	Spot
Date:	01-Mar-2012
Delta Exchange:	Spot 01-Mar-2012
Custom Rate:	

SELL	BUY
255.9 EUR pips	261.9 EUR pips
You Receive:	You Pay:
51,180 EUR	52,380 EUR

59

Reference: GN-01228828O212-103838

(b) Cortex FX, BNP Paribas [8]

FIGURE 1.1: Examples of SDPs

1.3 Research objectives

The objective of this thesis is to investigate the predictability of e-FX options using both Classical and Bayesian methods. To achieve this goal, a set of experiments were carried out using both supervised and unsupervised learning models. Machine learning typically involves the sequential steps highlighted in Figure 1.2. This thesis followed the approach in Figure 1.2 for each of the chosen models with slight variations for supervised versus unsupervised models, highlighted in Section 3.2. A brief introduction to each step of the process is described in this section.

1.3.1 RFQ data generation

Due to data protection and privacy laws, banks are unable to make available client RFQ data for academic research purposes. Hence, for the purpose of

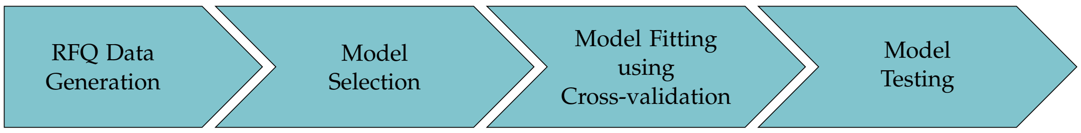


FIGURE 1.2: Machine learning steps applied in this thesis

this thesis, production-like RFQ data was generated in the form of a time-series i.e. a sequence of data ordered using the dimension of time represented in the form below:

$$\mathrm{RFQ}_{a:b} = \{\mathrm{RFQ}_a, \mathrm{RFQ}_{a+1}, \dots, \mathrm{RFQ}_b\} \tag{1.1}$$

A time-series format is a necessity in order to predict probability of RFQs in the next hour or indeed to predict the time intervals between each RFQ. RFQs were generated based on the EUR-USD currency pair and 'European vanilla' FX option type, along with a selection of standard option attributes (for example: strike, notional, premium, etc.), which typically form part of the RFQ. The RFQs were captured over a period of 1 month, which corresponds to 44,687 RFQ entries.

Once the RFQ data was generated, a 'binning' approach for representation of the RFQ time-series was chosen and applied to the data set prior to consumption by the models. The 'binned' data was then split and organised using the methodology depicted in Figure 3.5. Data wrangling was performed to transform the binned RFQ data into shapes suitable for consumption by the selected supervised and unsupervised models; this is discussed further in Section 3.2.

1.3.2 Model selection

In order to investigate the predictability of RFQs, this thesis evaluated the machine learning models laid out in the two-by-two framework in Figure 1.3.

In the complex 'unsupervised' column category in Figure 1.3, this thesis chose to evaluate only Bayesian unsupervised models. The reason is that they allow you to reason in a more sophisticated manner, for example reasoning about the hidden ('latent') states, discussed in detail as part of experiment 3 in Section 4.8.

Figure 1.3 splits the chosen machine learning models into Bayesian and

	Simple “Supervised”	**Complex** “Unsupervised”
Classical	• Multi-layer Perceptron Regresser (MLPRegressor, a neural network) • Ridge Regression (least-squares with L2 regularisation)	K-means Clustering
Bayesian	Bayesian Ridge Regression	Hidden Markov Model

FIGURE 1.3: Two-by-two model selection framework. Note: K-means is provided for illustration purposes only and will not be investigated as it is not as sophisticated as the HMM

Classical on the vertical axis and supervised and unsupervised on the horizontal axis. Bayesian and Classical models are based on two different philosophies in machine learning. Bayesians take a probabilistic approach based on a subjective prior belief and provide a solution in the form of a probability distribution whereas the Classicists provide a single answer based on the best fit to the data. They are both discussed in much greater detail in section entitled Classical versus Bayesian Machine Learning; refer to Section 2.2.

In comparison to unsupervised learning models, the task of supervised models is more directed in their interpretation of the given data and potentially easier to fit. In supervised learning, the learning is directed by the examples given (hence labelled ‘Simple’ in Figure 1.3). In comparison to supervised models, which find and predict patterns in the data, unsupervised models such as the hidden Markov models (HMMs) can find phenomena in the data, such as clusters, without the need for a labelled data set. Although unsupervised HMMs can also predict and find patterns based on the visible data, as they are able to infer hidden (‘latent’) states which can only be directly measured by observing the corresponding visible variables which they affect, for example: RFQ activity.

A hidden state can represent the underlying market dynamics and at any point in time, this could be a discrete variable in one of three underlying states (for example: increasing, decreasing or level) which will result in a probability that currently the market is in one of the three identified regimes. It is this property of the HMM which makes it much more insightful and more complex to deal with as a machine learning tool. There is further discussion in the HMM model set-up section.

1.3.3 Model fitting and testing

The RFQ data set was split into 90% training data and 10% test data. The 10% test data set was set aside to evaluate the final official unbiased performance of the best model. The training set was then split further into a validation set and a training set, depicted in Figure 1.4.

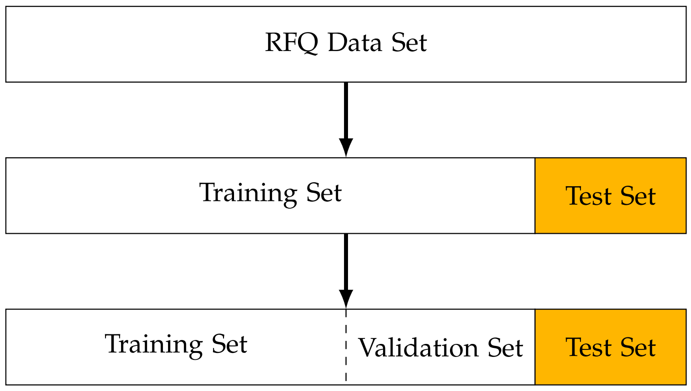


FIGURE 1.4: Train-validation-test data split

The models were evaluated on the training and validation portion of the data using 'cross-validation', a process where hyperparameters are fine tuned. Hyperparameters are configurations external to the model (unlike model parameters) whose values cannot be learned from the data during the training process and are typically specified prior to training by machine learning practitioners. An example of a hyperparameter would be the learning rate for a neural network regression model (e.g. MLPRegressor). The models were then retrained on the training set using the best hyperparameter thus returning the best model. The best model (with the optimal hyperparameters) is then run on the test data to see how well it generalises on the hitherto unseen test data. A detailed process map is provided in Section 3.2.1.

1.4 Research experiments

The research experiments for this thesis are presented under three categories in Section 4, in order of increasing complexity. They are designed to investigate predictability of RFQs using supervised and unsupervised models in both a Classical and Bayesian setting. **Experiments 1 and 2** attempt to predict RFQs using the selected supervised and unsupervised hidden Markov model respectively. More sophisticated insight is sought in **Experiment 3** by inferring the hidden ('latent') states of the RFQ market dynamics. This can be very insightful as it provides the ability to know at any point in time which type of regime the underlying market is operating in, for example: increasing, decreasing or level.

1.5 Thesis structure

The structure of this thesis is organised as follows:

Chapter 1 – Introduction. This chapter begins with providing the reader with the motivation for conducting this research. It highlights the validated business justification for this thesis as well as explaining why the timing is right. After which a very brief introduction to e-FX options, trading platforms and RFQs is provided based on intimate knowledge of financial markets. The chapter then highlights the research objectives and the overall methodology, which includes RFQ data generation, model selection, model fitting and testing. Finally the chapter introduces the three experiments conducted in this thesis which cover prediction of RFQs as well as inferring the hidden states using the hidden Markov model.

Chapter 2 – Background and Literature Review. The aim of this chapter is to describe in detail key background concepts accompanied by relevant literature. It first presents the key role which machine learning plays in Finance and the expected impact it will have on the financial services industry. Thereafter a comprehensive review of Classical versus Bayesian philosophies is provided, which includes a description of Bayes' Theorem.

This chapter then provides detailed coverage of the hidden Markov model (HMM) with precise definitions of relevant equations. After the HMM background and applications, belief networks are described, leading into Markov chains and subsequently the HMM. The iterative expectation maximisation (EM) algorithm is also covered in this chapter, both in the general form and specifically the Baum-Welch EM algorithm which is used by the HMM for learning. Next, artificial neural networks are presented with key features and applications. Thereafter the focus is aimed at feed-forward neural networks which are relevant to this thesis. The chapter also covers their architecture, key components, as well as the back-propagation and stochastic gradient descent, necessary for learning in a neural network.

Chapter 3 – RFQ Data Set. RFQ data generation of EUR-USD vanilla European options along with descriptive statistics is provided in this chapter. The 'binning' methodology for processing the RFQ data is presented and justified. The chapter also provides the reader with insight into the RFQ data using some key plots. Towards the end, the chapter describes the steps used to split the RFQ data for cross-validation, training and testing. The reader is also presented with data wrangling steps used to transform the RFQ data-set into a form suitable for consumption by each supervised and unsupervised model.

Chapter 4 – Experiments. To begin with, this chapter presents the overall model selection framework used for this thesis and the subsequent three experiments. Each experiment follows a similar format of: Introduction to the selected experiment and model, Model implementation, Results and discussion and finally a Conclusion. The experiments get progressively more sophisticated. **Experiment 1** begins with a brief introduction, the three selected supervised learning models are implemented, trained and evaluated in their ability to predict RFQs. **Experiment 2** creates and fits a hidden Markov model to make RFQ predictions using the most likely hidden state. This is demonstrated on both 1 (count) and 2 (count and notional) dimensional visible variables. The results for Experiment 2 are compared with those in Experiment 1. Finally, **Experiment 3** makes use of the hidden Markov model to make inferences about the underlying hidden ('latent') state of the RFQ market dynamics (e.g increasing, decreasing or level). The Viterbi algorithm is used to infer the most likely hidden path - the learning for this is done using the Baum-Welch iterative algorithm. This experiment also explains why a 'differences' methodology was adopted to transform the data into a Markovian state suitable for consumption by the hidden Markov model.

Chapter 5 and 6 – Conclusion and Further Work. The final 2 chapters provide the overall conclusion of this thesis as well as suggestions for further work.

2 Background and Literature Review

This chapter presents fundamental background information and concepts used in this thesis along with relevant literature review. The reader is presented with the role of machine learning in finance and the need for it. The chapter then compares and contrasts Classical versus Bayesian machine learning. A detailed introduction to the hidden Markov model is provided along with selected applications and learning methods utilising the expectation-maximisation algorithm. Finally, artificial neural networks are introduced along with some applications. Feed-forward neural networks are discussed along with their architecture, layout and learning method.

2.1 Machine learning in finance

Finance falls in the optimal spot for machine learning. Compared to many other sectors, this is a sector where transactional data is plentiful, where data is systematically captured (particularly in recent years due to increased regulatory requirements) and where there are many complex interrelationships across multiple variables that lend themselves to more sophisticated analytical techniques. And yet, machine learning is in its relative infancy in this sector.

In a recent research study by MIT Sloane Management Review and the Boston Consulting Group [31], more than 3,000 senior business executives, managers and analysts from different sectors were surveyed across the globe on the expected impact of AI on their sector. Whilst all sectors have high expectations of AI, the Finance sector has the biggest gap between current and future states – i.e., the sector with the greatest potential relative to where they are today (refer to Figure 2.1). This has likely been driven by the historical emphasis of banks on personal relationships to drive sales, banks having legacy infrastructure/practices, and banks being generally slow to adopt some of the new technologies.

Still, digitisation has been evolving the sector, laying the ground for machine learning. Over the past two decades, the front offices (sales and trading functions) of banks have seen extensive digital transformation. Digital channels have become one of the primary ways to access a bank's offerings and people – either through banks or third-party platforms to view and request price quotes or to execute transactions digitally.

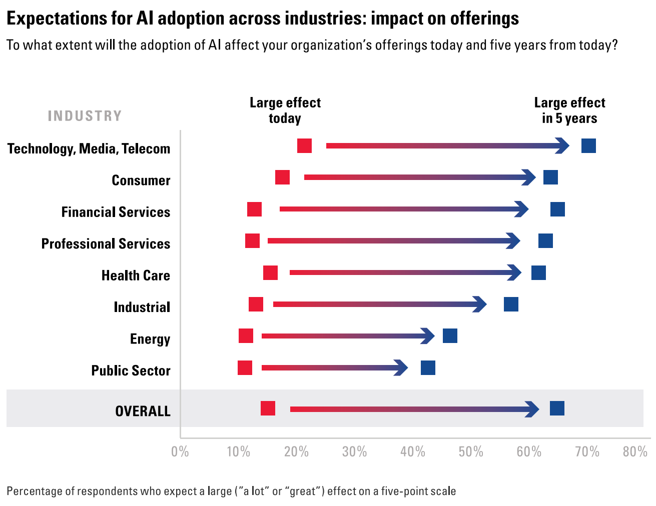


FIGURE 2.1: Expectations for AI's effect on businesses' offerings in 5 years time [31]

Now, the time is right for banks to take digitisation to the next level – to leverage machine learning to bring differentiated insight and intelligence that can generate enhanced revenues. In Fixed income commodities and currencies (FICC), which is the area of focus of this research, there are several business drivers to this change [26]:

Lack of differentiation.
Whilst almost all banks have launched digital offerings, many have failed to build distinctive propositions. As an example, most trading platforms offered by banks have similar capabilities (and largely similar functionalities) and are different only by look-and-feel. Also, the propositions often on these platforms do not tend to be particularly innovative – they are mostly offline offerings being moved online and often do not create fundamentally different revenue streams.

Misunderstood economics of electronification.
Many digital investments made by banks (some to the tune of $100m or more) have simply not paid back, partly due to the lack of differentiation mentioned above. Further, clients often use digital channels as 'shop windows' and continue making transactions through voice channels, meaning that digital channels become loss leaders due to hefty digital spend. Electronification has compressed margins without bringing in the increase in volumes expected.

Declining foreign exchange trade volumes.
For the first time since 2001, global foreign exchange (FX) trading volumes have declined between two consecutive surveys conducted by

the Bank for International Settlements (BIS). Global FX turnover was reported to have fallen $300 billion (almost 6%) from $5.4 trillion (April 2013) to $5.1 trillion (April 2016) worth of transactions per day [27]. Figure 2.2 below depicts FX turnover volume broken down by instruments (swaps, spots, forwards, options and other derivatives). This has further put pressure on banks to find more differentiated ways to generate revenues in a shrinking market.

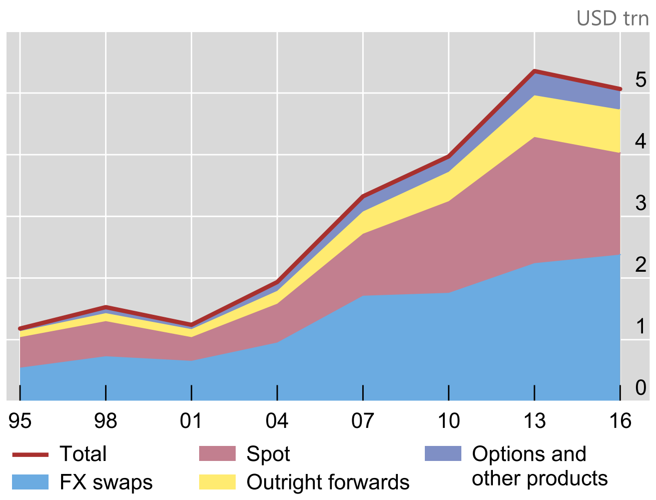


FIGURE 2.2: Turnover by Instrument [27]

Within this market context, there is now significant investment being poured into data analytics and innovation in financial institutions. Many have started to set-up Data/AI innovation groups labs in the past 2-3 years (for example BAML's Data and Innovation Group and BNP Paribas's Data Lab) as well as starting to persist eFX-data in databases.

2.2 Classical versus Bayesian machine learning

2.2.1 Probability and statistics

Probability is a self-contained mathematical discipline and probabilistic reasoning uses probabilistic models, which strictly obey the probability axioms [22]. Statistics, on the other hand, is more of an art, where for any given problem there may exist several plausible methods, which each give a different answer [6]. This distinction is important as machine learning is based on statistics and statistical learning theory. Tibshirani et al [21] describe statistical learning as a set of supervised and unsupervised tools for understanding data.

2.2.2 Classical versus Bayesian statistics

In the field of machine learning the 250-year old debate between Classicists (also called frequentists) and the Bayesians is widely known. They form two distinct competing camps when it comes to the approach and philosophy of statistical inference (decision making). Broadly speaking, Bayesian statistics dominated 19th century statistical practice, while the 20th century was more Classicist [14] The fundamental difference is on how they handle unknown models or variables. The Bayesians treat model parameters as random variables with known distributions, whereas the Classicists treat them as deterministic quantities, which happen to be unknown [6].

A Classicist's version of probability is defined as the long-term frequency of an observation or measurement. The Classicists believe in a single truth. They claim the more data they collect the more accurately they can predict the truth. For example in a coin flipping experiment, the Classicists would perform repeated flips to see if the long-term frequency is heads or tails, with the long-term frequency eventually tending to the truth [2].

Bayesians on the other hand define probability as the plausibility of a hypothesis given an incomplete and subjective knowledge. Bayesians are not seeking a single truth via data collection. Bayesians use data as evidence for particular hypothesis. If a Bayesian were to flip a coin, they would simply observe a number of flips and then use this information to make deductions about the fairness of the coin [2].

The Bayesians and Classicist each have a different interpretation of probability. If we take the example of flipping a fair coin, commonly we say the probability of obtaining a tails is 0.5. The Classicists view probability as long-run frequencies of events. In the coin flip example, they would think if we flip the coin many times, we expect it to land on tails half the time [28]. On the other hand Bayesians use probability to quantify uncertainly about something, therefore it is fundamentally related to information and not repeated trials. With regards to the coin flip, Bayesians believe the coin is equally likely to land on heads or tails on the next flip.

Many machine learning problems are well suited to the Bayesian philosophy particularly where we need to model uncertainty around events which do not have long-term frequencies. For example, will the Antarctic ice sheet melt in the next 20 years? Or we observe a blip on a radar screen and we want to compute the probability distribution over the location of the corresponding target (e.g. a missile) [4]. In both of these examples, the Classical approach falls short as the idea of repeated trials does not make sense. Table 2.1

provides a brief comparison of the Bayesian versus Classical philosophy.

