Birkbeck College, University of London

MSc. Data Science Master Thesis

The Applicability of Machine Learning Methods For Currency Trading

Author
Edward Gill

Supervisor
Dr. George Magoulas

A thesis submitted in fulfillment of the requirements
for the degree of MSc Data Science
in the

Department of Computer Science and Information Systems

September 15, 2019

Abstract

The availability of immense computing power has opened a veritable Pandora's box of highly complex and computationally intensive models which can now be applied to any dataset, whether appropriate or not. While machine learning models have led to rapid and impressive use cases across a host of different fields of research, this project will focus on the efficacy of machine learning models in global financial markets and specifically currency markets.

Applying complex methods within the financial domain requires a different approach from other areas such as medical research or consumer behaviour forecasting given the inherent non-stationarity and noisiness of financial time series. The aim of this project is to build on existing research and try to gauge the efficacy of applying machine learning methods to the investment process. The motivation for this thesis is to try to cut through some of the recent hype generated in financial media and try to uncover how machine learning may best be used by financial practitioners.

Contents

1 Introduction

The velocity of technological change that society has encountered since the turn of the millennium is striking, the world has embarked on a computational revolution which has brought the availability of immense processing power to the masses. Computational power at a scale that government sponsored computer science programs of the early nineties could only dream of. In fact, the processing power of most smartphones are larger than the supercomputers of yesteryear [1].
Moore's Law, the well known law which states that the number of transistors in a dense integrated circuit doubles about every two years, has held for over fifty years [2].

The allure of applying machine learning methods to finance is well entrenched, where researchers dream of uncovering repeatable patterns within the data which guarantees consistent alpha generation (returns unexplained by conventional market beta factors [3]). A quick search of Google, another revolution of big data processing and machine learning algorithms, uncovers a vast array of academic papers which profess of having found patterns in the data that could only have been uncovered by complex models with non-linear capabilities.

Using machine learning models in conjunction with big data can and do provide fantastic results in the area of the biological sciences or as recommender systems for movies or advertisements. However applying these same methods to financial markets is a different and more challenging task due to the underlying unpredictability of asset prices.

This paper delves into the world of financial machine learning and explores certain techniques to try to uncover how machine learning methods can be best applied to financial markets.

1.1 Project Goal

The main goal of this project is

- To create a fully systematic trading algorithm which uses various technical price based features to capture the trend in an asset price timeseries.
- To overlay a trend capture model using a separate machine learning model to identify regimes which are conducive to the positive performance of trend models.
- Finally to test and validate this particular trading model architecture across a host of different parameter variations, testing timeframes and features to thoroughly test the models generalisation capability.

The methodology behind creating prediction models in financial markets will differ vastly from similar projects in non financial domains, financial markets are inherently unstable and contain a substantial amount of noise which makes applying complex algorithms in finance susceptible to out of sample failure [4].

The task of uncovering repeatable patterns in financial data remains incredibly precarious, financial markets are non-stationary, exhibit leptokurtic (fat tailed) distributions and can shift regimes at highly unpredictable speeds. Much has been written on the complexity of financial markets and how researchers must overcome such dynamic systems [5]. While the explosion of increased computational power aided the application of machine learning in very useful tasks such as speech recognition and image processing, the applicability of these methods in finance are, on the face of it at least, questionable [6], thus this project has placed much greater emphasis

on the roubustness testing of the selected features and models, while also trying to keep the underlying features as intuitive and simple as possible.

1.2 Project Objectives

The main aim of the project is to understand how machine learning methods can be applied to currency forecasting and the classification of macroeconomic regimes in which a trend trading strategy may perform best.

The project attempts to outline best practices to use when applying machine learning techniques to currency trading. As the project progressed, it was clear that feeding the model masses of data in the hope of finding a successful pattern would not generalise very well in the future.

Much has been written about the lack of verifiable and replicable results in machine learning research which suggests the capacity to overfit is high [7]. Hence the approach taken in this project was to assign one machine learning model with a very narrow goal, not to predict the next days prices based on a host of different technical and fundamental features, but to simply capture the *trend* in the price series.
The difference between a model created to capture trend vs a model created to predict the next day or week ahead price is subtle but important.

Firstly, trend can be captured in very simple terms with few features (thus reducing any dimensionality issues) while it is also not reliant on the stability of correlations between non price data (such as economic indicators) and the price series itself. The only goal of a trend capture model is to identify the direction of the trend in the price series with the expectation that the machine learning model can identify the non linearities inherent in trend reversals which more simple linear trend models may not be able to capture. Thus the problem domain that the model needs to assist in solving becomes not one of outright price prediction but of trend prediction. Thus if there exists auto correlation in the price series, then the model is able to identify this trend and trade in the direction of that trend, while if no trend exists (or the asset price is exhibiting a mean reverting behaviour) then the model is expected to underperform.

This is where the second model is assigned the task of using economic and fundamental data to try to identify when the underlying price series has a higher probability of exhibiting sustained auto-correlation. Thus the key here to profitability , is not the ability to forecast the underlying price series on a day to day basis, but to create a strategy which has an asymmetric risk profile , where in trending markets the combinations of models perform well while in mean reverting regimes the second model reduces or switches off the first trend capture model and by corollary reduces the severity of losses. The appeal of this type of model architecture is that it is price series agnostic, such that it should work as well on currency data as it should on equity price data. This means one could relatively easily create a cross asset trading model which would likely mimic the returns of a Commodity Trading Advisor (CTA), a type of hedge fund which profits from trending price series.

The main objectives in order to achieve the stated aim of the projects are shown below.