TABLE 2.1: Bayesian and Classical methods compared and contrasted

Bayesian	**Classical**
The fairness of a flipped coin is answered by a probability distribution. It is not a fixed value and it is the experimenter's subjective perception. For example: $p\left(\text{coin} = H \vert \text{Data}\right)$	The fairness of a flipped coin is answered by a single value, which would be the inherent property of the coin. Thus everyone should arrive at the same answer. For example $p\left(\text{coin} = H\right) = 0.5$
In the coin flipping case, as you feed more data into your model the distribution $p\left(\text{coin} = H \vert Data\right)$ tightens around the 'true' value of 0.5	As you feed more data into your model your confidence interval tightens around the 'true' value of 0.5
Computationally intensive, but having a renaissance with more computing power. It requires sampling methods for all but a small family of distributions.	Less computationally intensive than the Bayesian approach.
Missing data are a common problem and Bayesians are able to deal with this by summing out variables.	All missing data must be provided for when multiplying matrices by using for example interpolation.
Common practice in 19^{th} Century.	Common practice in 20^{th} Century.
Unknown parameters are treated as random variables with known prior distributions	Unknown parameters are treated as constants to be determined
Requires knowledge or construction of a 'subjective prior'.	Does not require a prior.
Relies on presenting presents density graphs of posterior distributions.	Relies on presenting p-values and confidence intervals.
The main inference method is maximum a posteriori (MAP).	The main inference method is maximum likelihood (ML).
Returns parameter estimates which are close to the true parameter values in a probabilistic sense.	Returns parameter estimates which are nearly correct under any possible value of unknown parameter.
The classical confidence interval is equivalent to probability mass distribution between two values.	Unable to reject Null Hypothesis to establish likelihood of fair coin at the desired confidence interval e.g 95%

2.2.3 Bayes' Theorem

The driving force of Bayesian machine learning is Bayes' Theorem. We first present a brief recap of Bayes' Theorem (also referred to as Bayes' rule or Bayes' law). Named after the English statistician Thomas Bayes (1702–1761) Bayes' rule is a statement of conditional probabilities, and plays a pivotal role in probabilistic reasoning; as D Barber [4] points out, the rules of probability combined with Bayes' rule make it a complete reasoning system.

The probability of an event A given another event B (A conditioned on knowing B) is expressed as the conditional probability expression:

$$p(A|B) \tag{2.1}$$

It can further be shown that the above conditional probability can be expressed as:

$$p(A|B) = \frac{p(A \text{ and } B)}{p(B)} = \frac{p(A,B)}{p(B)} \tag{2.2}$$

$$p(B|A) = \frac{p(A \text{ and } B)}{p(A)} = \frac{p(A,B)}{p(B)} \tag{2.3}$$

Re-arranging equations (2.2) and (2.3):

$$p(A,B) = p(A|B) \cdot p(B) \tag{2.4}$$

$$p(B,A) = p(B|A) \cdot p(A) \tag{2.5}$$

Equating the right hand sides of Expressions 2.4 and 2.5 we arrive at Bayes' rule:

$$\overbrace{p(A|B)}^{\text{posterior}} = \frac{\overbrace{p(B|A)}^{\text{likelihood}}\overbrace{p(A)}^{\text{prior}}}{\underbrace{p(B)}_{\text{evidence}}} \tag{2.6}$$

Bayes' rule shows how from a generative model $p(B|A)$ of the data set plus your prior (initial) belief/knowledge $p(A)$ about the most appropriate variable values before any evidence is taken into account, one can infer the posterior distribution $p(A|B)$ of the variable given the observed data.

The most probable posterior is known as the maximum a posteriori probability (MAP). This is a setting which maximises the posterior, i.e.:

$$A_* = \text{argmax}_A \, p(A|B) \tag{2.7}$$

In the case of a flat prior i.e. when $p(A)$ is constant with respect to A, the MAP solution is equivalent to the maximum likelihood, i.e. the A that maximises the likelihood $p(B|A)$ of the model generating the observed data

[4]. The posterior distribution is what we infer in light of the observed data. It is the probability that A is true after considering the data.

The steps involved in Bayes' rule are best explained using a simple example of smoking and lung cancer with the aid of a Venn diagram (Figure 2.3).

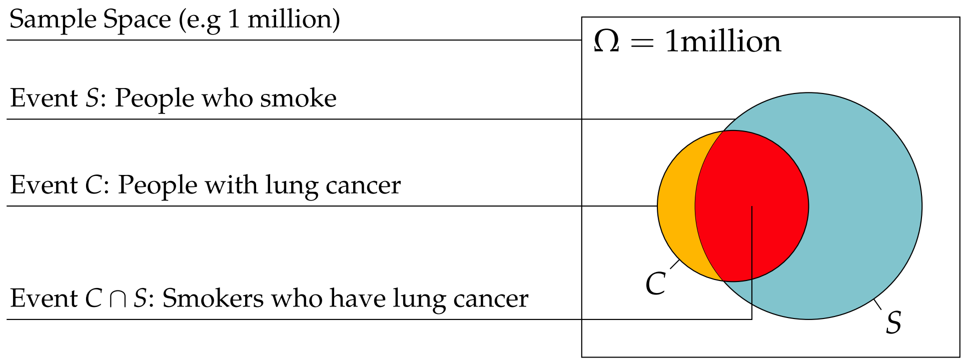

Sample Space (e.g 1 million)
Event S: People who smoke
Event C: People with lung cancer
Event $C \cap S$: Smokers who have lung cancer
$\Omega = 1$million
C
S

FIGURE 2.3: Venn diagram of smoking and lung cancer

In Figure 2.3 $p(C|S)$ represents the probability you that have lung cancer given that you are a smoker. The likelihood $p(S|C)$ is the flip side of the posterior distribution. This is the evidence about C provided by the data S.

In our example $p(S|C)$ the probability of being a smoker given that the person has lung cancer. One could easily measure this by interviewing people with lung cancer and collecting the data.

The prior $p(C)$ is an initial belief probability that C is true prior to considering the data i.e. the probability of having lung cancer.

The evidence $p(S)$ is known as the marginal likelihood or the normalisation constant. This can also be thought of as the total probability taking everything into account. In our example it would equate to the probability of being a smoker.

In further support of the Bayesian framework, D. Barber [4] points out that in fact much of science deals with problems of the form: tell me something about the variable A given that I have observed data B and have some knowledge of the underlying data generation mechanism.

2.3 Introduction to the hidden Markov model

2.3.1 Background and applications

Named after the Russian mathematician Andrey Markov, hidden Markov models (HMMs) are an essential framework for modelling time-series data.

HMMs are a type of Bayesian network (also referred to as a graphical model or a belief network) used for inference and learning, which are widely used in machine learning.

HMMs are used to solve real world problems with numerous successful applications, to name a few:

Bioinformatics – HMMs have applications in various strands of computational biology, such as gene finding and prediction programs [9], protein homology detection where the goal is to determine which proteins are derived from a common ancestor, protein structure prediction and DNA annotation to determine what each gene does [11].

Speech Recognition – This is a well-known application area of the hidden Markov model (HMM) that includes chatbots and virtual assistants such as Siri and Alexa [9], recognition of speech in noisy environments [1], and speech recognition for people with speech articulation difficulties [3].

Handwriting Recognition – Another application of the HMM is the detection of handwritten characters and words [19] which have benefits of automation (e.g. form or report analysis, reading postal addresses). Other applications include analysing historical documents and improving machine-human communication.

Travel forecasting – In the car industry, HMMs allow for prediction of possible driver destinations and routes based on historical and on-going trips [24]. As a result of this, personalised driver assistance such as risk assessment of routes, re-routing, speed advice and engine management are all possible.

Traffic prediction – HMMs are used to analyse and predict both car traffic congestion in major cities [38] as well as network traffic volume estimation and prediction [10]. Both of these will aid better transport policy decision-making as well as proposing best routes.

Finance Domain – HMMs have numerous applications here, for example, improved trading strategies [35], filtering and forecasting prices [13] as well as financial modelling applications [36].

Object Tracking – HMMs are used to track objects where given an observed sequence of visible variables, hidden positions can be inferred [4]. Example 3.1 – the burglar – is a classic example of this in David Barber's book [4].

2.3.2 Belief networks

HMMs are based on Markov chains which are Bayesian belief networks at heart. Here we provide a very brief overview of Bayesian Networks.

A belief network (also called a Bayesian Network) is the marriage of graph theory and probabilities. It is a probabilistic graphical model, which represents a set of variables and their causal relationships via a directed acyclic graph (DAG). Each node in the network is associated with the conditional probability of the node given its parents [4]. Thus a belief network provides a complete model for its variables and their relationships. In turn this helps to answer probabilistic questions by using inference and learning techniques such as variable elimination and expectation maximisation (EM) algorithm.

In general, the joint distribution is obtained by using the conditional probabilities and it is possible to write any joint distribution $p(x_1, \dots, x_N)$ as follows:

$$p(x_1, \dots, x_N) = \prod_{n=1}^{N} p(x_n | x_{1:n-1}) \tag{2.8}$$

For example, the distribution of 4 variables $p(a, b, c, d)$ can be written in cascade form without loss of generality:

$$p(a, b, c, d) = p(a) \cdot p(b|a) \cdot p(c|a, b) \cdot p(d|a, b, c) \tag{2.9}$$

And represented as a belief network graphically as per Figure 2.4.

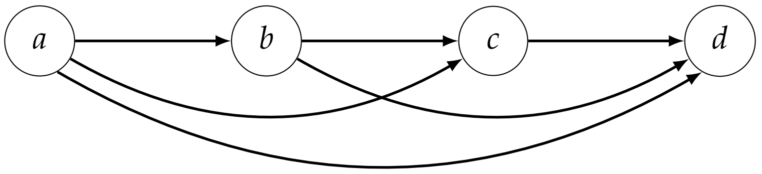


FIGURE 2.4: Cascade form belief network

2.3.3 Markov chains

A first order Markov Chain is depicted in the form of a belief network in Figure 2.5 and assumes that the immediate past is more relevant that the remote past in predicting the future [4]. It specifies a joint probability distribution $p(v_1, \ldots, v_T)$ over a collection of discrete or continuous variables. A first order Markov Chain is depicted in Figure 2.5.

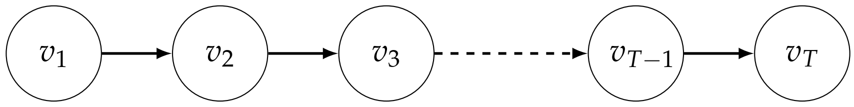


FIGURE 2.5: First order Markov chain

For a joint distribution of variables $v_1, ..., v_T$ the following conditional probability relationship holds [4]:

$$p(v_{1:T}) = p(v_1)p(v_2|v_1)p(v_3|v_2) \ldots p(v_T|v_{T-1}) \tag{2.10}$$

This can be generalised to the form below:

$$p(v_1, \ldots, v_T) = p(v_1) \prod_{t=2}^{T} p(v_t|v_{t-1}) \tag{2.11}$$

2.3.4 The hidden Markov model

The hidden Markov model (HMM) is a tool for representing probability distributions over a sequence of observations [16]. It has the following distinct properties [16]:

1. A visible observation at a particular time t is generated by some underlying process whose state H_t is 'hidden' (or 'latent') from the observer.
2. There is an assumption that the hidden process satisfies a first order Markovian property: where given the value of H_{t-1}, the current state H_t is independent of all the states prior to $t - 1$. Therefore the state at a particular time t captures everything we need to know about the history of the process in order to predict the future of the process [16]. The visible variables V_t also satisfy the Markov Property with respect to the hidden States, given V_t, H_t is independent of the hidden states and visible variables at all other times.

A first order HMM defines a Markov chain on hidden variables $h_{1:T}$. Each observed ('visible') variable is dependent on the hidden variables through what is known as the 'emission' distributions [4].

Figure 2.6 is a graphical model representing a HMM.

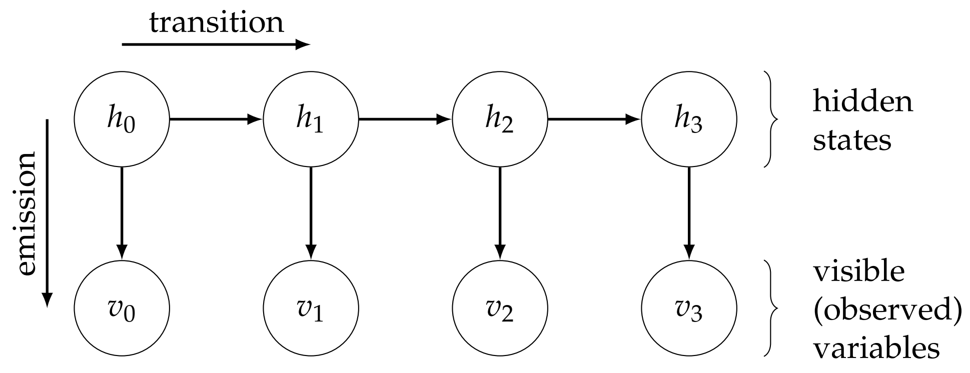


FIGURE 2.6: Hidden Markov model with 4 hidden states

The HMM Graphical model in Figure 4.5 can be more generally written as a set of joint distributions below:

$$p(h_{0:T}, v_{0:T}) = p(v_0|h_0)\underbrace{p(h_0)}_{a}\prod_{t=1}^{T}\underbrace{p(v_t|h_t)}_{b}\underbrace{p(h_t|h_{t-1})}_{c} \tag{2.12}$$

Where:

a: Is known as the prior

b: Represents the emission distribution, more generally written as: $p(v_t|h_t)$

c: Represents the transition distribution, more generally written: $p(h_t|h_{t-1})$

For discrete variables both the transition and emission distributions can be written in matrix form: a $H \times H$ matrix or the transition and $V \times H$ for the corresponding emission distribution. This matrix set-up and the format will be further discussed in the model set up for the RFQs.

At a high level a HMM is typically used to address 5 broad categories of inferential problems [4]:

1. ***Filtering*** – allows you to infer the present by obtaining the distribution of the hidden variable of interest ie $p(h_t|v_{1:t})$ using all the information up to the present $v_{1:t}$.

2. ***Prediction*** – $p(h_t|v_{1:s})$ inferring the future.

3. ***Smoothing*** – inferring the past $p(h_t|v_{1:u})$.

4. ***Likelihood*** – used to establish the likelihood of a sequence of observations $p(v_{1:T})$.

5. ***Most likely Hidden Path*** – this is known as the Viterbi Algorithm. $\text{argmax}_{h_{1:T}}\, p(h_{1:T}|v_{1:T})$

2.3.5 Learning using the hidden Markov model

2.3.6 Expectation maximisation algorithm

The Expectation Maximisation (EM) algorithm is a powerful iterative tool used to find maximising likelihood parameters when there is incomplete or missing data. This is very useful in real life examples where often we are faced with incomplete information. The EM algorithm applied to the HMM is known as the Baum-Welch [5] algorithm, but first we begin with the general form.

2.3.6.1 General form

To begin with, we set this up as a belief network below where we have 'visible' variables v_n and a hidden ('latent') variables h_n which can be laid out per belief network in Figure 2.7 where $\vec{\theta}$ is the collection of unknown parameters.

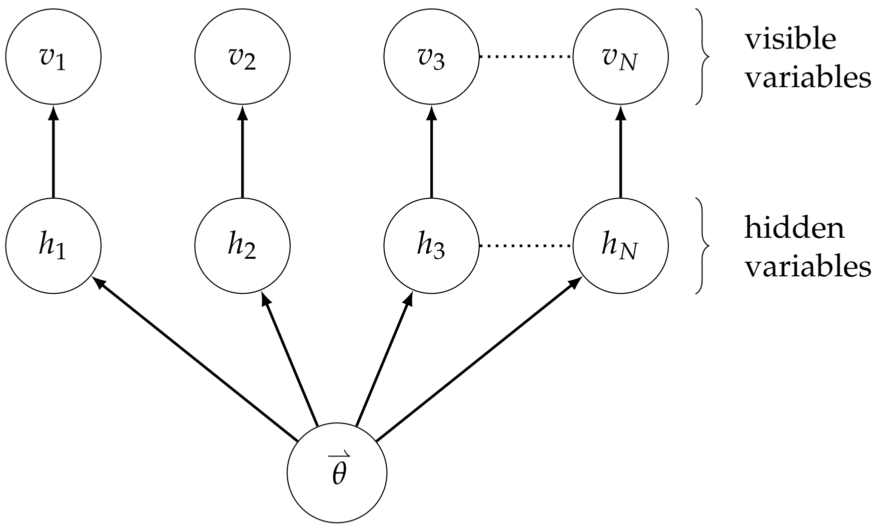


FIGURE 2.7: Belief network for a collection of parameters with hidden and visible variables

We can use 'maximum likelihood' to find values of $\vec{\theta}$ which maximise expression $p(v_1, v_2, \ldots, v_N|\vec{\theta})$, where v_n are actual values. This is another way of saying: what is the most likely $\vec{\theta}$ to have generated our data.

This objective is mathematically expressed as:

$$\max_{\vec{\theta}} p(v_1, v_2, \ldots, v_N|\vec{\theta}) \tag{2.13}$$

Which given the v_n are independent given $\overrightarrow{\theta}$

$$\max_{\overrightarrow{\theta}} \left(p(v_1|\overrightarrow{\theta}) \cdot p(v_2|\overrightarrow{\theta}) \cdot \dots \cdot p(v_N|\overrightarrow{\theta}) \right) \tag{2.14}$$

We can evaluate each term using the below expression:

$$p(v_1|\overrightarrow{\theta}) = \sum_{h_i} p(v_i, h_i|\overrightarrow{\theta}) = \sum_{h_i} p(v_i|h_i) p(h_i|\overrightarrow{\theta}) \tag{2.15}$$

However, maximising the overall expression below is difficult due to the summation and as such, there is no closed form solution.

$$p(v_1, v_2, \dots, v_N|\overrightarrow{\theta}) = \sum_{h_i, \dots, h_n} p(v_1, \dots, v_N, h_i, \dots, h_N|\overrightarrow{\theta}) \tag{2.16}$$

Therefore we use our iterative tool - the EM algorithm. We do not directly maximise (2.16) but we maximise a lower bound, which effectively pushes (2.16) into its maximum. This is repeated as a two-step process.

Lets call $p(v_1, \dots, v_N, h_i, \dots, h_N|\overrightarrow{\theta}) \rightarrow p(\overrightarrow{v}, \overrightarrow{h}|\overrightarrow{\theta})$ i.e. v and h vectors are gathered together. We also need to define a second probability distribution q which is what we need to change in each step.

Next, consider the Kullback-Leibler (KL) divergence expression below:

$$\mathrm{KL}(q(\overrightarrow{h}|\overrightarrow{v}), p(\overrightarrow{h}|\overrightarrow{v}, \overrightarrow{\theta})) \tag{2.17}$$

And as we know KL divergences are always non-negative (a proof for this provided in the Appendix B as this property is widely used), we therefore have the expression below:

$$\mathrm{KL}(q(\overrightarrow{h}|\overrightarrow{v}), p(\overrightarrow{h}|\overrightarrow{v}, \overrightarrow{\theta})) \geq 0 \tag{2.18}$$

Which can be expanded out to the form below (2.19)

$$\left\langle \log q(\overrightarrow{h}|\overrightarrow{v}) \right\rangle_{q(\overrightarrow{h}|\overrightarrow{v})} - \left\langle \log p(\overrightarrow{h}|\overrightarrow{v}, \overrightarrow{\theta}) \right\rangle_{q(\overrightarrow{h}|\overrightarrow{v})} \geq 0 \tag{2.19}$$

And given the 3 statements below:

$$p(\vec{h}|\vec{v},\vec{\theta}) = \frac{p(\vec{h},\vec{v}|\vec{\theta})}{p(\vec{v}|\vec{\theta})} \tag{2.20}$$

$$\log\left(\frac{a}{b}\right) = \log a - \log b \tag{2.21}$$

$$\langle a-b\rangle_q = \langle a\rangle_q - \langle b\rangle_q \tag{2.22}$$

Expression (2.19) can be re-written as:

$$\left\langle \log q(\vec{h},\vec{v})\right\rangle_q - \left\langle \log p(\vec{h},\vec{v}|\vec{\theta})\right\rangle_q + \left\langle \log p(\vec{h}|\vec{\theta})\right\rangle_q \geq 0 \tag{2.23}$$

Where for convenience q represents a short-hand for $q(\vec{h}|\vec{v})$. We can re-arrange the above expression to give:

$$\underbrace{\left\langle \log p(\vec{v}|\vec{\theta})\right\rangle_q}_{a} \geq \left\langle \log p(\vec{h},\vec{v}|\vec{\theta})\right\rangle_q - \left\langle \log q(\vec{h}|\vec{v})\right\rangle_q \tag{2.24}$$

The first term (a) in the preceding expression equates to $p(\vec{v}|\vec{\theta})$ as there is no q distribution term inside $\langle\cdots\rangle$ and the expectation with respect to q has no impact, thus resulting in the Expression (2.25):

$$p(\vec{v}|\vec{\theta})_q \geq \underbrace{\left\langle \log p(\vec{h},\vec{v}|\vec{\theta})\right\rangle_q}_{\text{known as the ``Energy'' term}} - \underbrace{\left\langle \log q(\vec{h}|\vec{v})\right\rangle_q}_{\text{known as the ``Entropy'' term}} \tag{2.25}$$

Therefore if we can manipulate and maximise the right hand side of the expression above, because of the "$\geq$" term we effectively maximise the left hand side as well. To maximise the right hand side, we use the EM algorithm.

We choose any initial values for our parameters in $\vec{\theta}$, then keep repeating the E-step followed by the M-Step below until $\vec{\theta}$ stops changing and you end up with final values of $\vec{\theta}$.

E-Step – Let each $q(h_n|v_n) = p(h_n|v_n,\vec{\theta})$

M-Step – update $\vec{\theta}$ by choosing $\vec{\theta}$ that maximises the energy term.

2.3.7 Baum-Welch algorithm

The Baum-Welch algorithm [5] represents the application of the EM algorithm to the HMM.

Given a set of data the algorithm helps us find the HMM emission matrix B and transition matrix A and an initial vector a which is most likely to have generated our data set $\mathcal{V} = \{\mathbf{v}^1, \ldots, \mathbf{v}^N\}$ where $\mathbf{v}^n = v^n_{1:T_n}$ is of length T_n [4].

The assumptions for application of the Baum-Welch algorithm are that data set V is comprised of discrete (i.e. not continuous) set of variables. It is also assumed that the number of both hidden (H) and visible (V) variables is known, and each sequence was generated as independent and identically distributed random variables (IID).

The Baum-Welch follows the form described earlier in the section on General Form EM Algorithm more specifically with M and E steps below [4]

M-Step – The 'energy' term is maximised with respect to the transition matrix (A), emission matrix (B) as well as the hidden states $h_{1:T_n}$ ($\mathbf{h}^n$) and the initial vector a.

$$\sum_{n=1}^{N} \langle \log p(v^n_1, v^n_2, \ldots, v^n_{T^n}, h^n_1, h^n_2, \ldots, h^n_{T^n}) \rangle_{p^{\text{old}}(\mathbf{h}^n|\mathbf{v}^n)} \tag{2.26}$$

Applying the HMM structure to the above we obtain:

$$\sum_{n=1}^{N} \left\{ \langle \log p(h_1) \rangle_{p^{\text{old}}(h^n_1|\mathbf{v}^n)} + \sum_{t=1}^{T_n-1} \langle \log p(h_{t+1}|h_t) \rangle_{p^{\text{old}}(h_t, h_{t+1}|\mathbf{v}^n)} + \sum_{t=1}^{T_n} \langle \log p(v^n_t|h_t) \rangle_{p^{\text{old}}(h_t,|\mathbf{v}^n)} \right\} \tag{2.27}$$

The probability that the first hidden variable is in a state i can be expressed as $p^{\text{new}}(h_1 = i)$ and optimising Expression 2.27 by forcing $p(h_1)$ to be a distribution and avoiding the sequencing of h variables index we obtain:

$$a^{\text{new}}_i \equiv p^{\text{new}}(h_1 = i) = \frac{1}{N} \sum_{n=1}^{N} p^{\text{old}}(h_1 = i|\mathbf{v}^n) \tag{2.28}$$

Expression 2.28 is the number of times on average the first hidden variable is in a particular state i. It then follows that the M-Step for the transition between the hidden variable is:

$$A^{\text{new}}_{i',i} \equiv p^{\text{new}}(h_{t+1} = i'|h_t = i) \propto \sum_{n=1}^{N} \sum_{t=1}^{T_{n-1}} p^{\text{old}}(h_t = i, h_{t+1} = i'|\mathbf{v}^n) \tag{2.29}$$

Basically, how many times do we transition from one state (i) to another state i'? Normalising Expression 2.29 using a normalisation constant results in Expression 2.30.

$$A^{\text{new}}_{i',i} = \frac{\sum_{n=1}^{N}\sum_{t=1}^{T_n-1} p^{\text{old}}(h_t = i, h_{t+1} = i'|\mathbf{v}^n)}{\sum_{i'}\sum_{n=1}^{N}\sum_{t=1}^{T_n-1} p^{\text{old}}(h_t = i, h_{t+1} = i'|\mathbf{v}^n)} \tag{2.30}$$

We can then perform the M-Step for the emission from hidden states to visible below. This is effectively the expectation of the number of times the visible variable is in a state j and the hidden in a state i (the normalisation constant is replaced with the proportional symbol).

$$B^{\text{new}}_{j,i} \equiv p^{\text{new}}(v_t = j|h_t = i) \propto \sum_{n=1}^{N}\sum_{t=1}^{T_n} || \, [v_t^n = j] \, p^{\text{old}}(h_t = i|\mathbf{v}^n) \tag{2.31}$$

E-Step – The smoothing inference method described earlier ($p(h_t|v_{1:u})$) is used to obtain the expression below. The solution then involves repeating (2.28), (2.30), and (2.31) above until convergence where the algorithm converges to a maximum likelihood.

$$p^{\text{old}}(h_1 = i|\mathbf{v}^n) \tag{2.32}$$

$$p^{\text{old}}(h_t = i, h_{t+1} = i'|\mathbf{v}^n) \tag{2.33}$$

$$p^{\text{old}}(h_t = i|\mathbf{v}^n) \tag{2.34}$$

Note, the EM algorithm may not always find the global maximum of the likelihood and such it turns out that sensible initialisation of the emission distribution is key for success [4].

2.4 Artificial neural networks

2.4.1 Background and applications

The first mathematical model of artificial neurons was introduced in 1943 by McCulloch and Pitts [25]. The idea was further developed by Franck Rosenblatt [33] who in 1958 introduced the concept of a perceptron algorithm (generic name given to artificial neural networks).

Artificial neural networks (ANNs) are powerful, layered machine learning models, which attempt to computationally model the information processing abilities of biological neurons in our brains. There are many types of ANNs, such as feed-forward neural networks (NNs), convolutional neural networks and recurrent neural networks. This introduction will focus mainly on feed-forward neural networks which are the most common type of NN; for example, the MLPRegressor model in the popular python library scikit-learn.

Neural Networks have a wide range of applications in the real world, to name a few:

Image recognition – Geoff Hinton et al [23] use deep convolutional neural networks (CNNs) to successfully classify 1.2 million images into 1,000 different classes from the ImageNet data-set with record breaking low error rates compared to previous attempts.

Medicine – Neural networks (NNs) have been applied successfully in disease diagnosis. One application of NNs has been in Urology where specialists are often faced with difficulties in obtaining a correct diagnosis. This is mainly due to the multiple complex variables involved in urological dysfunctions. David Gil et al [17] successfully used multilayer perceptron trained with backpropagation to help urologists make more accurate diagnoses.

Time-series prediction – Neural networks (NNs) have been widely used to model time-series data in a variety of fields but in particular finance and economics. Huang et al [20] used backpropogation NNs to successfully model and predict corporate credit ratings for the Taiwan and US markets making such information more accessible to investors.

Natural language processing (NLP) – NNs have helped to solve many problems in this area with applications in text classification, part of speech tagging, character recognition, spell checking and machine translation. Sebastien Riedel and Tim Rocktäschel [32] recently used neural networks to infer facts from a given knowledge base. This has enabled automation completion of large Knowledge Bases by establishing complex reasoning patterns based on similarities.

2.4.2 Feed-forward neural networks

Feed-forward Neural Networks (also referred to as deep forward neural networks or multi-layer perceptrons (MLPs)) are primarily used for supervised learning.

Figure 2.8 depicts a typical graphical architecture for a feed-forward neural network where the circles represent neurons. The input layer information

x_1:x_3, which needs to be evaluated, flows forward through the one hidden layer to the output layer y_1:y_2. They are called feed-forward, as there are no feedback connections where output information is fed back on itself into the network. There are weights and bias parameters assigned to each arrow, which are typically initialised to a small random value at the start.

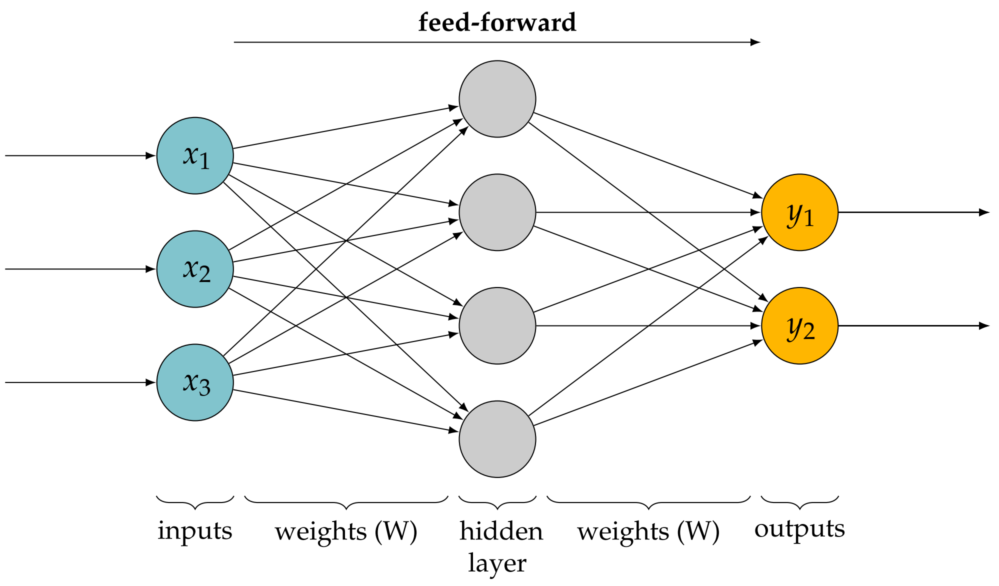


FIGURE 2.8: Example high-level architecture for a feed-forward neural network with 3 inputs and 2 outputs

Figure 2.9 below is a simple feed-forward neural network with 1 input vector 'x' and 1 hidden layer. The predicted output 'y' is produced as a result of feeding 'x' input(s) through the hidden layers and eventually it translates to a loss or cost function when compared with the actual value.

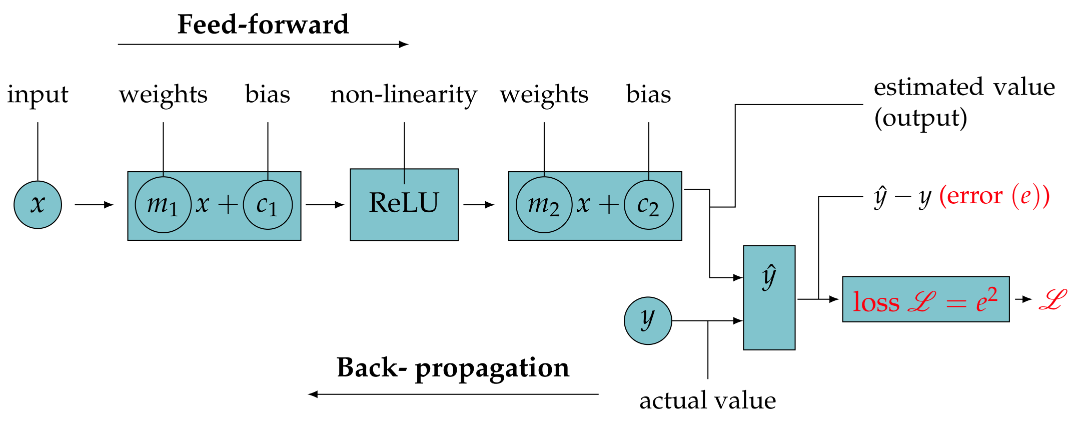


FIGURE 2.9: Example of a feed-forward neural network with labelled components

The total error of the network is called the loss (or cost) function L, which measures the difference between the actual and predicted values. The objective is to minimise this loss, which is a mean squared error (MSE) of the general form below:

$$\mathscr{L} = \frac{1}{n} \sum_{i=1}^{n} \left(y^{(i)} - \hat{y}^{(i)} \right)^2 \tag{2.35}$$

The 'squared' loss function is smooth and is differentiable, which is necessary for back-propagation. The chain rule is used to calculate the derivative of the loss function with respect to the weights and bias term.

Rather than focusing on the individual neurons, feed-forward neural networks are more commonly thought of as using the framework of linear algebra.

Sets of variables at each layer of the network can be represented in vector form and the linear multiplication layers can be represented as matrix multiplications. An advantage of this view is that it allows neural network training and prediction calculations to be performed efficiently and at scale as it makes use of already-existing linear algebra software libraries.

The number of hidden layers determines the 'depth' of the model and the existence of more than one hidden layer has lead to the term 'deep' neural networks or 'deep learning'.

In designing a neural network one should consider the number of hidden layers as well as the type of 'activation function' used to calculate the hidden layer values.

The component ReLU stands for rectified linear unit and is the most common 'activation functions' used in feed-forward neural networks. Other activation functions such as sigmoid or Tanh can also be used in NNs. ReLU is in fact a non-linear function, where a given input 'x' and output 'y' (Figure 2.10) can be represented by Expression 2.36 and the line plot in Figure 2.11.

$$\begin{tikzcd} x \arrow[r] & \boxed{\text{ReLU}} \arrow[r] & y \end{tikzcd}$$

FIGURE 2.10: Non-linear rectified linear unit block with input and output

We end up with the mathematical relationship below as a result of the ReLU transformation:

$$y(x) = \begin{cases} x & x \geq 0 \\ 0 & \text{otherwise} \end{cases} \tag{2.36}$$

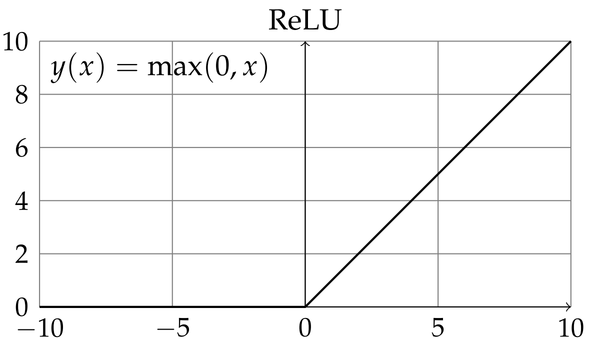


FIGURE 2.11: Line plot of ReLU

ReLU plays an important role in the NN architecture - without it, the network ends up being a group of cascaded linear functions which overall are still linear and thus unable to fit well to complex data sets. Therefore the non-linearity is an important property of the network which also helps to deal with a large spread of input values.

The back-propagation algorithm is used to compute the gradients of functions in our network. The purpose of back-propagation is to update the assigned weights in our neural network such that the overall NN loss is minimised. Thus the partial derivative of the loss is obtained with respect to all parameters (namely the weights and bias terms). Once these are known we use gradient descent to update the values. For example an update for weight m_1 is shown below with learning rate α which is a constant.

$$m \leftarrow m_1 - \alpha \frac{\partial L}{\partial m_i} \tag{2.37}$$

As a side note, Goodfellow et al [33] point out the term back-propagation is often misquoted to be the entire learning algorithm for multi-layer neural networks. In fact it refers to a method for computing the gradient while an iterative algorithm such as stochastic gradient descent ('sgd') is used to perform learning using this gradient to in order to minimise the loss value.

3 RFQ Data Set

This chapter presents the RFQ data time-series which was generated as part of this thesis for the predictability and hidden state inference investigation. It provides graphical descriptions of the data, along with justification for choosing a 'binning' methodology to process the time-series. Finally it presents the model training, validation and test split methodology used as well as the required transformations performed using data wrangling techniques.

3.1 Data generation and descriptive statistics

The RFQ time-series data was generated for the purpose of this investigation based on EUR-USD currency pair and product type European vanilla option. The frequency of RFQs is representative of typical patterns of activity observed centred on London trading hours. This resulted in 44,687 RFQs generated over a period of 1 month. Table 3.1 is a snapshot of the first 9 entries in the RFQ data set.

TABLE 3.1: Ordered extract of generated RFQ data

ID	QUOTE_TIME_UTC	CCY_PAIR	SIDE	PRODUCT	NOTIONAL	CALL_PUT	STRIKE	EXPIRY_DATE	PREMIUM
1	30/10/17 00:19	EURUSD	Buy	European	1,000,000	C	1.168	13/11/17	4,060
2	30/10/17 00:56	EURUSD	Buy	European	1,000,000	C	1.174	13/11/17	2,380
3	30/10/17 01:18	EURUSD	Buy	European	1,000,000	C	1.178	13/11/17	1,610
4	30/10/17 01:57	EURUSD	Buy	European	20,000,000	C	1.17	14/12/17	187,800
5	30/10/17 02:28	EURUSD	Buy	European	1,000,000	C	1.1697	03/11/17	1,870
6	30/10/17 02:58	EURUSD	Buy	European	1,000,000	C	1.1607	06/11/17	5,350
7	30/10/17 03:20	EURUSD	Buy	European	500,000	P	1.14	30/01/18	2,571
8	30/10/17 03:55	EURUSD	Buy	European	500,000	C	1.18	30/01/18	4,589
9	30/10/17 04:20	EURUSD	Buy	European	500,000	C	1.18	30/01/18	4,591

This data set would typically form a subset of a SDP's RFQ data set, which would include at least G11 currencies as well as other option product types (American options, target forwards, Accumulators etc.). For simplicity the focus for this thesis was the RFQ subset generated.