1.2.1 Data Retrieval, Cleaning and Standardisation

- One of the most important aspects of a trading model is to ensure that the underlying data is of high quality and doesn't contain erroneous values. The proposed model will use hourly spot exchange rates for G10 currencies [1], various economic data from the Federal

[1] Hourly FX Spot Data Provided by Citi Foreign Exchange

Reserve Economic Database (FRED), the International Monetary Fund (IMF) and the Organisation for Economic Cooperation and Development (OECD) will be used to try to uncover exploitable patterns in the data.

- This data will also need to be cleaned and mapped to correct date scales so that the model only uses data available at the time of trade signal generation. The raw data itself will need to be standardised to allow the model to understand the true relationships between the feature vectors and the output.

1.2.2 Feature Extraction

- Feature extraction for financial time series prediction is a very challenging task due to the inherent noise in the data. Including features which have an intuitive justification for predicting future prices is important [8]. Feature selection driven by mining the training set greatly increases the risk of overfitting and should be avoided when working with financial times series.

1.2.3 Model Training and Architecture Selection

- The project itself will test the efficacy of a range of algorithms in trying to predict currency movements, this project will also make use of non price data in order to assist the model in understanding potential regime changes in the trading environment.

1.2.4 Model Validation and Strategy Backtesting

- Once a model architecture has been selected, it is important to understand not only classification rates but also how the model would perform in a live market setting. This can be approximated by creating a backtesting 4.5 framework which depicts the model's trading performance and also includes the impact of transaction costs.

1.2.5 Testing on Fully Withheld Data

- While the test data allows us to understand how the model performs, there will also be an opportunity to analyse how the selected model will work on new unseen data which provides a useful guide as to how the model will generalise in the future.

1.3 Project Organisation

The project is structured as followed,

- Section 2 provides a description of the underlying problem of applying machine learning (ML) to financial timeseries.
- Section 3 provides an overview of the latest research being undertaking in the financial machine learning domain.
- Section 4 dives into the data set needed and the specifications of the system architectures used in this project alongside the model features used and the reason for their selection.
- Section 5 delves into the core of the model structure and how two separate trading architectures are combined to create the a full stack trading strategy.

- Section 6 analyses the performance metrics of each model and discusses the main takeaways of the results and offers thought on the direction for future work that should be undertaken when applying machine learning models to financial trading.
- Section 7 concludes with final thoughts on the project overall and highlights some interesting points which were revealed as the project progressed.

2 Description of the Problem

The problem to overcome when applying ML to trading is the level of signal to noise in the data. ML practitioners in finance face a very challenging task as the past is rarely a perfect guide to the future. For example in image processing, if a neural network is fed a matrix of pixels which form the image of cows, and then trains based on the historical patterns it observes, the trained neural network will be able to identify a cow with high classification rates. However, if we provide an image of a mouse, the model likely classifies the mouse as a small cow due to the historical patterns that have been learned. Machine learning models, no matter how complex or sophisticated the underlying features are, require regularity for prediction.
In finance, you can learn all the historical patterns and probabilities surrounding an asset and the model may even perform well in the testing period, however, in finance market regimes can change rapidly. Structural breaks can and do occur in past relationships. Even the most robust artificially intelligent machines would likely not survive the impact of a 280 character tweet from Donald Trump entirely changing the rules of the game. Thus, the approach taken in this project is to narrow the scope of the problem which the ML model is required to solve. This project sets the main trading model a narrowly defined task to recognise the patterns of a trending asset price only. This could be considered akin to using ML models in finance like a surgical scalpel rather than a hammer for every nail.

2.1 Fickle Correlations in Financial Markets

Correlations and patterns in markets can be extremely fickle, complex and dynamic in nature [9]. Why is this so one may ask? Well part of the issue lies in the efficiency of informational flow across markets, barriers to entry are much smaller than ten or twenty years ago and there has likely been a compression in the informational edge that smart money investors (such as hedge funds) previously have had [10][2]. The market price itself is a reflection of all these moving parts and thus even if a robust pattern has been found, it is likely to be arbitraged away as more market actors step in to trade away any mis-pricing [7]. This is quite literally like an image recognition model having to recognise cats which have the ability to morph into dogs.

One issue that is particularly pertinent in the financial domain is the issue of data dimensionality and the underlying size of each individual data set where even if there were fifty years of daily financial data available, it amounts to less than 20,000 data points[3]. This means that one must think carefully before adding features to the model.

The use of exhaustive grid search algorithms will simply overfit the model and should not be used. This is due to the fact when we change and search for optimal hyperparameters, given the number of possible variations and only one small set of live price points, the probability of finding a strategy by luck remains extremely high. Arnott et al. provide an example of a trading

[2]Not to mention the crack down on information sharing between investment banks and hedge funds
[3]Assuming a 252 trading day calendar

strategy which would have passed even the most stringent evaluation methods in both training, validation and testing phases which actually turned out to have been created using a ranking of stocks based on alphabetical ordering [8].

Given the inherent challenges faced when developing systematic trading strategies, one may wonder if creating a strategy based on machine learning methods is a fruitless endeavor. Due to the unique issues related to applying machine learning in the financial domain, this project took a different view on how best to use machine learning by first creating a model to only predict trends using price data and then allowing a second model to try to capture the likelihood of the current regime to be conducive to strong auto correlation in the price series (i.e identifying times when the chance of seeing a trend was high). This diversifies the trading model as now the problem of alternative/changing drivers is reduced, as the model only trades based on price momentum which can be driven by any factor. While the second model will increase the allocation of risk capital to the trend model based on the models predicted assessment of the probability distribution of future trend regimes.

3 Literature Review

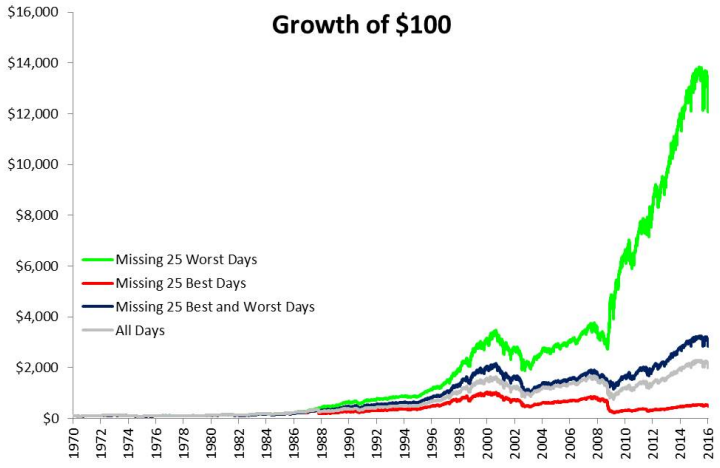
3.1 Heteroskedasticity in Financial Markets

Complex systems that continuously evolve make applying ML techniques in a way that results in a robust model very difficult [8]. Financial markets have been shown to exhibit heteroskedasticity in their returns [11] and as such, creating a robust systematic trading strategy relies on more than just a simple data input/signal output type of architecture. In trading systems, classification of regimes and subsequent buy or sell signals must also be compared vs the expected value of the that trade. A model with an 80% classification success rate will under perform if it mis-classifies the the 20% of samples which result in large moves in the asset.

The corollary of that is by only targeting classification of outliers you reduce your sample set and thereby reduce the statistical validity of the results. There is rarely a free lunch in financial markets and as such there will be no free lunch when using machine learning methods to predict financial times series.

For example, figure 1, shows that missing out either the best or worst 25 days of the Standard and Poors 500 companies equity index (S&P 500) over the last fifty years drastically changes the returns compared to always being invested [12]. Given that 25 trading days represents only 0.005% of the total trading days, it shows the importance of being correctly positioned for outlier events and also the fact that one model can be severely impacted by one off or black swan type events [13]. Hence analysing simple classification or accuracy statistics alone when assessing model performance is not enough when applying ML to finance. Trend based models tend to exhibit positive skew as they can pick up and ride the wave on these types of outlier events [14]. Previous research has centered on price based data possibly as it is easily available and also because it is one dataset that is likely to have higher frequency data available ([15], [16], [17]).

Figure 1: Growth of $100 Invested Since 1975



Source: Market Watch [12]

3.2 Machine Learning Model Description

3.2.1 Artificial Neural Networks

Artificial Neural Networks (ANN) were conceptually first written about in the early 1940s [18] when neurophysiologist Warren McCulloch and mathematician Walter Pitts wrote a paper on the theoretical functioning of neurons. Donal Webb then expanded on this in his paper "The Organization of Behavior" while the first computerised neural networks were developed by IBM in models called "ADALINE" and "MADALINE" [18]. At their core, ANNs are a series of large matrix multiplications where weights and biases, which represent the strength of connections between nodes, are optimized to provide the best approximation to the desired output.

The popularity of ANN methods have increased as enhanced computational power led to the lowering of barriers to entry in the field. One of the more popular networks used in practice is the mutlilayer perceptron, initially introduced by Frank Rosenblatt in 1958 [19]. A perceptron is the simplest form of a layered network and consists of a single neuron with an adjustable weight. The perceptron can divide the hyperplane into a 2-dimensional space which is then used to classify the given inputs. The hyper plane is defined by the linearly separable function below.

$$\sum_{i=1}^{n} x_i w_i - \theta = 0 \tag{1}$$

Multi Layered Perceptrons (MLP) can be defined as

$$NeuralNet_j^l = \sum_{i=1}^{N_l} w_{ij}^{l-1} y_i^{l-1} \tag{2}$$

where $NeuralNet_j^l$ is the lth layer of the jth neuron which gives us the weighted sum of its inputs. The weights of the connections to the next layer $l = 1$ are used to calculate the weighted sum of inputs for the subsequent layer in the network.

$$Y_j^l = ActivationFunction(NeuralNet_j^l) \tag{3}$$

This weighted sum of inputs is then passed through to an activation function which maps the output of the hidden layer to a space defined by the activation function, in this step we can introduce non linearity into the model itself by using a non linear activation function such as the softmax function described below (which is used in this project for the recurrent neural network model (RNN)).

$$y = \frac{\exp(x_i)}{\sum_{j=0}^{k} \exp(x_j)} \quad for\ i = 0, 1, 2, ..k \tag{4}$$

This maps any series of values onto a plane of [0,1] and the output depends on the values produced by the node which feeds the activation function.

While there are a vast array of model training techniques , one method to train an ANN is to use backpropagation, where updates to the network's weights are fed backwards through each layer. Each model has an error function which is used to grade performance, which could be the the squared difference between the models output and the desired output (mean squared error, used in training the RNN in this project).

The universal approximation theorem [20] of the MLP model in neural networks states that ANNs can approximate any function and thus in theory, should be able to identify any patterns within the data that another method would also identify.
The temptation to introduce multiple layers with thousands of connections in order to find a solution is ever present, however (and especially in the realm of finance) this increases the risk of over-fitting or selecting a model based on sheer luck, which fails when tested on live unseen data. This project uses a recurrent neural network architecture as standard neural networks have limitations. They rely on the assumption of independence among the training and test examples. After processing each data point, the entire state of the network is lost. If the data points are generated independently (such as an individuals medical diagnosis), this presents no problem. But if data points are related in time or space, this approach is flawed.