3.1.0.1 'Binning' methodology

In order to prepare RFQ data for suitable analysis a decision had to be made upfront on how best to represent the RFQ time-series. The two approaches considered were 'binning' and 'time-intervals'. The binning approach involves bucketing discrete RFQs into 'bins' of equal width (for example 15 minute bins). The 'time-interval' approach would measure the differences in time between subsequent times. It was decided that the 'binning' approach would be more suited to predicting the probability of a RFQ in the next 'bin' due to the high frequency of RFQs especially in peak activity times. Also, the prediction of time intervals was not a realistic real-life scenario given the proximity of the RFQs.

As such the RFQ data was bucketed into 15 minute 'bins', with each 'bin' assigned the number of RFQs in that 15 minute slot. To demonstrate the distribution of RFQ binned data a histogram plot of the count frequency is presented in Figure 3.1. The plot shows that the count distribution is dominated by 'bins' with zero RFQ requests, this makes sense when you consider all the non-peak times in any given day when there are few requests.

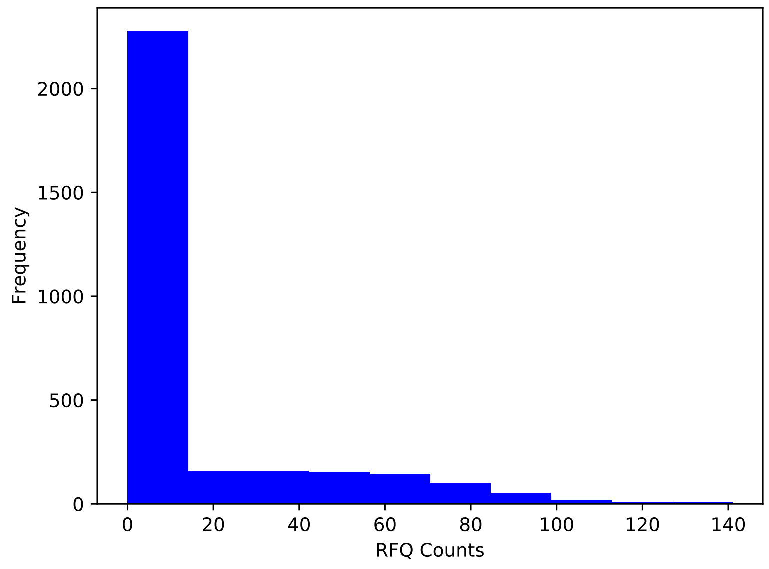


FIGURE 3.1: Histogram plot of RFQ count frequency

This appears to follow a power law distribution below, where x obeys a power law if drawn from a probability distribution where α is the scaling or exponent parameter and C is a normalisation constant [12].

$$p(x) = Cx^{-\alpha} \tag{3.1}$$

Figure 3.2 presents a plot of the best fit power law fitted to frequency and non-zero RFQ counts. Observed intervals with zero counts were removed to fit the power law. The parameters for equation 3.1 were found to be $C = 219.956$ and $\alpha = 0.826$.

Figure 3.2 shows the 44,687 RFQ data count distribution over the 1-month period. This clearly shows there is heightened RFQ activity daily with quieter periods in between peak times.

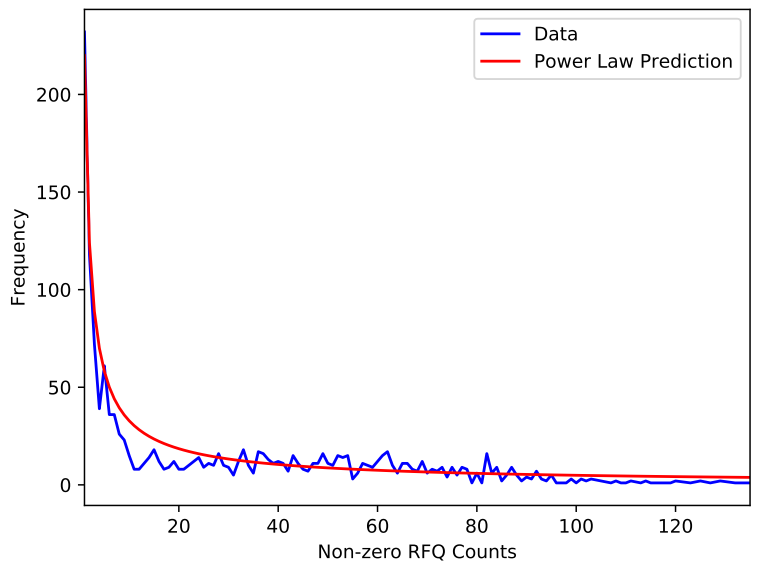


FIGURE 3.2: Plot of best fit power law distribution

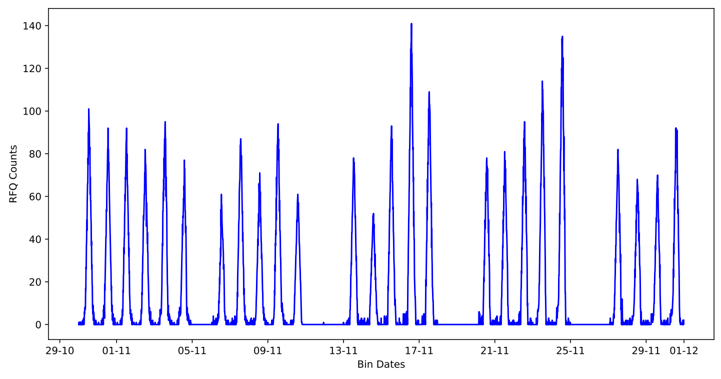


FIGURE 3.3: Distribution of RFQ data over a period of 1 month

The distribution for EUR-USD RFQ for any given day broadly follows the pattern shown in Figure 3.4. This is in line with the expectation of London trading hours. The RFQ activity starts to ramp up from 8am to around mid-day then starting to tail off until early evening when it is level again with RFQs from other time zones. It is expected that there will be peaks around significant political or economic news.

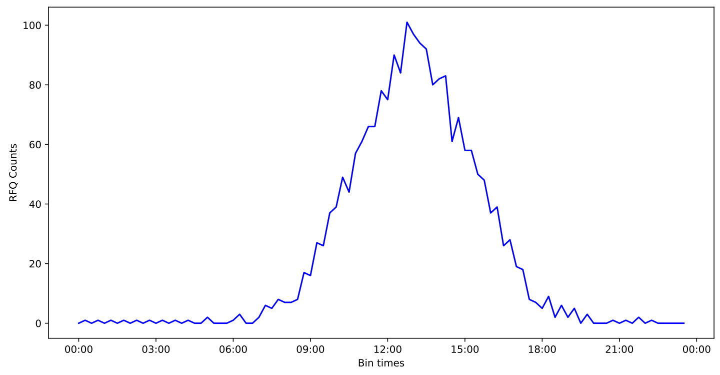


FIGURE 3.4: Distribution of RFQ data over a single day

3.2 Data split and wrangling

3.2.1 Data train-test split

The schematic in Figure 3.5 describes how the RFQ data is split for training, validation and testing. Broadly the steps apply to both our supervised and unsupervised models, which we are investigating however there are some differences with respect to the HMM, which will be highlighted. The 4 stages highlighted in Figure 3.5 are explained further below:

1. RFQ train-test split – this divides our data set into 2 subsets of training and test, which the model is first trained on and subsequently tested on to provide an unbiased evaluation of the performance. Note, the test data is set aside and untouched until the end to see how well the best model generalises. The train portion of this split is used for cross-validation.

2. Cross-Validation – the training portion of the dataset is used in this section. A k-fold cross-validation approach is taken using the either the default or a selected set of hyperparameters to be tuned. The model is trained on k-1 folds as the training data and the resulting model then validated on the remaining part of the data (labelled validation) to establish performance measures such as R^2 Score or RMS error. The cross-validation then reports the average values computed overall and selects the best model.

3. Re-Training – the model is then retrained on the training set using the best hyperparameters. Model performances are reported officially for the training set e.g. R^2 Score and RMS error.

4. Testing – this is the final step to evaluate how well the trained model generalises to a real world scenario. Performance measures such as RMS error and R^2 Scores are reported for the test data set.

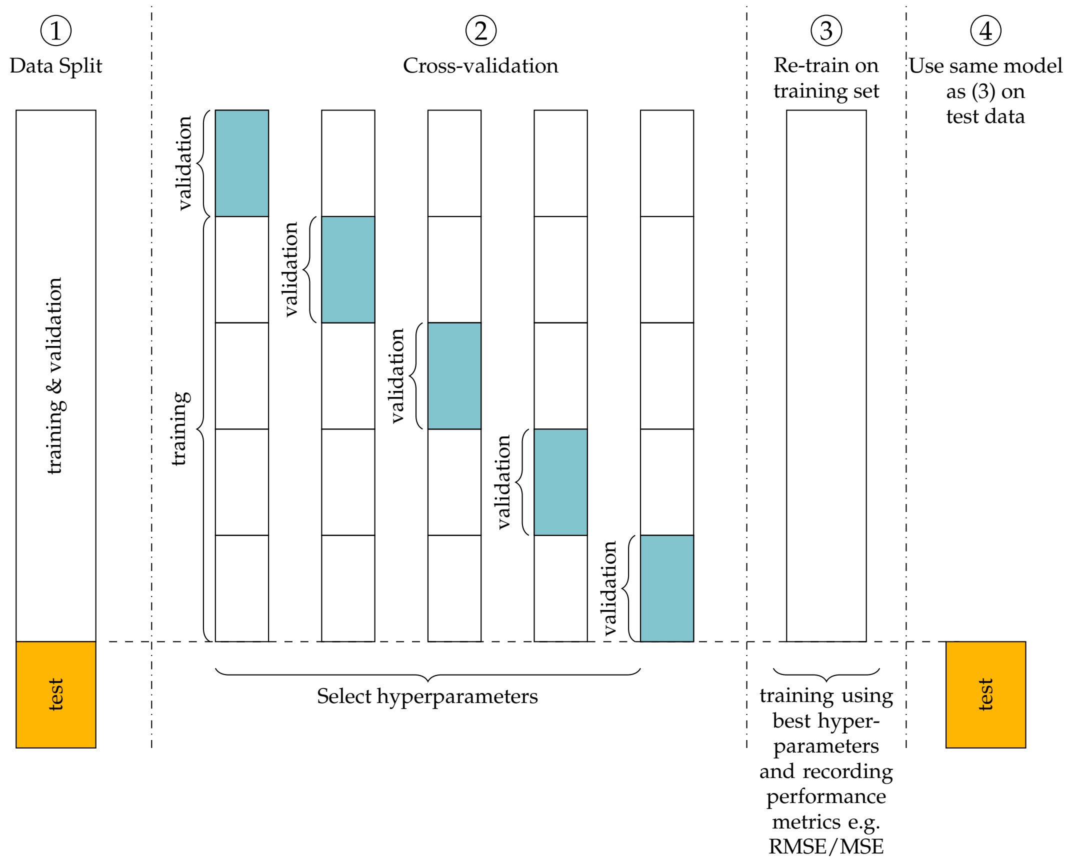


FIGURE 3.5: Methodology used for model training, validation and testing

The training process (steps 2 and 3 above) for the HMM is slightly different.

For the selected hyperparameters, the HMM is trained using maximum likelihood of the training data occurrence given the model. i.e. the HMM found is the mostly likely model to have generated the training data. This model is then evaluated on the likelihood of the validation data and the process is repeated for the each fold. In contrast to the classical models, where performance measures such as RMS error and R^2 Score are reported. The HMM, being a Bayesian model, is evaluated using the likelihood criteria and a prediction is made based on a probability distribution. This thesis reports RMS error for RFQ predictions made by the HMM.

3.2.2 Data wrangling

Data wrangling (also called data transformation or data munging) involves transforming the raw data into a suitable form for consumption by machine learning models.

The supervised (MLPRegressor, Ridge, Bayesian Ridge) and unsupervised (HMM) models we are investigating as part of this study require data to be in different shapes.

Supervised models expect data that contains both the input and the target, collectively referred to as labelled data. For this reason our chosen supervised models (MLPRegressor, Ridge, and Bayesian Ridge) need to be transformed into labelled data. The data is fitted into two matrices, a 'X' matrix (input data) and a 'y' matrix (labelled data), as shown in Figure 3.6.

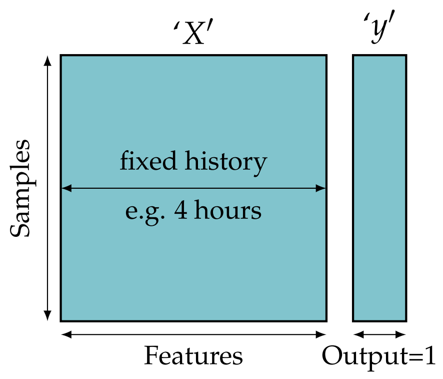


FIGURE 3.6: RFQ data transformed into an input matrix 'X' and labelled column matrix output 'y'

The rows in matrix 'X' are a set by the number of samples in the data set and the features equate to the history of RFQs and 'y' is a columnar target matrix of size (samples $\times$ 1), and it represents the RFQ counts for each bin. The ('X','y') matrix was produced 8 times, each instance corresponding to the data files in Table 3.2. As previously mentioned this study settled on investigating 15 minute bin widths and the files in Table 3.2 are all generated using 15 minute bin data. Files 1–4 represent labelled data for feature RFQ count, and files 5-8 represent features RFQ count and RFQ notionals. The 30 minute, 1 hour, 2 hours and 4 hours time variations refer to the length of history used in each file. For example a 30 minute history is comprised of two 15 minute columns for the 'X' input matrix.

Unlike the supervised models, there is no need to transform RFQs into labelled data for the HMM as it can deal with 'un-labelled' sequences. However, it was necessary to ensure that the 'visible' RFQ data was formulated

TABLE 3.2: Varying histories of labelled data used for supervised models

#	Files	Description
1	30min.csv	30 minute RFQ count history
2	1hr.csv	1 hour RFQ count history
3	2hr.csv	2 hour RFQ count history
4	4hr.csv	4 hour RFQ count history
5	notional_30min.csv	30 minute RFQ count and notional history
6	notional_1hr.csv	1 hour RFQ count and notional history
7	notional_2hr.csv	2 hour RFQ count and notional history
8	notional_4hr.csv	4 hour RFQ count and notional history

in such a way so that it plausibly obeys the Markov assumption made by the HMM. Hence, as explained in Section 4.4.2.1 a 'differences' method was used for the HMM. The additional feature called 'notional' was obtained by calculating the average FX option notional size in each 15 minute 'bin'. The average notionals were subsequently mapped using Table C.1 in Appendix C and used as a second feature in the construction of files 5, 6, 7, and 8 in Table 3.2 which contain both count and notional.

3.3 Conclusion

A comprehensive, production like time-series of EUR-USD vanilla European options RFQs was successfully generated for a period of 1 month. This was centred on London trading hours. A 'binning' methodology was chosen to analyse the time-series and for consistency, RFQ counts and notional were 'binned' into 15 minute buckets for all the analysis throughout this thesis. The histogram count frequency demonstrated that the RFQ count data follows a power law distribution. Finally, an overall methodology for preprocessing the data for model training, validation and testing was established as well as suitable data wrangling necessary to transform data into suitable forms for consumption by the models.

4 Experiments

The objective of this chapter is to present the experiments performed to investigate the predictability of RFQs. The chapter begins with the model selection framework common to all experiments. Each experiment is composed of an introduction, model implementation, presentation of results and discussion followed by a conclusion. Experiment 1 focuses on supervised learning covering both classical and Bayesian techniques. Experiments 2 and 3 are both unsupervised and Bayesian in nature and they use the hidden Markov model where the data consumed is transformed using a 'differences' method.

4.1 Overall model selection framework

The objective of this thesis is to investigate the predictability of e-FX option RFQs from both a classical and a Bayesian perspective. In order to perform this investigation effectively a number of models were carefully selected. They are presented in Figure 4.1 which provides a framework for the Experiments that follow.

	Simple "Supervised"	**Complex** "Unsupervised"
Classical	• Multi-layer Perceptron Regresser (MLPRegressor, a neural network) • Ridge Regression (least-squares with L2 regularisation)	K-means Clustering
Bayesian	Bayesian Ridge Regression	Hidden Markov Model

FIGURE 4.1: Two-by-two model selection framework. Note: K-means is provided for illustration purposes only and will not be investigated as it is not as sophisticated as the HMM

4.2 Experiment 1 - Predicting RFQs using supervised learning models

4.2.1 Introduction

This experiment was based on three supervised learning models, described as follows:

MLPRegressor – This supervised learning algorithm comprises of a feed-forward multi-layer perceptron (MLP) regressor which could be thought of as a 'deep' neural network in the presence of many hidden layers. Trained iteratively using back-propagation, the MLPRegressor is able to learn non-linear relationships. The configurable hidden layers (in number and size) are similar to those discussed in Section 2.4.2 for feed-forward neural networks. The MLPRegressor model uses an alpha parameter for L2 regularisation which prevents overfitting by assigning a penalty to large weights. The MLPRegressor model has the ability to learn using either Stochastic Gradient Descent (SGD), Adam (adaptive moment estimation) or Limited-memory BFGS. 'Adam' is a fairly commonly used learning method, and on the whole was found to give better results compared to SGD; therefore it was used consistently in Experiment 1. Using a gradient based solver for learning ensures the parameters in the network are updated using the gradient of the loss function with respect to the parameter being updated. The squared loss function for the MLPRegressor model is of the form [30]:

$$\text{Loss}(\hat{y}, y, W) = \frac{1}{2}||\hat{y} - y||_2^2 + \frac{\alpha}{2}||W||_2^2 \tag{4.1}$$

The algorithm initialises random weights and the network aims to minimise the Loss as it updates the weights. Once the network squared loss is known, the back-propagation method is used to propagate the loss back through the network and providing each weight parameter in turn with an adjustment to reduce the loss. 'Adam' can be used to make the adjustment below, where the current W is adjusted by the gradient of the loss [30].

$$W^{i+1} = W^i - \epsilon \nabla \, \text{Loss}_W^i \tag{4.2}$$

Where ϵ is the learning rate (greater than zero) and i is each iteration step. The update carries on until either the maximum number of iterations is reached or the smallest loss has been found.

Ridge Regression – This is a linear least squares regression algorithm with the addition of a L2-norm regularisation. It is similar to ordinary least squares (OLS), but attempts to improve on OLS by imposing a penalty on the size of coefficients and it aims to minimise a penalised sum of squares [30]:

$$\min_{w} ||Xw - y||_2^2 + \alpha||w||_2^2 \tag{4.3}$$

The first term $||Xw - y||_2^2$ represents the OLS loss and the second the regularisation (also called shrinkage penalty) which penalises the size of the regression coefficients. (Note the L2 norm squared definition: $||\mathbf{w}||_2^2 = w_1^2 + w_2^2 + \cdots + w_n^2$)

The hyperparameter alpha ($\alpha \geq 0$) controls the shrinkage. When $\alpha = 0$, the shrinkage penalty disappears and the result is OLS, when $\alpha \to \infty$ the shrinkage penalty has becomes more dominant, shrinking the estimates towards zero, introducing a bias while reducing the variance of the estimate.