In financial assets, the market reacts and learns from the history of the price series, thus data points are indeed related. Recurrent neural networks look promising as they learn the objective function by analysing sequences of points (hence they have been used extensively in text sentiment analysis where context of words within the sentence is important).

This memory element can be very useful in financial timeseries as past behaviour tends to dictate the future prices path dependancy. The specific model architecture this paper uses is the Long Short Term Memory (LSTM) model [21]. LSTM has traditionally been used in the prediction of sequences of text in natural language processing. The use of LSTM models in finance is gaining popularity with [22], [23] and [24] all using LSTM for time series prediction. LSTM was introduced by Hochreiter et al. in 1997 [21] and makes use of various "gates" to control the flow of information across various nodes.

$$F^i{}_t = \sigma(W_F u^i{}_t + V_F h^i{}_{t-1} + bias_f) \tag{5}$$

$$I^i{}_t = \sigma(W_I u^I{}_t + V_I h^I{}_{t-1} + bias_i) \tag{6}$$

$$O^i{}_t = \sigma(W_O u^i{}_t + V_O h^i{}_{t-1} + bias_o) \tag{7}$$

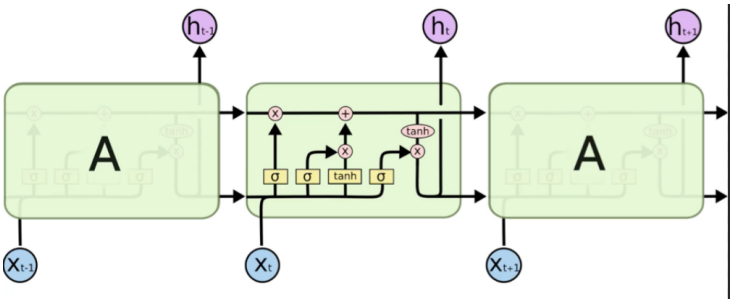
$$C^i{}_t = F^i{}_t * C^i{}_{t-1} + I^i{}_t * tanh(W_C u^i{}_t + V_C h^i{}_{t-1} + bias_C) \tag{8}$$

$$H^i{}_t = O^i{}_t * tanh(C^i{}_t) \tag{9}$$

$$Z^i{}_t = g(W_z H^i{}_t + bias_z) \tag{10}$$

where $*$ denotes element wise multiplication, σ is the sigmoid activation function, W and V are weights matrices of different layers, and $F^i{}_t$, $I^i{}_t$, $O^i{}_t$ are the forget , input, output gates and $C^i{}_t$, $H^i{}_t$, $Z^i{}_t$ are the cell, hidden and output states of the LSTM. The LSTM uses the cell states to summarise past information and controls the flow of historical memory using the forget gate. The pass through of new information is set by the input gate. Thus the LSTM model should be better equipped than an ANN in learning the long term relationships relevant to assisting in the prediction of the target variable (in this case the future price move). Figure 2 depicts the structure of a Long Short Term Memory Model.

Figure 2: Long Short Term Memory Model



Source: colah.github.io, Google Images

3.2.1.1 Background Research

ANNs have been used extensively in academic financial journals, Czekalski et al. made use of feedforward multilayer perceptrons to predict the direction of the EURUSD currency rate [25], while Wang et al. applied probabilistic neural networks and Least Square Support Vector Machine models to assist in the prediction of Chinese equities [17] . Literature on using ANNs to forecast currencies has a twenty year history as even at turn of the millennium Joarder et al. tested three separate ANN based forecasting models using standard backpropagation, scaled conjugate gradient and backpropagation with Baysian regularisation to predict movements in various Australian dollar currency crosses [26]. Gunduz et al. tested a variety of ANN models and Support Vector Machines (SVM) to try to predict the price of equities traded on the Istanbul Stock Exchange, they found model classification accuracy rates consistently around 70% on test data [27] .

Most of the research cited makes use of various technical indicators which are derived directly from the price series, for example, Gunduz et al. make use of ten price based technical indicators to uncover patterns within Turkish equities [27]. Prior research suggested above 70% classification rates were attainable on average, however in contrast to this, there is evidence that ANNs often show inconsistent performance on noisy data ([28], [29], [30]), which highlights the challenges in applying complex models as the risk of overfitting the data remains an ever present threat.

3.2.2 Support Vector Machines

One model which can be used to find patterns in financial data are Support Vector Machines (SVM) and Support Vector Regressions (SVR). SVMs were initially introduced by Cortes and Vapnik [31] which led to the development of non-linear models which provide binary classifications of target vectors while SVRs output real-valued predictions. In similar fashion to an ANN, the SVM tries to classify an n-dimensional feature vector according to where that data lies on an n-dimensional space.

The model makes use of the kernel trick [32] to transform the original data into a higher dimensional space such that we can then uncover a separable hyperplane. The function updates the parameters of the kernel function such that it can more accurately separate and classify the data points. The error correction model in the case of a SVM or SVC (Support Vector Classifier) is a maximisation problem where the model tries to maximise the distance between all the separating hyper planes (maximal margin classifier) it has constructed. The solution to

the optimisation problem can take the form of the following

$$Maximise_{\beta_0,\beta_1,...\epsilon_1...M} M_{margin} \tag{11}$$

Subject to

$$\sum_{j=1}^{p} \beta_j^2 = 1 \tag{12}$$

$$y_i(\beta_0 + \beta_1 x_1 + \beta_2 x_2 + ... + \beta_j x_{ij}) \geq M(1 - \epsilon_i) \tag{13}$$

such that $\epsilon_i \geq 0$ and $\sum_{i=1}^{N} \epsilon_i \leq C$. Where M is the width of the margin to be maximised, β_i denotes the weight of the coefficients and x_i is the i^{th} observation ($i = 1, 2, ..., N$) of the dataset, ϵ_i is a variable whose value sets the degree to which individual observations are allowed on the wrong side of the margin or hyperplane. C is a non negative tuning parameter which sets out the total error allowed for observations to be on the wrong side of the margin [33].

3.2.2.1 Background Research

Kara et al. utilised SVMs to discover patterns in Turkish equities achieving classification success rates of around 60% when tested on hold out data [34]. Both polynomial and radial basis SVMs were trained on a host of price based technical data with the radial SVM outperforming the polynomial SVM. The polynomial SVM also had a much higher variability in classification success rates and thus the radial basis may be best suited to predicting financial time series. The results of the radial SVM were significant with a test statistic of 3 and p-value of 1%. The radial basis kernel is used in this project.

Similarly, Patel el al. tested a selection of four different machine learning methods (ANN, SVM, Random Forest and Naive Bayes) using ten technical indicators [35]. The predictor space was also transformed into trend deterministic states, which map the original continuous values into discrete values based on the trend of the continuous valued indicator (downward trend = -1, upward trend = 1). Results suggested that model performance is improved when trend deterministic data is used instead of continuous data. They postulate that the models perform better on the trend deterministic data as this transformation inputs the underlying trend of the indicator which then allows the model to learn the relationship between the underlying predictor trends and the actual output trend.

SVM classification rates using this approach (for both radial and polynomial kernels) approached nearly 90% on average. Huang et al. also tested the performance of an SVM model (radial kernel) to predict Nikkei futures prices[4] and achieved classification rates of 73% on test data. The authors also found that combining model forecasts increased out-of-sample test results [15]. Overall, prior research has found interesting and impressive results for using SVMs as a method to predict future time series such as equities ([36], [16], [28]) and currency movements [37].

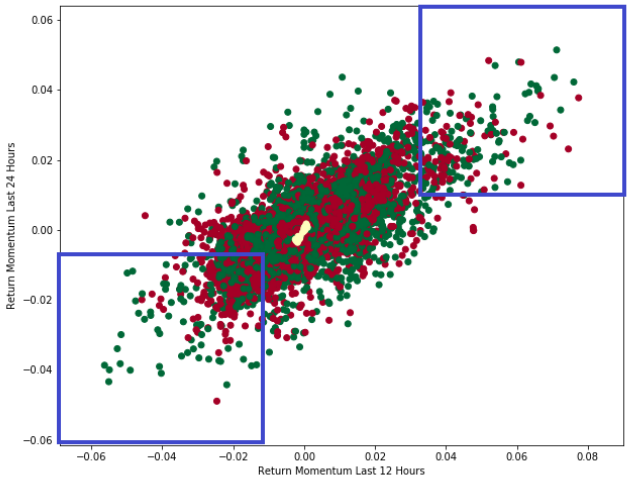
3.2.3 Ensemble Methods: Random Forests

Ensemble methods are a technique in machine learning which uses the outputs of multiple models to form a consensus around the prediction, while there are many different techniques one can

[4]Nikkei 225 is one of the main Japanese Stock Market Indices

use, this paper proposes to explore whether ensemble methods can be used to reinforce the accuracy of the model on unseen data. One approach is to use tree based methods [38], which involve segmenting the predictor space into a number of regions. Figure 3 looks at the prior 12 hour (x-axis) and 24 hour (y-axis) return of EURUSD with the subsequent 24 hour ahead move in the currency colour coded from red (negative return) to green (positive return). Even in such

Figure 3: Tree Based Methods Segment the Predictor Space into Numerous Regions



Source: EURUSD FX Rate, Kaggle Database

a simple split of the predictor space, we see some evidence that more subsequent positive days occur in the bottom left of the chart (when prior 12 and 24 hour moves in the currency have been negative). Tree based methods use this type of inference to make predictions on future data points. The goal of the decision tree is to find multiple regions which then minimise the squared error of predictions

$$\sum_{j=1}^{J}\sum_{i=1}^{R_j}(y_i - y^*_{R_j})^2 \tag{14}$$

where $y^*_{R_j}$ is the mean response for the training observation within the j^{th} region. The methods tested in the project will revolve around Random Forests (introduced by Leo Breiman [39]) which averages the prediction across hundreds of single decision trees. The trend estimation model uses a random forest architecture to identify trend conducive regimes,

3.2.3.1 Background Research

Research has shown that Random Forests can generalise better than single decision trees as they maintain high variance while reducing the bias of the model [40]. Random forests use bootstrapping to take multiple random samples from the training data and a random subset of feature vectors to create a diverse array of decision trees which aid model generalisation. This method helps to create decision trees which are uncorrelated to each other and thus have a

lower probability of overfitting the data. Patel et al. showed that the it was possible to achieve classification rates of over 80% for various stock price predictions when using the random forest algorithm and input features which consisted of technical price based indicators [35]. The research used a random selection of three feature vectors (out of a total of ten) to create each single decision tree in the random forest model and tested the results from 20 to 200 trees using a majority vote for the final classification.
Kumar et al. performed extensive tests to compare the performance of support vector machines to random forests and found that on average support vector machines outperformed random forests when predicting the direction of the Indian stock exchange [29]. Three predictors were considered for each node while the number of trees for each model ranged from 200 to 1500. There was no significant reduction in the error rate for the out-of-sample data based on the number of trees. The RF model achieved classification rates of over 67% on out of sample data. Chatzis et al. found that RF methods outperformed SVMs and logistic regression techniques when predicting stock market crisis events, they also note that RF models are relatively robust to overfitting due to each forest only being exposed to a subset of the available feature vectors [41].