Bayesian Ridge Regression – Being a Bayesian model this is the probabilistic version of the Classical Ridge regression model. A model deliberately chosen to compare against the Classical Ridge equivalent. The output y is a Gaussian distribution over Xw [30]:

$$p(y|X, w, \alpha) = \mathcal{N}(y|Xw, \alpha) \tag{4.4}$$

Figure 4.2 plots the Gaussian prediction distribution of output y. The plot was produced based on a sample RFQ test data using a trained Bayesian Ridge model. This plot demonstrates the Bayesian nature of this model, with the peak being the most likely prediction of the count outcome.

The prior for the coefficient w is given by a zero mean Gaussian [30]:

$$p(w|\lambda) = \mathcal{N}(w|0, \lambda^{-1}\mathbf{I}_p) \tag{4.5}$$

α and λ are both prior gamma distributions, with each offering two hyperparameter settings for Bayesian Ridge (α_1, α_2, λ_1 and λ_2).

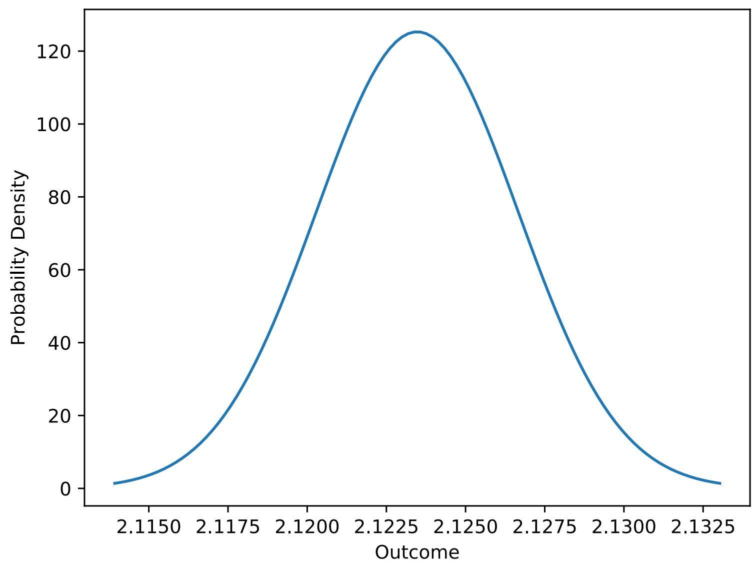


FIGURE 4.2: Gaussian RFQ count prediction for a single output y

The L2 regularisation in Ridge regression is equivalent to a MAP (maximum posteriori) estimate using a Gaussian prior over the coefficients w with precision λ^{-1}. λ is known as the precision of the weights and it is possible to estimate it from the data. Similarly α which can also be estimated from the data and is referred to as the precision of the noise. This thesis used the scikit learn implementation [30] of the supervised models above.

4.2.2 Model implementation

Figure 4.3 highlights the main implementation steps followed for all three supervised learning models. Each component is explained as follows:

1. Define hyperparameters – the first step involves defining the hyperparameter space. These are all the values which will be optimised using the cross-validation step.

2. Cross-validation – this step returns the best hyperparameters by tuning the provided hyperparameters using a grid search technique. The chosen method was `GridSearchCV` which iteratively considers all combinations of hyperparameters under a procedure called cross-validation. This experiment used a five-fold cross validation method where each model was trained on four of the folds used as training data and subsequently validated on the remaining one fold (Figure 3.5). This process

was repeated five times per model. The score and error measures reported from cross-validation were the average across all folds.

3. Fit model – this step fits each of the candidate models using the chosen hyperparameters to the RFQ data matrix $'X'$ and the columnar matrix labels $'y'$. This process quantifies the loss. This step now provides us with a trained model i.e. the best model trained using the best hyperparameters.

4. Evaluate Best Model – the best model was evaluated on the entire training data once again as well as the test data and error and score measures were captured.

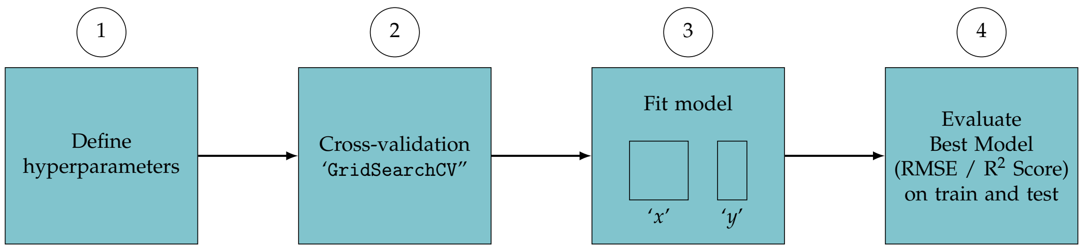

1
2
3
4
Define hyperparameters
Cross-validation 'GridSearchCV"
Fit model
'x'
'y'
Evaluate Best Model (RMSE / R² Score) on train and test

FIGURE 4.3: Experiment 1 model implementation and evaluation methodology

During the implementation (Figure 4.3) phase this study examined the Ridge model to ascertain if it behaved as expected. The target variable Alpha (the regularisation strength) was varied and its effect on the calculated weights was plotted (Figure 4.4). The plot was based on a 4 hour count history. This exercise also helped with selecting Alphas in a range which reduced the variation in weights. As the alpha tends to zero, the Ridge model behaves like ordinary least squares (OLS) and is very noisy. As the alpha increases the weight coefficients tend to zero as the shrinkage term effect dominates the loss expression.

4.2.3 Results and discussion

The results of Experiment 1 are shown in Table 4.1. This table is based on the RFQ count histories of 30 minutes, 1 hour, 2 hours and 4 hours. The equivalent results for count and notional are presented in Appendix A, Table A.1. The evaluation metrics of choice were RMS (root mean square) error (Expression 4.6) and R^2 Score. These two measures were reported for all three models on the train and test sets for all RFQ count and RFQ 'count

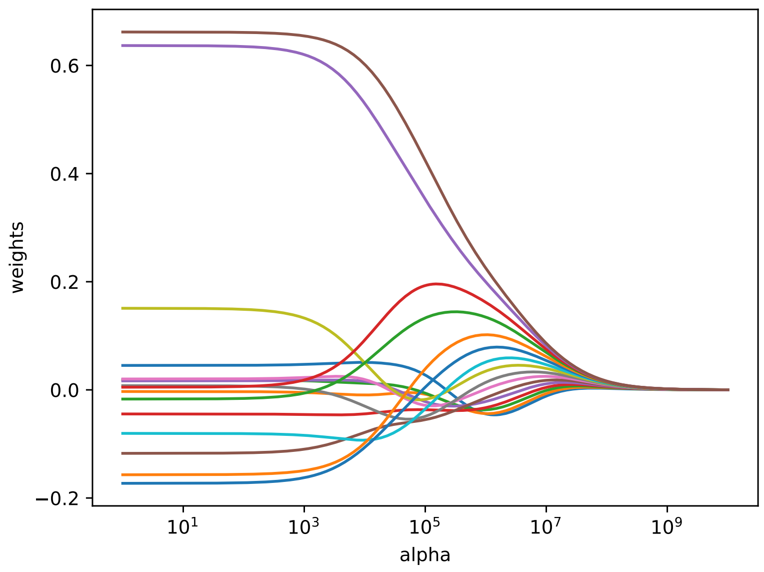


FIGURE 4.4: Plot of Ridge weights as a function of alpha, demonstrates that the Ridge model behaves as expected

and history' data files. The RMS error presents the standard deviation of the model prediction errors. It was chosen as a measure because it gives greater weighting to larger errors versus for example a mean average error (MAE) measure. The R^2 Score (also called coefficient of determination) is often used for regression models. The best models have a score of 1.0, constant models despite variations in features would be scored as 0. Note, the R^2 can also be negative if the model is arbitrarily worse.

$$\text{RMS Error} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}\left(y^{(i)} - \hat{y}^{(i)}\right)^2} \tag{4.6}$$

Table 4.2 presents the best hyperparameters found after performing cross-validation on the count data. They are the parameters which were used when evaluating all three models to obtain the RMS error and R^2 Scores in Table 4.1. The equivalent of Table 4.2 for both count and notional features is presented in Appendix A, Table A.2.

Table 4.2 demonstrates that the MLPRegressor model (the neural network) outperforms both the Ridge and Bayesian Ridge models, particularly for larger histories. This is not surprising as neural networks are more powerful models for predictions versus linear models. The R^2 Score (coefficient of determination) is very high (i.e close to 1) for all models and history lengths. This indicates that all models fit the all data files very well. The discussion

TABLE 4.1: Experiment 1 – results table for RFQ counts

Model	History Length	RMS Error Train	RMS Error Test	R^2 Score Train	R^2 Score Test
MLPRegressor	30 mins	4.14	4.17	0.98	0.97
	1 hour	3.05	3.17	0.99	0.98
	2 hours	2.28	2.64	0.99	0.99
	4 hours	2.27	2.51	0.99	0.99
Ridge Regression	30 mins	4.22	4.23	0.97	0.97
	1 hour	3.42	3.49	0.98	0.98
	2 hours	3.22	3.30	0.98	0.98
	4 hours	3.20	3.29	0.99	0.98
Bayesian Ridge Regression	30 mins	4.22	4.23	0.97	0.97
	1 hour	3.42	3.49	0.98	0.98
	2 hours	3.22	3.30	0.98	0.98
	4 hours	3.20	3.29	0.99	0.98

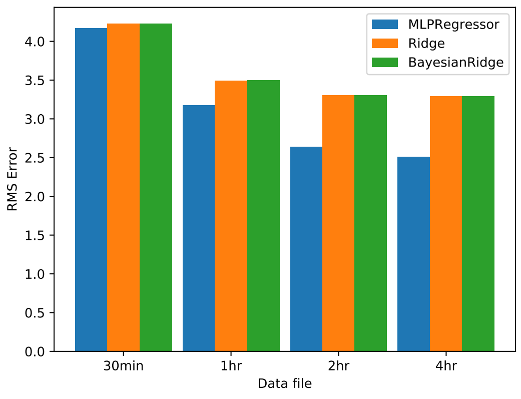
TABLE 4.2: Experiment 1 – hyperparameters chosen for model evaluation of the historical RFQ count data

	MLP Regressor Hyperparameters		Ridge Regression Hyperparameters	Bayesian Ridge Regression Hyperparameters			
RFQ History Length	hidden layer size	initial learning rate	alpha	alpha$_1$	alpha$_2$	lambda$_1$	lambda$_2$
30 mins	50, 50, 50	0.000316	210.49	1	1×10^{10}	359,381.37	1×10^{10}
1 hour	100, 100	0.000316	132.19	1×10^{-10}	1×10^{5}	1×10^{10}	1×10^{10}
2 hours	50, 50, 50	0.000316	849.75	1×10^{-10}	1×10^{5}	0.0774	1×10^{-10}
4 hours	400	0.000316	2,154.43	1×10^{5}	1×10^{10}	2,154.43	1×10^{5}

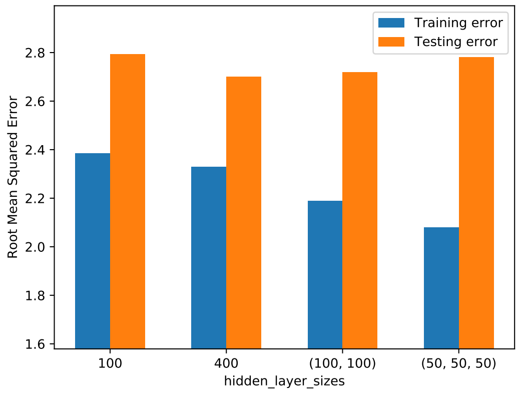
of results in Table 4.2 will focus mainly on RMS errors as there is very little variance in R^2.

The RMS error score for the MLPRegressor model indicates a lower standard deviation of the model prediction errors across all histories, versus the Ridge and Bayesian Ridge models. Both Ridge and Bayesian Ridge models perform very similarly in terms of errors. Figure 4.5a plots the test RMS error for all three models across four count histories. The MLPRegressor model performs better overall and particularly when there are longer histories. On the other hand, Ridge and Bayesian Ridge models only improve in error by a smaller fraction as the history increases from 30 minutes to 4 hours.

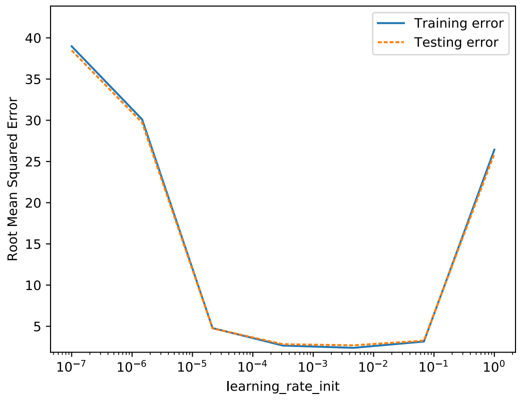
Analysing the MLPRegressor model further, Figure 4.5b shows the impact of different hidden layer architectures in the network on the RMS error. This plot was based on a 4 hour count history where '100' represents a single hidden layer formed of 100 neurons and '50, 50, 50' represents 3 hidden



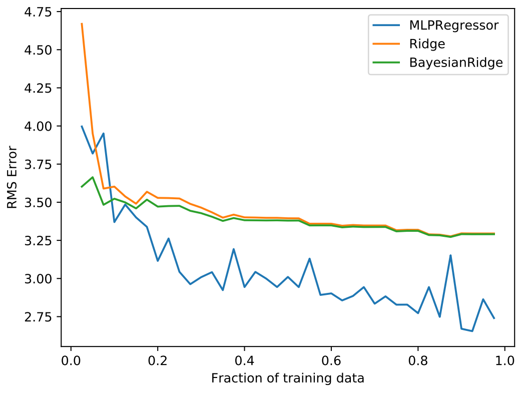
(a) Model RMS test errors for count



(b) MLPRegressor RMS error vs. hidden layers using 4 hour count history



(c) MLPRegressor RMS error vs. initial learning Rate hyperparameter using 4 hour count



(d) Model RMS test error vs. training data size for 4 hour count and notional

FIGURE 4.5: Experiment 1 – selected results plots

layers of length 50 neurons each. The plot shows that the more layers in the network the better the fit on the train data, however a single hidden layer with '400' neurons has the lowest test error.

Figure 4.5c plots the performance of a range of initial learning rate values versus RMS error for both the train and test sets. This was based on a 4 hour RFQ count history. The initial learning rate controls the step size during gradient descent for both 'SGD' and 'Adam' solvers (Experiment 1 used 'Adam'). It shows that an initial learning rate which is set too high ($> 10^{-1}$) or too low ($< 10^{-5}$) is not optimal for learning and results in a poor prediction and hence a large error. The optimal set of learning rates for initialisation are between 10^{-3} and 10^{-2}. The plot also shows that the train and test errors are very similar suggesting that the MLPRegressor model generalises well to the test data. Similar analysis was performed on hyperparameters for the

Ridge (alpha) and Bayesian Ridge (alpha$_1$, alpha$_2$, lambda$_1$, lambda$_2$) models. For the Ridge model, it was found that train and test RMS error were almost identical for values of alpha from 10^{-11} to 10^4. Beyond 10^4, the RMS error increased from below 5 to around 25 (Figure A.4a). For the Bayesian Ridge model, lambda$_1$ showed a similar step change behaviour to the Ridge model in RMS error. For lambda$_1$ values from 10^{-10} to 10^1, the RMS error was less than 5. Above 10^1, the RMS error increased to over 25 (Figure A.4b).

Figure 4.5d compares the RMS test error for all 3 models against the size of the training data. Overall we observe consistent behaviour where the MLPRgressor model outperforms both Ridge and Bayesian Ridge models. All three models perform better in the presence of more training data which is perhaps expected. Interestingly, the Ridge model performs better than Bayesian Ridge model in the presence of less training data but they both converge to the same error as we use 100% of the training data.

Table 4.2 displays the hyperparameters provided to each model (the hyperparameters not given assume default values). For each length of count history it presents the best hyperparameter value selected for each model using the cross-validation `GridSearchCV` method. These hyperparmaeters were then used in the 'best model' and this was evaluated on the training and test data sets to report the RMS error and R^2 scores in Table 4.1.

4.2.4 Conclusion

Experiment 1 demonstrates that all three supervised models are able to predict RFQ counts to a high degree and they generalise well to the test set. The MLPRegressor model is clearly a better predictor of RFQ data compared to the Ridge and Bayesian Ridge models, with an RMS error of only 2.27 for a 4 hour history. The MLPRegressor RMS error has greater sensitivity to the hyperparameters chosen, namely the initial learning rate and the hidden layers. On the other hand, Ridge and Bayesian Ridge models were found to be fairly insensitive to hyperparameters except for very large values of alpha and lambda$_1$. It is also evident that the presence of more training data greatly reduces the RMS test errors observed.

4.3 Experiment 2 - Predicting RFQs using the Bayesian unsupervised hidden Markov model

4.3.1 Introduction

Although Experiment 2 is about predicting RFQs, it should be noted that the hidden Markov model (HMM) is able to give much more sophisticated insight into the underlying process (as Experiment 3 will demonstrate). In theory, for prediction, a first order Markov chain (Figure 2.5) could be used, however what the HMM gives beyond a simple Markov chain is the ability to infer and reason about hidden states. An example of a HMM set-up for predicting RFQs is provided in Figure 4.6, where visible RFQ v_3 can be predicted having just observed visible RFQs v_0:v_2.