4 Methodology

This section provides a deep dive into the underlying methodology of the full life cycle of the model, including how the data pipeline was structured, the training and testing methodology and various model evaluation techniques. The model architecture selected was to combine two different models, the first model with the explicit goal of capturing the trend in the time series while the second model would act as a regime identifier and allocate capital to the trend model when the current regime was conducive to the positive performance of trend. The model design is covered in greater detail in section 5.

4.1 System Specifications

The Python programming language (Python 3.6, [42]) will be used as the main programming application to achieve the objectives, with the python packages of Numpy [43], Scipy [44], Sci-Kit Learn [45], Matplotlib [46] and Pandas [47] all providing very useful data manipulation functions to help store, clean and process data. For more complex ANN learning, the machine learning package Keras [48] will be used in the training and testing of Neural network models. In order to train the more intensive models such as the Long Short Term Memory model, a cloud server was used, the specifications of the machine were as follows, 1 CPU with 1.7GB RAM and 250GB SSD. While not computationally very powerful, the machine was selected due to its low running costs which allowed the RNN to train on the dataset overnight. Total time spent running the LSTM model across various training/test periods was over 250 hours.

4.2 Data Description

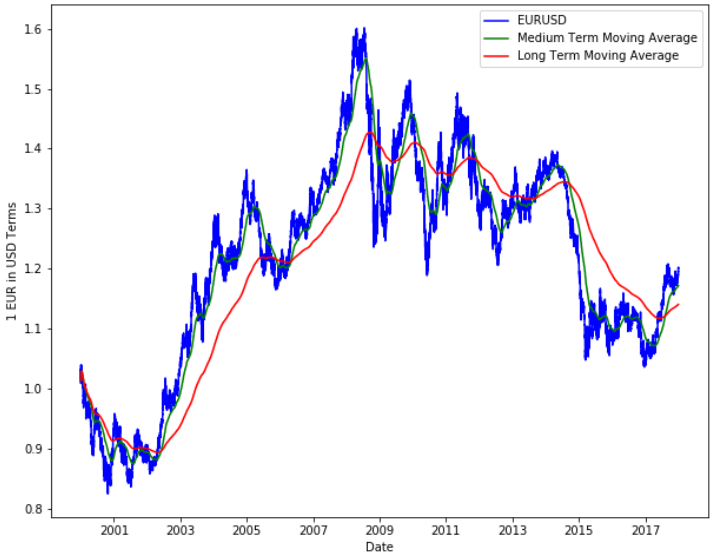
One of the most important aspects of a trading model is to ensure that the underlying data is of high quality and doesn't contain erroneous values. The trading model uses hourly spot exchange rates for G10 currencies [5] alongside various economic data and currency valuation data from the CitiFX and the Organisation for Economic Cooperation and Development (OECD) and national statistics offices.

The data used for this project consisted of hourly foreign exchange data for the EURUSD currency pair[6] and related economic and currency valuation data. Figure 4 shows the performance of EURUSD over the last sixteen years formed of hourly snapshots and its 55 day and 55 week moving averages. In total the hourly data was sourced from 2001 to the start of 2019, which consisted of just over 120,000 rows of data. One issue encountered was that training the LSTM (even using powerful GPU machines on the cloud server) took a prohibitively long amount of time. This made cycling through various model iterations challenging, with the process often left to run overnight. There is evidence of the existence of momentum in currencies [49] and this project will explore ways of capturing the trending nature of currencies more accurately than a simple linear model.

[5] Hourly FX Spot Data Provided by Citi Foreign Exchange

[6] Value of 1 European EUR in terms of the United States Dollar

Figure 4: EURUSD Rate Since 2001



Source: EURUSD FX Rate, Kaggle Database

4.3 Data Pipeline

4.3.1 Objective One: Data Cleaning and Standardisation

The data was initially screened for errors and missing data points. Cleaning of the data was broken down into outlier detection (to capture any erroneous inputs) and cross checking the data vs known events in time (such as Brexit, election dates) to see if the observed outliers match the expected time frames. This data was also screened and mapped to correct date scales so that the model only used data available at the time of trade signal generation. Once the data had been cleaned appropriately, the next step was to standardise the data and then choose the model features. The data was stored and manipulated using Pandas dataframes while Sci-Kit Learn provided the functionality to run both the random forest and support vector regressions. While the package also has some very useful functions such as StandardScaler to standardise the data, in this project a bespoke scaling method was used where the data is standardised by the standard deviation of the feature over a rolling three year lookback window. This is a more realistic approach to use when creating systematic trading strategies as this method uses all available data up to the point of trading and expresses all point in relation to its most recent history, allowing the denominator (the standard deviation) to vary over time as the market changes regime.