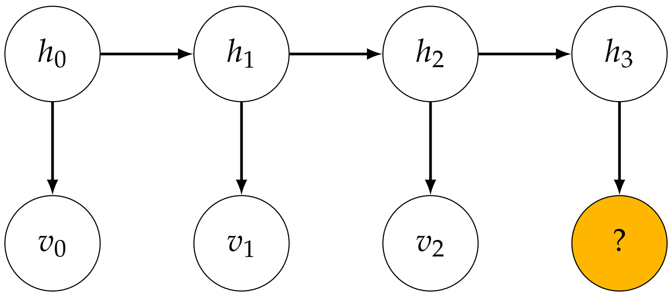


FIGURE 4.6: Hidden Markov model set-up to predict RFQ visible variable 'v_3'

4.3.2 Model implementation

In this application it turns out the HMM is not the best tool for predicting the next RFQ. Theoretically it is possible to make predictions via the hidden states of the HMM. Using Figure 4.6 as an example framework, it is possible predict what the RFQ count is at v_3 having just observed visible RFQs v_0:v_2. This achieved by calculating the conditional probability $p(v_3|v_0, v_1, v_2)$

The HMM in Figure 4.6 can be written as the expression below to make use of message passing inference:

$$p(h_0, h_1, h_2, h_3, v_0, v_1, v_2, v_3) = \\ p(h_0) \cdot p(v_0|h_0) \cdot p(h_1|h_0) \cdot p(h_2|h_1) \cdot p(h_3|h_2) \cdot p(v_1|h_1) \cdot p(v_2|h_2) \cdot p(v_3|h_2) \quad (4.7)$$

Where the proportionality below holds once the normalisation constant has been removed:

$$p(v_3|v_0,v_1,v_2) \propto \sum_{h_0,h_1,h_2,h_3} p(h_0,h_1,h_2,h_3,v_0,v_1,v_2,v_3) \tag{4.8}$$

It should be noted that the HMM being Bayesian could easily cope with missing data (unlike the classical methods). This property was used in the implementation where it was shown that it is possible to substitute any number with a `nan` and the model copes. It handles missing data by summing out probabilities, for example in the presence of three data points and where one was missing, it could simply sum out the missing variable. For instance in the expression below variable 'a' could be summed out if it was missing.

$$\sum_a p(a,b,c) = p(b,c) \tag{4.9}$$

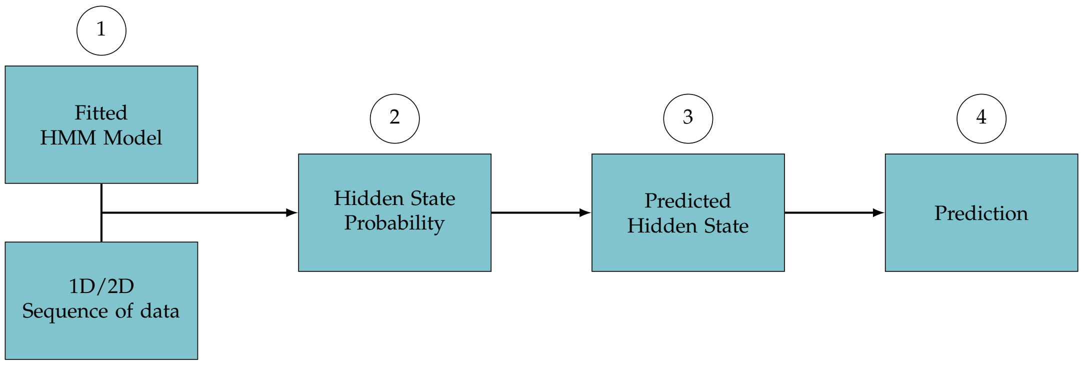


FIGURE 4.7: Implementation steps to predicting next RFQ using the HMM

Following on from the theory, below describes the implementation steps in Figure 4.7 which use the HMM to predict RFQs:

1. Fit HMM on 1 and 2 dimensional sequences – to begin with, it is necessary to create and fit a HMM. The HMM is then able to consume both 1-dimensional RFQ count sequence (1D per Figure 4.9a) and 2-dimensional RFQ count and notional sequence (2D per Figure 4.9b).

2. Hidden state probability – the probability of each hidden state at the end of the sequence of data is obtained.

3. Predicted hidden state – this step calculates the probability of the hidden states at the next time step using a defined transition matrix.

4. Prediction – finally a RFQ prediction is made by selecting the most likely state by using the mean of its emission distribution.

4.3.3 Results and discussion

Table 4.3 presents the RMS errors for prediction of RFQs based on 'differences' sequences taken from the data split of train and test. The sequences used were 2, 4, 6 and 8 in length equivalent to 30 minutes, 1 hour, 2 hours and 4 hours. This was performed for both 'count' sequences and 'count and notional' RFQ sequences.

The test errors marginally improve when using 2D data (i.e. when also incorporating notional sequences) when predicting using shorter sequences. However, surprisingly, as the history lengthens, only the 'count' sequences improve in error compared to 'count and notional' where the error gets marginally worse using longer sequences. In comparison to Experiment 1, the HMM performs worse than all three supervised models with an average RMS error of 4.40 for count.

TABLE 4.3: Hidden Markov model RFQ prediction error

Sequences	RMS Train Error	RMS Test Error
30 mins	4.59	4.57
1 hour	4.40	4.37
2 hours	4.21	4.20
4 hours	4.19	4.33
Count and Notional 30 mins	4.26	4.29
Count and Notional 1 hour	4.26	4.31
Count and Notional 2 hours	4.26	4.33
Count and Notional 4 hours	4.27	4.39

4.3.4 Conclusion

Experiment 2 demonstrated that it is possible to make use of the hidden Markov model to make RFQ predictions by using sequences of 'differences'. However, the hidden Markov model does not appear to predict as well as the supervised models due to larger RMS errors. Hence, Experiment 3 attempts to make more insightful use of the the HMM and its hidden states to reason about the underlying RFQ market regime.

4.4 Experiment 3 - Inferring RFQ hidden states using the hidden Markov model

4.4.1 Introduction

There are two possible approaches for the set-up in Experiment 3. Firstly using our domain knowledge to subjectively impose an emission and transition matrix, instead of training those using the processes of 'filtering' and the 'Viterbi' algorithm to infer the most likely path. The second option is to use the EM algorithm to train our model, where the EM algorithm would perturb the emission and transitions distributions as well as the 'prior' to find the set which best fits the data. This thesis uses the EM learning approach.

4.4.2 Model implementation

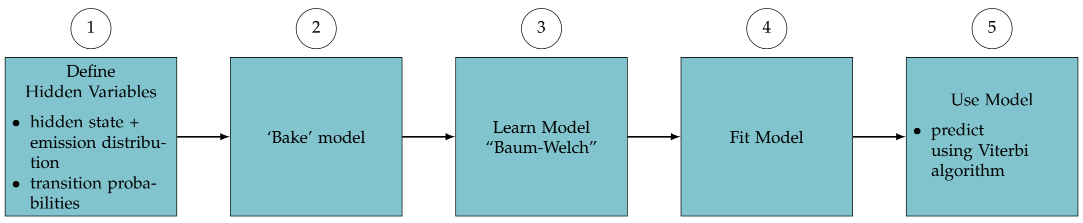

1
2
3
4
5
Define Hidden Variables
• hidden state + emission distribution
• transition probabilities
'Bake' model
Learn Model "Baum-Welch"
Fit Model
Use Model
• predict using Viterbi algorithm

FIGURE 4.8: HMM inference implementation steps for 1D and 2D sequences

Experiment 3 involved two different implementations for inference, based on 1-dimensional count 'difference' sequences and 2-dimensional 'count and notional' difference sequences. Figure 4.9a shows the set up for a HMM with three hidden states and three corresponding visible states (1D). The 2-dimensional (2D) implementation can be represented by the example in Figure 4.9b which is a HMM belief network with three hidden states, three visible states for 'count' and three for 'notional'. The main implementation steps for both 1D and 2D are depicted in Figure 4.8, a brief description is provided below, after which more detailed analysis follows:

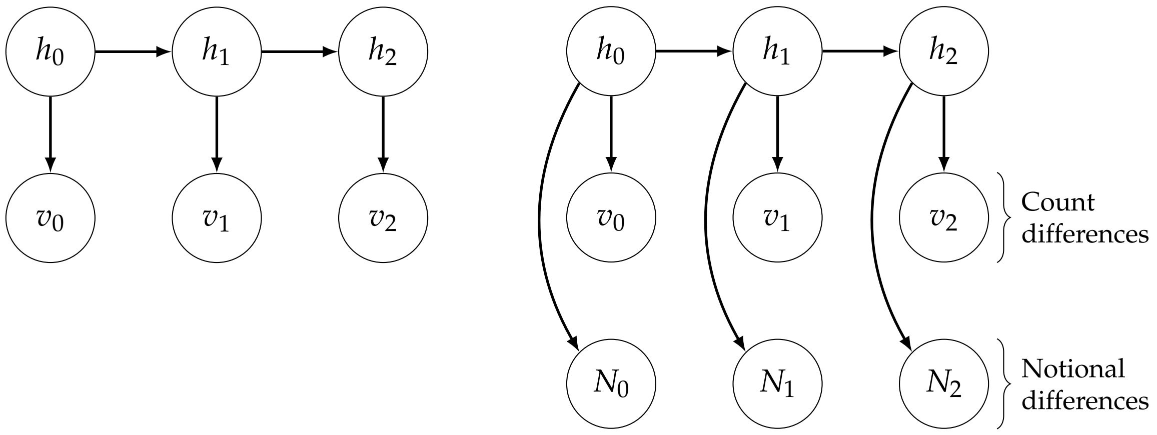
1. Define Hidden Variables – prior to training the HMM, it was necessary to define the initial values for the hidden regimes as well transition and emission matrices.

2. 'Bake' model – another term for creating the HMM using the initialised values.

3. Learn Model – the Baum-Welch EM algorithm was used in this step, which forces the hidden states to assume the emission distributions of the underlying regime in the data (increasing, decreasing, level)

4. Fit Model – the HMM is fitted on the training data sequence using the chosen hyperparameters (distribution inertia and maximum iteration)

5. Use Model – finally the 'Viterbi' algorithm is used on the the model predictions to make inferences about the most likely hidden states i.e. the RFQ market regimes.



(a) Hidden Markov model with 1-dimensional visible variables (count)

(b) Hidden Markov model with 2-dimensional visible variables ('count and notional')

FIGURE 4.9

Let us assume that our hidden states h_n are all discrete variables, which reflect the three underlying regimes of RFQ market activity. In other words the underlying market regime will always be in any one of three states: 'increasing', 'decreasing' or 'level' activity. This implies that the generation of RFQs, i.e. our 'visible' count variable, is driven by the regime the market is currently in.

The transformation of RFQ data into a suitable state is absolutely critical for successful inferential modelling using the HMM. As mentioned previously, unlike the supervised models there is no need to transform RFQs into labelled data sets of 'X' and 'y' for the HMM as it can deal with 'un-labelled' sequences. However, we do need to ensure that the 'visible' RFQ data is formulated in such a way which fits the HMM assumptions. The HMM must obey the Markov assumption, i.e. the future depends on the present state and assumes that is a sufficient representation of history. That is another way of saying the present state is sufficient to decide what comes next. Therefore we are able to draw it as per the examples setup in Figure 4.9a which is a belief network embodiment of the Markov assumption where the relationship below holds:

$$v_2 \perp\!\!\!\perp v_1 | h_2 \tag{4.10}$$

Expression 4.10 means given 'hidden' state h_2, our 'visible' variable v_2 is independent of the past v_1. With this knowledge, the first attempt was to model the RFQs naively using the HMM. Figure 4.10 below shows a series of RFQs over time with 3 labelled as v_0, v_1 and v_2, all three of which are in an increasing regime with corresponding hidden states for each.

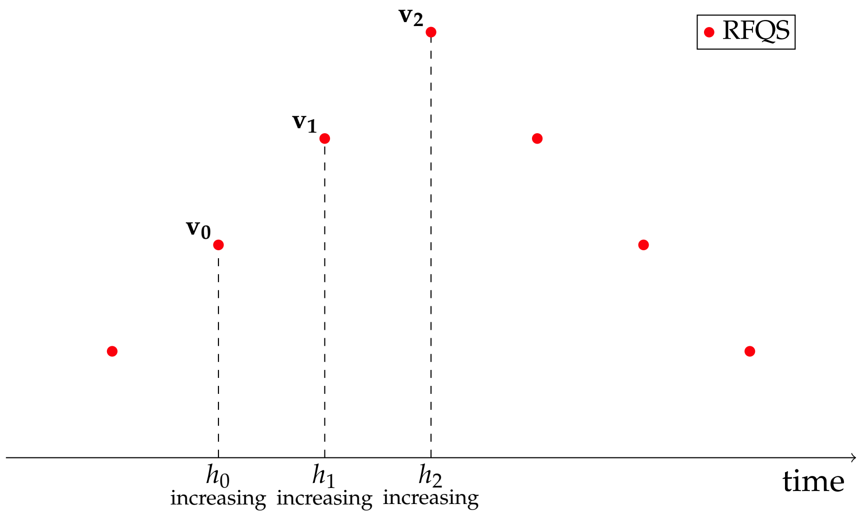


FIGURE 4.10: RFQs in an 'increasing' regime

4.4.2.1 'Differences' methodology

Next it is important to ascertain if our real world knowledge of the RFQ data is aligned with our knowledge of a HMM. i.e. does the RFQ data obey the Markovian assumption of $v_2 \perp\!\!\!\perp v_1 | h_2$ imposed by our HMM?

Figure 4.10 clearly shows that the Markovian assumption does not hold and that v_2 is not independent from v_1 given h_2 i.e.: $v_2 \not\!\perp\!\!\!\perp v_1 | h_2$. The reason is that given h_2 is in an increasing regime, v_2 is highly likely to be greater than v_1. Therefore, because knowing v_1 tells us something about v_2 this means they are not independent. Hence, we propose to take a 'differences' approach. Transforming our RFQ data into a 'differences' universe is much more likely to make v_2 independent from v_1 given the regime we are in.

Therefore, the 'differences' approach is much more suitable for a HMM as the Markov assumption holds.

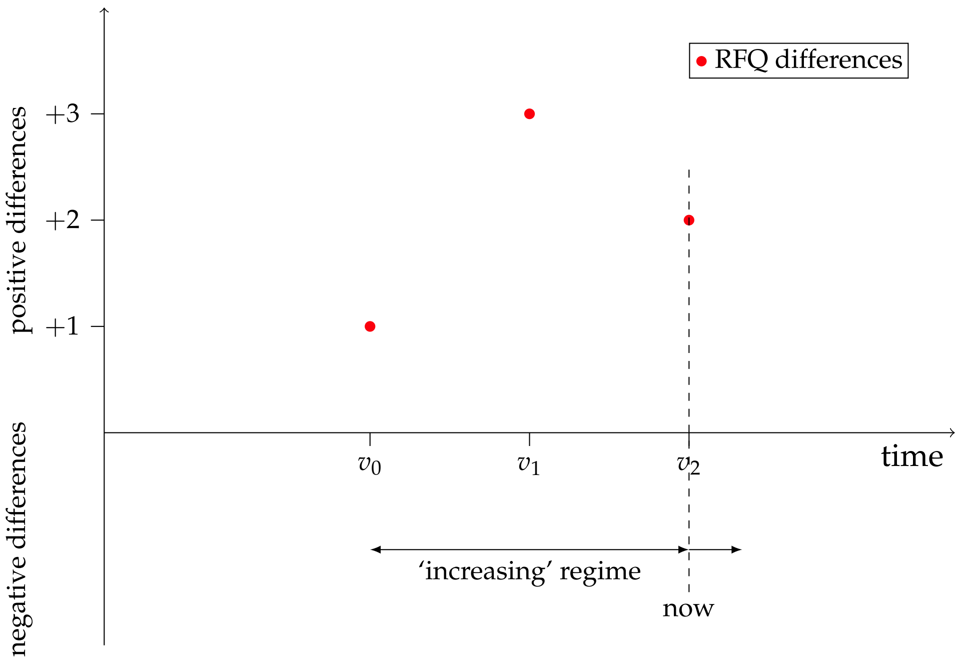


FIGURE 4.11: RFQs in an 'increasing' regime post application of 'differences' method

Figure 4.11 demonstrates the effect of taking 'differences'. Assuming visible variables v_{0-1}, v_0, v_1 and v_2 have RFQ count values of 1, 2, 5 and 7 respectively we can plot the 'differences' between them per Figure 4.11. Examining the differences closely, knowing that v_0 has a 'difference' of $+1$ and given that we are in an increasing regime, it doesn't actually tell us what v_1 is going to be. On the other hand, if we don't take the 'differences', given an increasing regime, we can predict v_1 is going to be greater than v_0 which is not very profound. Therefore we have demonstrated that the naïve approach is simply not suitable for the HMM.

4.4.2.2 Implementation detail

In order to 'bake' (create) the HMM, we need to initialise certain values and make some assumptions upfront. The hidden states h_0, h_1, h_2 and their corresponding three distributions were initialised as per Table 4.4. The EM algorithm will force the hidden states to assume the emission distributions of three phenomena it finds in the data i.e. the market regimes of: increasing, decreasing and level.

TABLE 4.4: Initialisation values for emission distributions for each regime

Hidden States	**Regime Initialisation**	**Emission Distribution Initialisation**
h_0	'decreasing'	$e_0 \sim N(-5, 5)$
h_1	'increasing'	$e_1 \sim N(5, 5)$
h_2	'level'	$e_2 \sim N(0, 5)$

In addition, prior to training it is also necessary to specify a prior along with the transition matrices (see Table 4.4). Hidden states h_0, h_1, h_2 were assigned a flat prior of 0.333 each and the transition matrix was also created per Table 4.5. For example the first entry in the table for $h_0 \rightarrow h_0$ means the next state is the same as the current state with the probability of 0.8. The EM algorithm will adapt these probability values, but these initial values capture the observation that prevailing market conditions tend to persist.

TABLE 4.5: Transition probabilities between states

Transition	**Probabilities**
$h_0 \rightarrow h_0$	0.8
$h_0 \rightarrow h_1$	0.1
$h_0 \rightarrow h_2$	0.1
$h_1 \rightarrow h_0$	0.1
$h_1 \rightarrow h_1$	0.8
$h_1 \rightarrow h_2$	0.1
$h_2 \rightarrow h_0$	0.1
$h_2 \rightarrow h_1$	0.1
$h_2 \rightarrow h_2$	0.8

Once the model has been initialised, we generate the differences data based on the RFQ counts. The data is split into 2 sub-sets of train and test (90%–10%) and the train data split further for cross-validation (70%–30%). When training the model during the cross-validation process, a new model is created at each training step. The results are obtained, the model is evaluated and then discarded. Subsequently a new model is created thus preventing any pollution of results from the previous round of training. Due to the nature of sequence data and the unsupervised model, a bespoke cross-validation process was created for this experiment.

The HMM in Experiment 3 has two hyperparameters: 'distribution inertia' and 'maximum iteration'. The cross-validation process searched for the best hyperparameter values which will be presented in the results section of the report.

The HMM was then re-fitted to the training data using the best hyperparameters found, as well as the corresponding transition matrix for those.

Once the model has been re-fitted using the EM algorithm we can get information about each state and its associated emission matrix distribution (e.g. in our case 'NormalDistribution'), the mean and standard deviation as well as whether it is increasing, decreasing or level.

Finally the 'Viterbi' algorithm is used on the test data to make HMM predictions, where the algorithm asks the question: what was the most likely hidden state ('regime') of the variable given the data? The HMM was able to learn the regimes (level, increasing, decreasing) from the data without being explicitly told - see 'Viterbi' predictions in Figures 4.13 and 4.14

4.4.3 Results and discussion

In Experiment 3 the HMM has two hyperparameters, namely: 'distribution inertia' and 'maximum iteration'. The values found for those are displayed in Table 4.6. No differences were observed between hyperparameters for 1D and 2D data.