4.3.1.1 Standardisation

Standarising the data remains an important initial step that has been used across a wide variety of prior research. Subtracting the mean and dividing by the standard deviation (z-scoring) [22],

min-max standardisation[7] ([27], [29]) and other techniques such as mapping functions to scale feature vectors onto a pre-defined scale ([34], [17], [35]) all help to achieve the overall goal of scaling, which is to ensure that larger or more volatile features do not overwhelm features of smaller magnitude and lower volatility. Below shows a code snippet of the function to standardise the dataset.

```
def standardise_data(dataset, full_cols, standardised_cols,window):
    '''
    This function computes the standardised returns on a rolling basis looking
        backwards over the window specified.
    This is the most realistic in terms of a trading strategy and also means the test
        data is standardised on the correct basis using the
    latest data available at each timestep.
    :param dataframe:
    :param cols:
    :return: standardised dataframe of values
    '''
    train_standardised =
        dataset[standardised_cols].subtract(dataset[standardised_cols].rolling(window).mean())
    train_standardised =
        train_standardised.divide(dataset[standardised_cols].rolling(window).std())
    # we will only return the data which is outide the initial window standardisation
        period
    # add non standardised features
    for feature in full_cols:
        if feature not in standardised_cols:
            train_standardised[feature] = dataset[feature]
    # This function now returns the neccessary file with both standard and non
        standardised columns.
    return train_standardised.loc[window:,:]
```

4.3.2 Objective Two: Feature Extraction

The model features were chosen with the goal of capturing the underlying trend of the time series, it is very important in finance that the features are not chosen using a grid search of the hyperparameters. Feature selection driven by mining the training set greatly increases the risk of overfitting and should be avoided when working with financial times series [8]. Thus the features which had an intuitive justification for defining the trend are shown in the following table. Financial time series prediction is a very challenging task given the inherent noise in the data, thus the number of available features were kept to a minimum and in effect used only the deviation of the currency price from the short, medium and long term price averages. We use these features as they allow the model to capture the underlying trend (in either an upward or downward direction) by looking at the current price level in relation to historical levels.

The following formula was used to calculate the deviation of the current price to its moving average over a set horizon. This project used an exponentially weighted moving average.

[7]Map observations onto a scale of [0,1] with 0 being the minimum value of the observation and 1 denoting maximum

Table 1: Model One: Trend Capture

Indicator	Description	Type
Short Term Momentum	5,10, 21, 55 Period Moving Average	Technical
Medium Term Momentum	100, 155, 200, 255 Period Moving Average	Technical
Long Term Momentum	300, 350, 400, 455 Period Moving Average	Technical

$$EMA_t = price_t * k + EMA_{t-1} * (1 - k) \quad for\ i = 1, 2, 3...t \tag{15}$$

where $k = \frac{2}{(N+1)}$, and N is the length of the moving average and $price_t$ is the price of the underlying asset at time, t.

Dimensionality reduction (via principle component analysis) was also performed on the features, however this did not have any positive impact on the models performance.

4.3.2.1 Feature Selection For Trend Regime Prediction

The feature selection process for model two (estimating the likelihood of a trending price series) was based on economic intuition as this can lessen the probability of over-fitting [8]. As patterns and correlations in prices can change quite quickly, it is important not to overfit the model on the training data. The technical price based indicators are captured in the first model, while the second model measures non-price based information such as economic momentum and risk appetite (which is a measure of markets current sentiment). Table 1 outlines the features which will be used in model two.

Table 2: Model Two: Trend Strength Estimation

Indicator	Description	Type
Relative Economic Momentum	1 Year Economic Momentum	Economic
Global Economic Momentum	1 Year Global Economic Momentum	Economic
Currency Volatility	Annualised Volatility	Technical Stress Factor

The approach of combing two separate models each with specific goals is a novel way to use machine learning in finance, in essence one model controls the price prediction component while the second model is used to gauge how likely the first model is to generate positive returns.

4.3.3 Justification of Features

The aim of the features used in this project was to try to juxtapose higher frequency technical price data alongside lower frequency economic data to assist in a model's prediction accuracy. While price based metrics offer the easiest route to higher frequency data, they also lack the ability to capture the fundamental economic factors that can drive currency direction [50]. Thus combining two separate models to handle each aspect of the trade signal process can create synergies between each model. Below we showcase factors that can impact currency direction.

4.3.3.1 Economic Fundamentals

One of the most important factors that sets the broad direction of a currency pair is the relative economic performance between two countries. Dahlquist et al. show that measuring relative economic momentum between different countries has the ability to forecast currency direction over medium term horizons [51]. That paper showed statistically significant results that economic fundamentals have an important role in relative capital flows and subsequent currency direction. This economic feature is used in model 2 to capture relative economic dispersion between the Euro Area and the United States, with the view that diverging economies should create an environment where currencies trend in the direction of the economic outperformance.

4.3.3.2 Price Momentum

Price momentum here refers to the historical trend in the currency, research does suggest there exists exploitable momentum within the currency space [52]. The reasoning for the existence of momentum in currencies is likely due to decision delays amongst different investor segments who build up positions in currencies over time. Auto-correlation in currency returns may also exist as information is priced slowly over time (such as economic data) and thus as more economic data is published, this adds further confirmation to a trading signal and thus conviction and trade size increases. Model one is tasked with capturing price momentum.

4.3.3.3 Risk Appetite

Risk appetite in financial trading refers to the wider investor communities appetite for higher risk assets. For example when risk appetite is high, then investors buy equities and currencies with high interest rates as they are happy to take the higher probability of losses (.i.e. the investment risk) associated with the higher return. Aggregated global economic data help to forecast changes in risk appetite as investors react positively to strong global growth and negatively to global growth shocks. Low risk appetite (growing investor fear) tends to be accompanied by regime shifts, and also increases in price momentum as the asset searches for a new equilibrium price level. Model two uses this feature for regime prediction.

4.3.3.4 Currency Volatility

The currencies volatility is a measure of how erratic the currency is behaving and increases in currency volatility can point towards changing regimes as the market tries to find a new equilibrium price level. Model two uses currency volatility as one of the features to assist in identifying a trending regime.