The objective of Experiment 3 is to make inferences about the most likely regime of the hidden states by using the Viterbi algorithm on the HMM.

Figure 4.12 demonstrates the expected Viterbi prediction for the implementation of the HMM in Experiment 3. The chart (in blue) of count versus bin dates displays a count distribution displayed over three days. Visibly there are different regimes over these three days when the count is increasing, decreasing or perhaps level.

The Viterbi prediction below Figure 4.12 displays the most likely hidden state of the variables given the sequence of data. If the most likely hidden state corresponds to a level regime, then the Viterbi flags this as a '2'. Similarly, increasing and decreasing regimes are flagged as '1' and '0' respectively.

Figure 4.13 and 4.14 display the actual Viterbi predictions run on a sequence of 1D and 2D data respectively. The sequences used were 'differences' which for illustration purposes were aligned to the RFQ count distribution in Figure 4.12.

Figure 4.13 clearly shows that for the 1D case the Viterbi algorithm run on the hidden Markov model is able to give the correct insight about the

RFQ regime. The increases and decreases align well to the visible RFQ behaviour. Equally Figure 4.14 demonstrates the 2D case (using both count and notional). This approach provides more granularity during 'increasing' RFQ regimes and overall aligns well to the underlying visible RFQs. However, the 2D case does not predict well for the tail of the last count distribution peak where it predicts an 'increasing' regime even though the underlying data is decreasing.

TABLE 4.6: Experiment 3 – The HMM hyperparameters chosen

Hyperparameters	1D and 2D
Distribution Inertia	0.01
Maximum Iteration	10.0

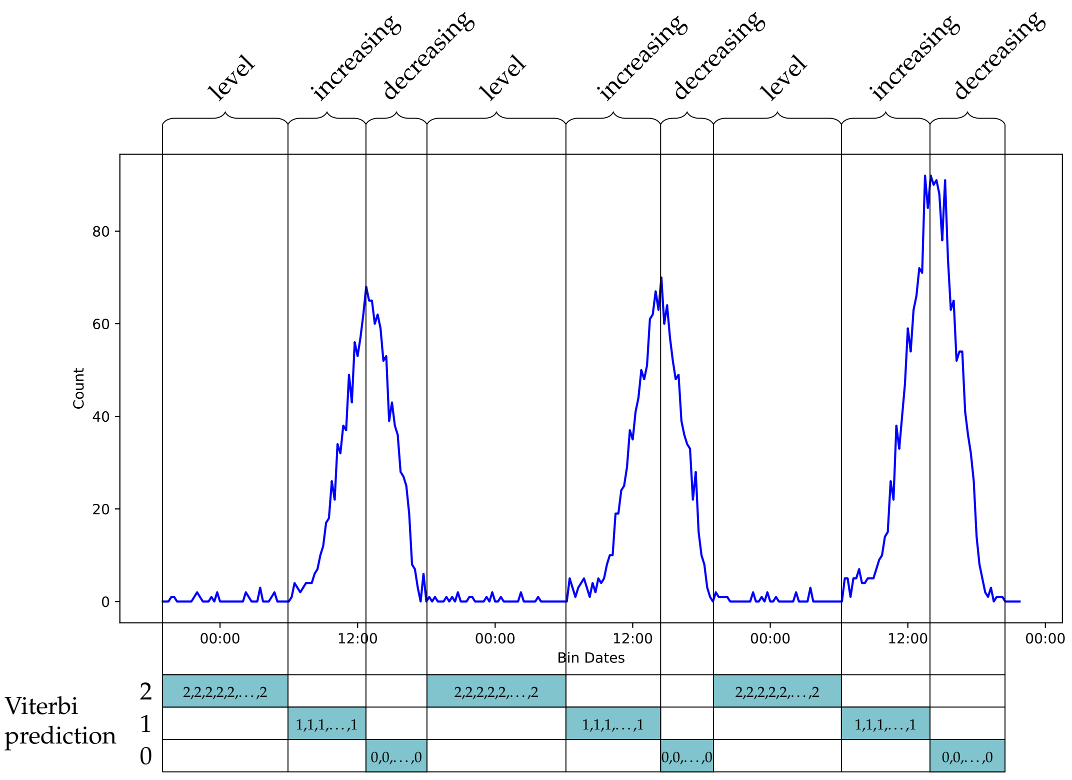


FIGURE 4.12: Experiment 3 – Viterbi algorithm prediction aligned to a 3 day RFQ distribution

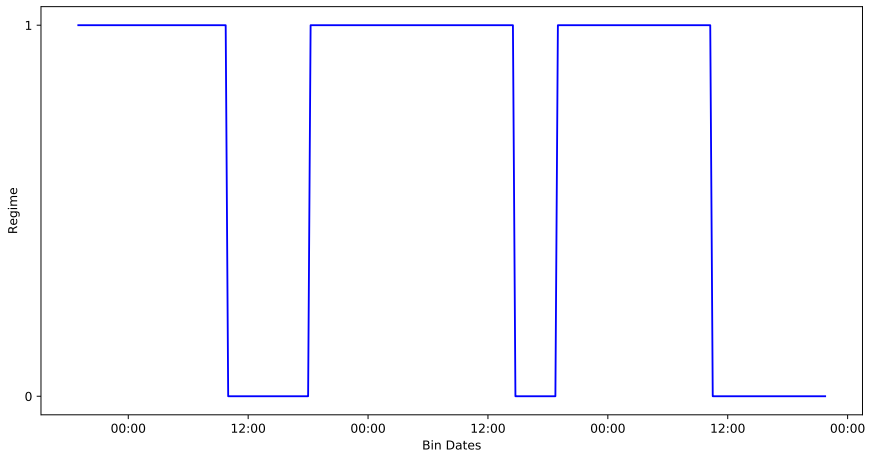


FIGURE 4.13: Viterbi algorithm prediction on 1D sequence

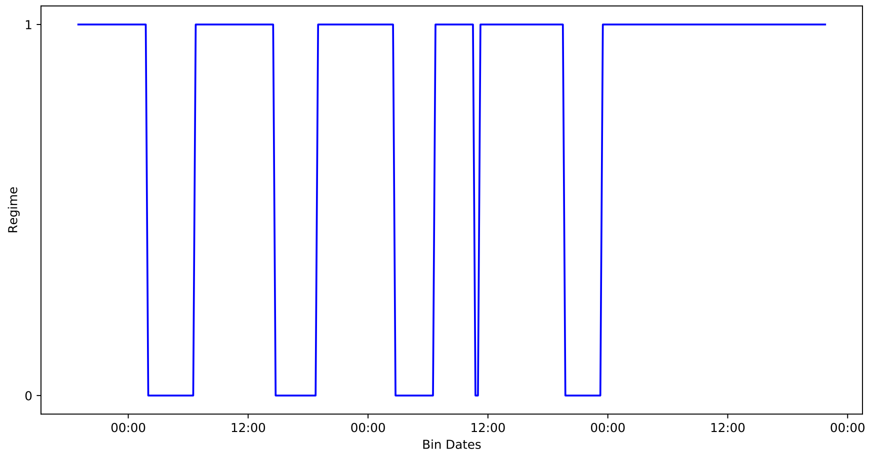


FIGURE 4.14: Viterbi algorithm prediction on 2D sequence

4.4.4 Conclusion

The objective of Experiment 3 was to use the Bayesian hidden Markov model to infer and learn the hidden states of the underlying visible RFQ data. This objective was met with good success for both 1D and 2D cases. The ‘Viterbi’ algorithm was used to make predictions of the hidden states based on the ‘differences’ method. The RFQ regimes identified by the ‘Viterbi’ algorithm aligned well with the observed data, and clearly indicated the changing regimes for both 1D and 2D sequences.

5 Conclusion

This thesis set out to investigate the predictability of e-FX option RFQs using the two different philosophical approaches to machine learning, namely Classical and Bayesian. The investigation was carried out by conducting three experiments on supervised and unsupervised models which straddled both philosophies.

The importance of this thesis is validated by the intelligence it is able to provide to the sales and trading outfits of investment banks. Overall, this thesis managed to predict RFQs with good success using both Classical and Bayesian techniques.

The Classical models used in **Experiment 1** were a feed-forward neural network (MLPregressor) and a least squares model with regularisation known as Ridge Regression. Also, in **Experiment 1** the Bayesian Ridge Regression model was used as the Bayesian equivalent of Ridge Regression. Bayesian Ridge Regression is Bayesian in terms of its parameters but not the data, which were labelled. Finally, the hidden Markov model (HMM) was used as a fully Bayesian model of an unsupervised nature.

In **Experiment 3** the HMM was explored to a much deeper level, demonstrating its ability to give insight on an underlying, unobservable process (RFQ regime), using the observable RFQ data. The HMM was also used to make predictions, via the hidden states (**Experiment 2**). This is a very different method of making predictions from the Classical models examined in this thesis, as it performs the predictions via inferring the hidden states. In some ways it could be argued that this is a more insightful prediction method than, say, regression, as it is attempting to model the underlying phenomenon rather than just matching patterns per **Experiment 1**.

Models like the HMM that treat the data in a Bayesian manner also have the advantage that they have a principled way of treating missing values, by simply normalising those variables out so they are not considered. On the other hand, Classical methods typically need to select a method of data imputation to handle missing data which may impact the results.

6 Suggestions for Further Work

This investigation produced promising results whilst pursuing its objectives. A list of possible improvements and suggestions for future work is as follows:

- Further extend the investigation of feed-forward neural networks into recurrent neural networks.
- Investigate a larger RFQ data set over a longer time horizon, with multiple currency pairs and FX option types. This would allow for more specific predictions, for example: type of instrument, the currency pair, the deal size as well as the specific client RFQ source.
- Further investigate the performance of the models chosen in this thesis, by incorporating more features. These could include:
  - Inclusion of market data such as spot and forward prices as well as different components of FX volatility surfaces could be explored.
  - Addition FX option deal features such as strike, premium and expires etc.
  - Exploring macro features, for example: the impact of political and economic news on RFQ activity.
  - Assessing the impact of different time-series 'bin' widths.
- Finally, this study was performed from the view point of a single financial institution. It would be interesting to explore client behaviour towards different suppliers. One could then identify opportunities when one financial institution is preferred to another, thus informing the institution on corrective actions to improve their offering.

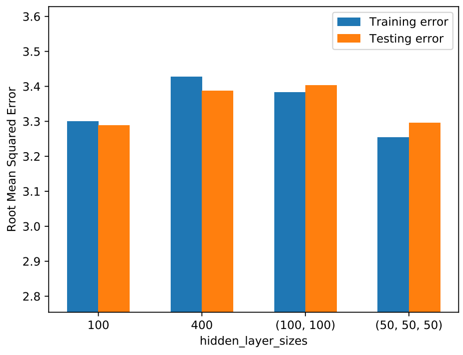
A Results

TABLE A.1: Experiment 1 – results table for RFQ counts and notionals

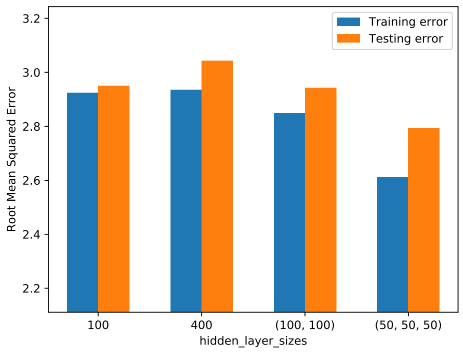
Model	History Length	RMS Eorr Train	RMS Error Test	R^2 Score Train	R^2 Score Train
MLPRegressor	Count and Notional 30 mins	4.09	4.17	0.98	0.97
	Count and Notional 1 hour	2.93	3.16	0.99	0.98
	Count and Notional 2 hours	2.28	2.80	0.99	0.99
	Count and Notional 4 hours	1.96	2.64	0.99	0.99
Ridge Regression	Count and Notional 30 mins	4.22	4.21	0.97	0.97
	Count and Notional 1 hour	3.37	3.47	0.98	0.98
	Count and Notional 2 hours	3.15	3.28	0.99	0.98
	Count and Notional 4 hours	3.10	3.28	0.99	0.98
Bayesan Ridge Regression	Count and Notional 30 mins	4.22	4.21	0.97	0.97
	Count and Notional 1 hour	3.37	3.47	0.98	0.98
	Count and Notional 2 hours	3.15	3.28	0.99	0.98
	Count and Notional 4 hours	3.11	3.28	0.99	0.98

TABLE A.2: Experiment 1 – hyperparameters chosen for model evaluation of the historical RFQ count data

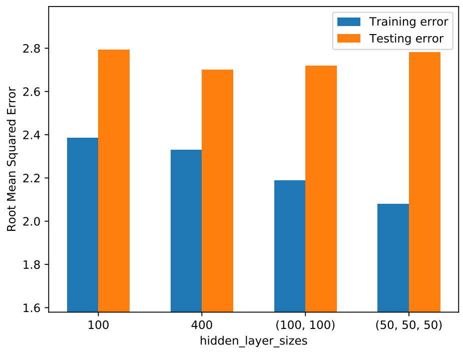
	MLP Regressor Hyperparameters		Ridge Regression Hyperparameters	Bayesian Ridge Regression Hyperparameters			
RFQ HistoryLength	hidden layer size	initial learning rate	alpha	alpha_1	alpha_2	lambda_1	lambda_2
Count and Notional 30 mins	100, 100	0.000316	335.16	1×10^{-10}	1×10^{10}	359,381.37	1×10^{10}
Count and Notional 1 hour	50, 50, 50	0.000316	335.16	1	1×10^{5}	359,381.37	1×10^{5}
Count and Notional 2 hours	100, 100	0.000316	1,353.05	1×10^{-10}	1×10^{5}	12.92	1
Count and Notional 4 hours	100, 100	0.000316	2,154.43	1	1×10^{10}	1×10^{-10}	1×10^{-5}



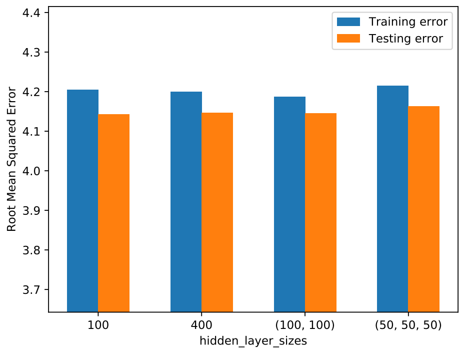
(a) RMS error vs. architecture for 1 hour count history



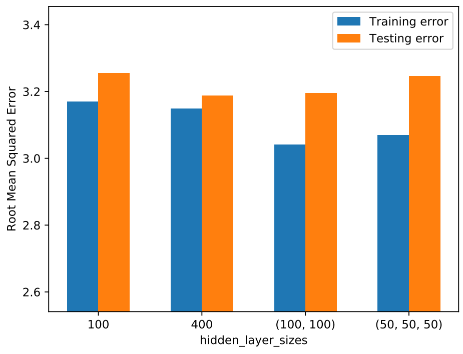
(b) RMS error vs. architecture for 2 hours count history



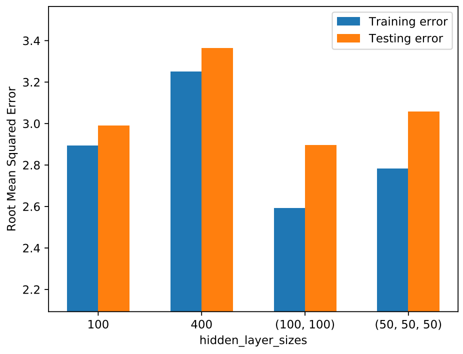
(c) RMS error vs. architecture for 4 hours count history



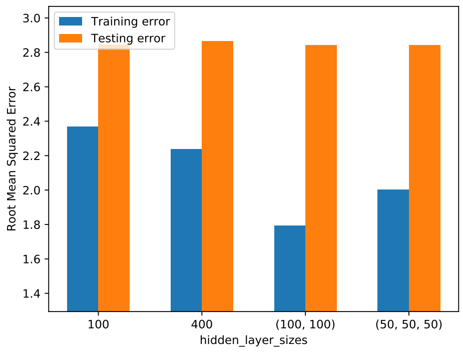
(d) RMS error vs. architecture for 30 mins count history



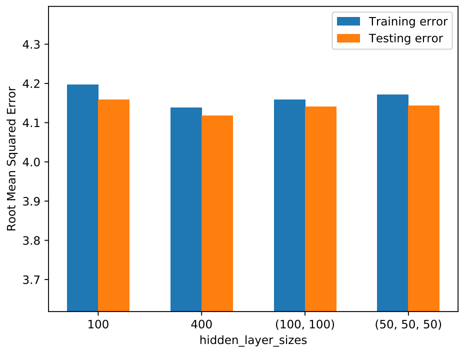
(e) RMS error vs. architecture for 1 hours count and notional history



(f) RMS error vs. architecture for 2 hours count and notional history

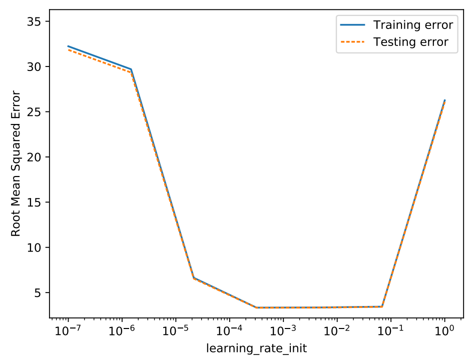


(g) RMS error vs. architecture for 4 hours count and notional history

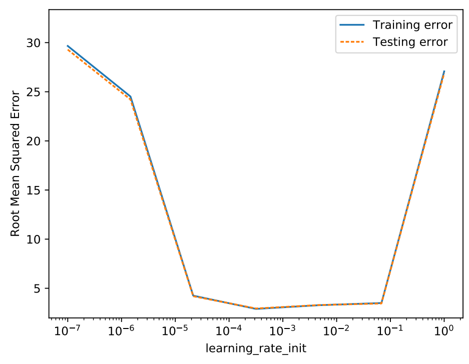


(h) RMS error vs. architecture for 30 mins count and notional history

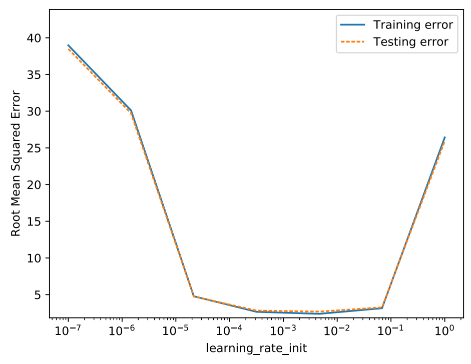
FIGURE A.1: Experiment 1 – MLPRegressor RMS error vs. network architecture plots



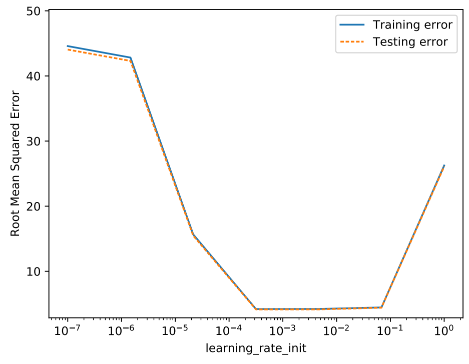
(a) RMS error vs. initial learning rate for 1 hour count history



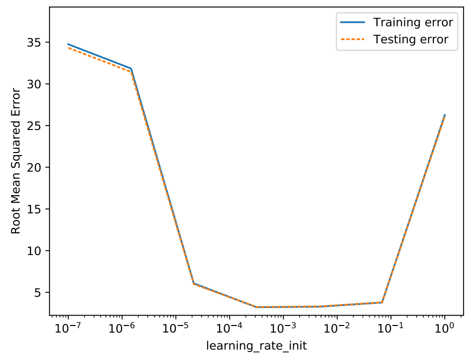
(b) RMS error vs. initial learning rate for 2 hours count history



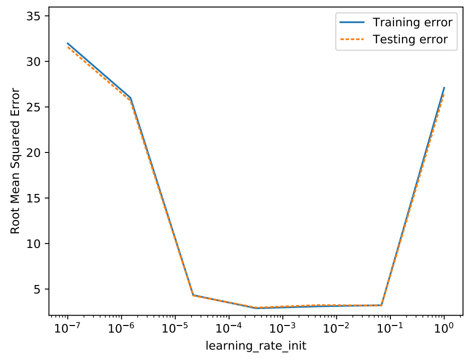
(c) RMS error vs. initial learning rate for 4 hours count history



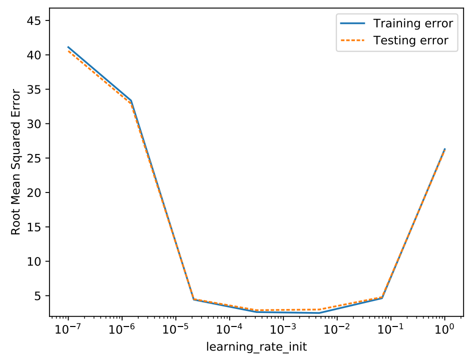
(d) RMS error vs. initial learning rate for 30 mins count history



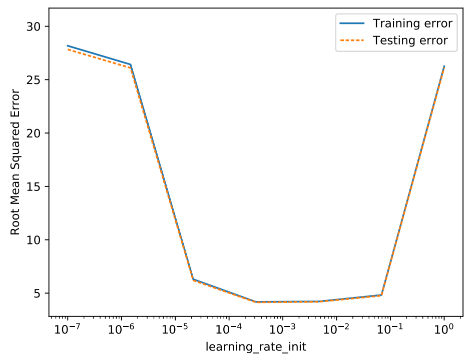
(e) RMS error vs. initial learning rate for 2 hours count and notional history



(f) RMS error vs. initial learning rate for 2 hours count and notional history

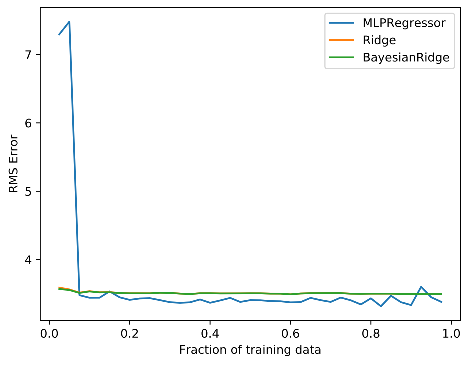


(g) RMS error vs. initial learning rate for 4 hours count and notional history

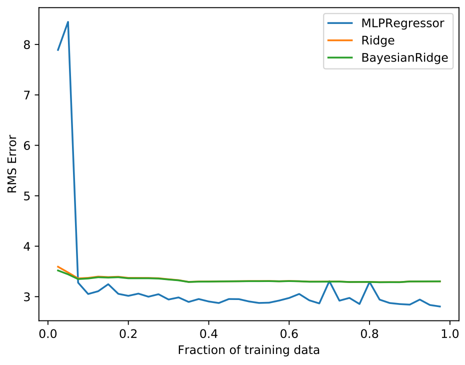


(h) RMS error vs. initial learning rate for 30 mins count and notional history

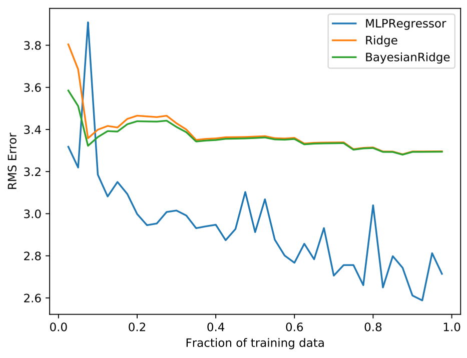
FIGURE A.2: Experiment 1 – MLPRegressor plots of RMS error sensitivity to initial learning rate



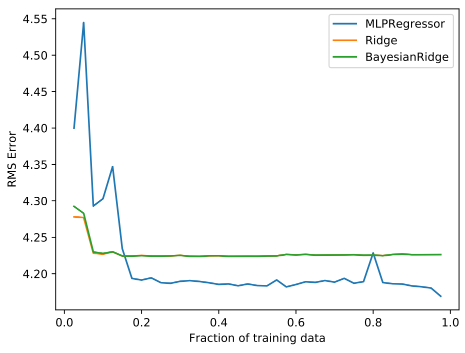
(a) RMS Test error vs. size of training data for 1 hour count



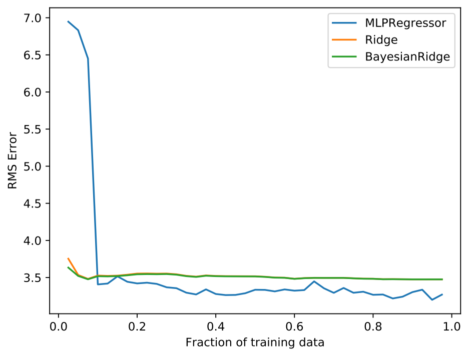
(b) RMS Test error vs. size of training data for 2 hours count



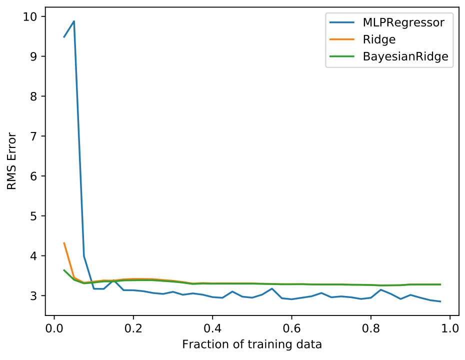
(c) RMS Test error vs. size of training data for 4 hours count



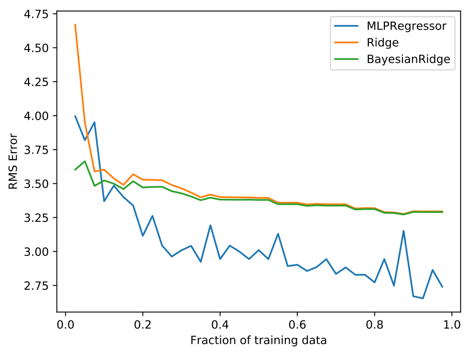
(d) RMS Test error vs. size of training data for 30 mins count



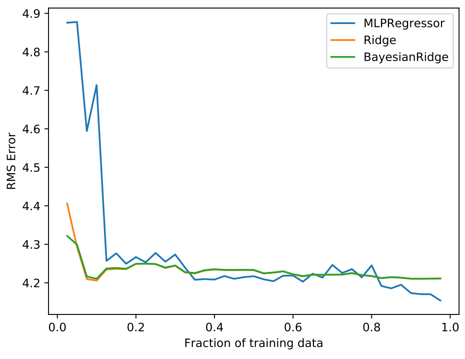
(e) RMS Test error vs. size of training data for 1 hour count and notional



(f) RMS Test error vs. size of training data for 2 hours count and notional

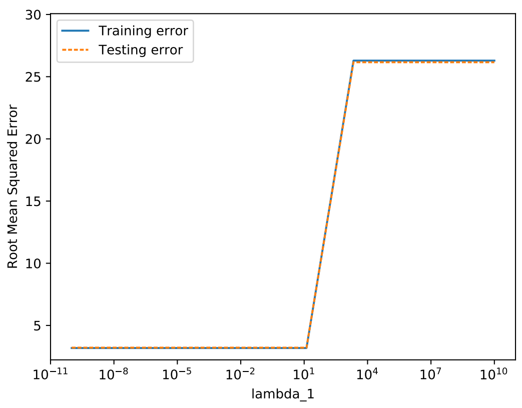


(g) RMS Test error vs. size of training data for 4 hours count and notional

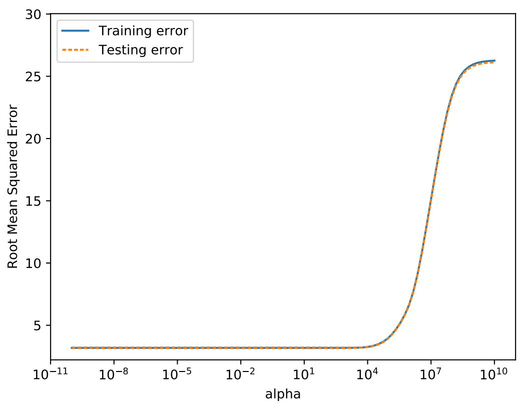


(h) RMS Test error vs. size of training data for 30 mins count and notional

FIGURE A.3: Experiment 1 – RMS Test error for each model vs. size of training data



(a) RSM error sensitivity vs. lambda_1 hyperparameters for Bayesian Ridge



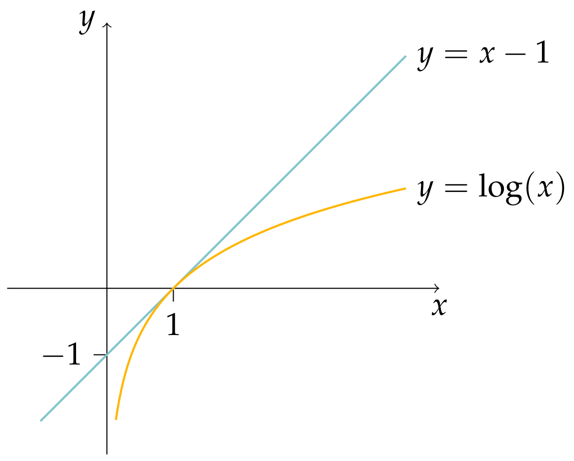
(b) RSM error sensitivity vs. alpha hyperparameters for Ridge

FIGURE A.4: RSM error sensitivity vs. hyperparameters for Bayesian Ridge and Ridge

B KL Divergence Non-negative Proof

Proof: KL divergence ≥ 0

1. Consider a function $y = \log(x)$ bound by $y = x - 1$ per below:



2. It follows that:

   $\log(x) \leq x - 1$

   with equality at $x = 1$ (note: $\frac{\mathrm{d}y}{\mathrm{d}x} = 1$)

3. Let $x = \frac{p(x)}{q(x)}$

   $\Rightarrow \log \frac{p(x)}{q(x)} \leq \frac{p(x)}{q(x)} - 1$

4. Therefore:

   $\Rightarrow q \log p - q \log q \leq p - q$

5. Summing over x:

   $\Rightarrow \langle \log q \rangle_q - \langle \log p \rangle_q \leq 0$

6. Multiplying throughout by -1

   $\Rightarrow \langle \log q \rangle_q - \langle \log p \rangle_q \geq 0$

   $\equiv \mathrm{KL}(q|p) \geq 0$

C Labels Used for Average Notional Size

TABLE C.1

Average Notional	Label
0	0
$< 100{,}000$	1
$< 500{,}000$	2
$< 1{,}000{,}000$	3
$< 5{,}000{,}000$	4
$< 10{,}000{,}000$	5
$< 25{,}000{,}000$	6
$< 50{,}000{,}000$	7
$< 100{,}000{,}000$	8
$< 200{,}000{,}000$	9
$> 200{,}000{,}000$	10

D Software Packages Used

TABLE D.1: List of software packages and version used for this thesis

Package	**Version**
python	3.7.0
numpy	1.16.4
pandas	0.24.2
pomegranate	0.11.0
scikit-learn	0.21.2
matplotlib	3.1.1

Bibliography

[1] Alex Acero et al. “HMM adaptation using vector Taylor series for noisy speech recognition”. In: *Sixth International Conference on Spoken Language Processing*. 2000.

[2] Maarten HP Ambaum. “Frequentist vs Bayesian statistics-a non-statisticians view”. In: *arXiv preprint arXiv:1208.2141* (2012).

[3] Fabio Ballati, Fulvio Corno, and Luigi De Russis. “" Hey Siri, Do You Understand Me?": Virtual Assistants and Dysarthria.” In: *Intelligent Environments (Workshops)*. 2018, pp. 557–566.

[4] D. Barber. *Bayesian Reasoning and Machine Learning*. Cambridge University Press, 2012.

[5] Leonard E Baum et al. “A maximization technique occurring in the statistical analysis of probabilistic functions of Markov chains”. In: *The annals of mathematical statistics* 41.1 (1970), pp. 164–171.

[6] Dimitri P Bertsekas and John N Tsitsiklis. *Introduction to probability*. Vol. 1. Athena Scientific Belmont, MA, 2002.

[7] BIS. *Electronic trading in fixed income markets*. Tech. rep. 7. Markets Committee Publications, 2016.

[8] *BNP Paribas Cortex FX Options Trading Platform*. https://cortex.bnpparibas.com/public/fx.html. Accessed: 2019-08-26.

[9] Jack Cahn. “CHATBOT: Architecture, design, \& development”. In: *University of Pennsylvania School of Engineering and Applied Science Department of Computer and Information Science* (2017).

[10] Zhitang Chen, Jiayao Wen, and Yanhui Geng. “Predicting future traffic using hidden markov models”. In: *2016 IEEE 24th International Conference on Network Protocols (ICNP)*. IEEE. 2016, pp. 1–6.

[11] Khar Heng Choo, Joo Chuan Tong, and Louxin Zhang. “Recent applications of hidden Markov models in computational biology”. In: *Genomics, proteomics \& bioinformatics* 2.2 (2004), pp. 84–96.

[12] Aaron Clauset, Cosma Rohilla Shalizi, and Mark EJ Newman. “Power-law distributions in empirical data”. In: *SIAM review* 51.4 (2009), pp. 661–703.

[13] Paresh Date, Rogemar Mamon, and Anton Tenyakov. “Filtering and forecasting commodity futures prices under an HMM framework”. In: *Energy Economics* 40 (2013), pp. 1001–1013.

[14] Bradley Efron. “Bayesians, frequentists, and scientists”. In: *Journal of the American Statistical Association* 100.469 (2005), pp. 1–5.

[15] Christina Erlwein. “Applications of hidden Markov models in financial modelling”. PhD thesis. Brunel University, School of Information Systems, Computing and Mathematics, 2008.

[16] Zoubin Ghahramani. "An introduction to hidden Markov models and Bayesian networks". In: *Hidden Markov models: applications in computer vision*. World Scientific, 2001, pp. 9–41.

[17] David Gil et al. "Application of artificial neural networks in the diagnosis of urological dysfunctions". In: *Expert systems with applications* 36.3 (2009), pp. 5754–5760.

[18] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT press, 2016.

[19] J Hu, MK Brown, and W Turin. "HMM Based On-Line Handwriting Recognition". In: *IEEE Trans. Pattern Analysis and Machine Intelligence* 18.10 (), pp. 1–039.

[20] Zan Huang et al. "Credit rating analysis with support vector machines and neural networks: a market comparative study". In: *Decision support systems* 37.4 (2004), pp. 543–558.

[21] Gareth James et al. *An introduction to statistical learning*. Vol. 112. Springer, 2013.

[22] Andreĭ Nikolaevich Kolmogorov and Albert T Bharucha-Reid. *Foundations of the theory of probability: Second English Edition*. Courier Dover Publications, 2018.

[23] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. "Imagenet classification with deep convolutional neural networks". In: *Advances in neural information processing systems*. 2012, pp. 1097–1105.

[24] Yassine Lassoued et al. "A hidden Markov model for route and destination prediction". In: *2017 IEEE 20th International Conference on Intelligent Transportation Systems (ITSC)*. IEEE. 2017, pp. 1–6.

[25] Warren S McCulloch and Walter Pitts. "A logical calculus of the ideas immanent in nervous activity". In: *The bulletin of mathematical biophysics* 5.4 (1943), pp. 115–133.

[26] Mckinsey&Company. *Two Routes to Digital Success In Capital Markets*. Tech. rep. 10. McKinsey Working Papers on Corporate & Investment Banking, 2015.

[27] Michael Moore, Andreas Schrimpf, and Vladyslav Sushko. "Downsized FX markets: causes and implications". In: *BIS Quarterly Review December* (2016).

[28] Kevin P Murphy. *Machine learning: a probabilistic perspective*. MIT press, 2012.

[29] Jeremy Orloff and Jonathan Bloom. "Comparison of frequentist and Bayesian inference". In: *Massachusetts Institute of Technology. Recuperado de https://ocw. mit. edu/courses/mathematics/18-05-introduction-to-probability-and-statistics-spring-2014/readings/MIT18_05S14_Reading20. pdf* (2014).

[30] F. Pedregosa et al. "Scikit-learn: Machine Learning in Python". In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830.

[31] Sam Ransbotham et al. "Reshaping business with artificial intelligence: Closing the gap between ambition and action". In: *MIT Sloan Management Review* 59.1 (2017).

[32] Tim Rockt\"aschel and Sebastian Riedel. "End-to-end differentiable proving". In: *Advances in Neural Information Processing Systems*. 2017, pp. 3788–3800.

[33] Frank Rosenblatt. "The perceptron: a probabilistic model for information storage and organization in the brain." In: *Psychological review* 65.6 (1958), p. 386.

[34] Mario Stanke and Stephan Waack. "Gene prediction with a hidden Markov model and a new intron submodel". In: *Bioinformatics* 19.suppl_2 (2003), pp. ii215–ii225.

[35] Anton Tenyakov and Rogemar Mamon. "A computing platform for pairs-trading online implementation via a blended Kalman-HMM filtering approach". In: *Journal of Big Data* 4.1 (2017), p. 46.

[36] Anton Tenyakov, Rogemar Mamon, and Matt Davison. "Modelling high-frequency FX rate dynamics: A zero-delay multi-dimensional HMM-based approach". In: *Knowledge-Based Systems* 101 (2016), pp. 142–155.

[37] *UBS Neo FX Options Trading Platform*. https://about-neo.ubs.com/content/fxoptions. Accessed: 2019-08-26.

[38] John FW Zaki et al. "Framework for Traffic Congestion Prediction". In: *International Journal* 7 (2016), p. 6.

[39] Yingjian Zhang. "Prediction of financial time series with Hidden Markov Models". PhD thesis. Applied Sciences: School of Computing Science, 2004.