While the above drivers of currency markets are not exhaustive, it can provide a frame work for constructing a systematic trading strategy. Currency drivers can shift over the short and medium term and while this can all be partially captured by price action, it is also prudent to try to understand the dynamics of other potential drivers as discussed in the previous section.
Therein lies the main problem when combining machine learning and finance, that other areas such as the biological or physical sciences do not encounter, is that at the core of each price, are human or human created trading algorithms which have collectively decided on a price level at which to transact.
Every price of a trade in financial markets consist of the dreams, fears and greed of human emotion. Not only this, but market investors are influenced by past patterns and incorporate new information (including newly published research) into future decisions, thus altering the

environment in which the historical pattern has been learned. Bartram at al. provide evidence that anomaly (excess) profits published in research papers tend to decay substantially in the period after publication, which suggests any new findings are arbitraged away[8] quickly as the new information is incorporated into the market [7].

While early academic research espoused the efficient market hypothesis, there are certain times where the market is not fully rational [53]. Thus opportunities do exist, but finding repeatable trading opportunities requires more than just analysing price patterns in data. Given the randomness of markets, even poor strategies can perform well for long periods of times due to sheer luck. This creates new challenges for practitioners of machine learning in finance, in fact, new research has also started to focus on protocols and best practices around data mining in finance [8].

Being able to identify trends and trend regimes would allow the trading model architecture to outperform other simpler linear models while not churning (over trading) the portfolio if currency starts to enter a pattern of mean reversion. Building the model to this asymmetric specification can assist in the generalisation of the model.
Thus it was decided to first train a model to identify trends in each series and then overlay that model with another capable of using the non-price data to identify periods when currencies exhibit trending behaviour. For example, periods of elevated economic dispersion could be conducive to trend strategies outperforming.

4.4 Objective Three: Training and Testing Methodology

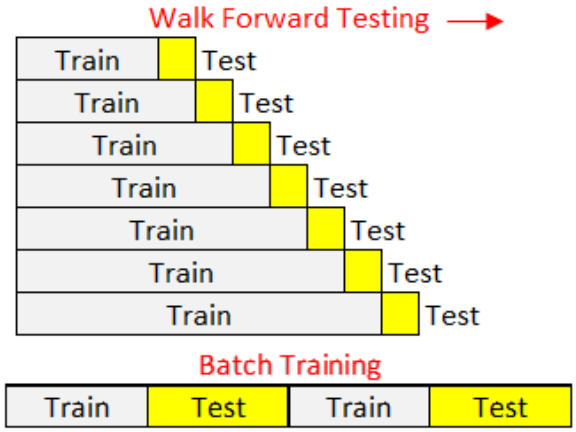
The model training stage allows the learning algorithm to understand and capture the relationship between the features and the classification target vector (the target vector will be the future direction of the asset price 24 hours in the future). Prior research has generally used binary classification of the future asset price move when developing models([54], [27], [41]).
Model training allows us to understand which combination of models (RNN, SVM, RF) and underlying architecture (hidden layers, number of decision trees, kernel type) are best suited to capturing the nuances in the data. This stage of the process also provided an opportunity to experiment with various models to see the differences in performance. The training and testing phases were performed over multiple different windows such as walk forward testing, K-fold testing 5 and simple train and test splits over the full dataset. The structure of the different training/testing methods are shown below. The results of training and testing on the price data will be further explored in section 6. The code below shows a snippet of a python function called create_train_test file written to split the data set into various train and test periods based on whether one wants k fold training or walk forward training.

```
    # training size is the first x data points and the test data is appended onto the
        train data
    # such that we use a walk forward testing framework
    if concat_results:
        # provide data up till the test data zone
        train_data = int(data_file.shape[0]) - (test_size + test_buffer)
        train_original = data_file.iloc[:train_data, :].reset_index(drop= True)
        # provide data in the last x points of test data available
        test_original = data_file.iloc[-test_size:, :].reset_index(drop= True)
    else:
```

[8] Of course a more sinister reason could be model overfitting on historical data

```
# this effectively acts as a batch training/split
    train_original = data_file.iloc[:int(data_size), :].reset_index(drop= True)
    test_original = data_file.iloc[int(data_size) + test_buffer: (int(data_size) +
        int(test_size)), :].reset_index(drop= True)
```

Figure 5: Walk Forward and K-Fold Train/Test Split



4.5 Objective Four/Five: Model Evaluation Methods

In order to validate the model, the standard procedures of analysing the mean-squared error, classification hit rates and precision and recall will be analysed. However, given this is a trading system, the selected model and its structure will need to be tested in a similar setting to real world trading. This requires the construction of a backtesting engine which can simulate each models performance taking into the signal lag [9] and the currency performance per each trading signal. This will provide a much more accurate analysis of the selected trading strategy performance[10]. While classification success rates will be important in gauging model accuracy, high classification rates may not lead to positive strategy performance if large currency moves are mis-classified. A code snippet taken from the trade backtesting engine is shown below .

[9] The time between signal generation and trading time

[10] The vast majority of research papers referenced in this thesis did not perform this type of testing

```
def backtester(results, test, trade_horizon):
    '''
    Calculate the returns of the trading strategy and store the results in the test
        file.
    :param results: the trading signals generated by the model.
    :param test: this contains the test data
    :param trade_horizon: this is the length of time of the trading signal prediction
    :return: A dataframe which has the test data and also the results of the trading
        strategy in pct terms of log returns.
    '''
    # This needs to change to handle the change in the target
    predictions = pd.DataFrame({"Date": test['Date'], "Predictions": results})
    test_results = pd.merge(test, predictions, how="left", on="Date").fillna(0)
    # calculate the returns of the signal for the signal which has been put through
        the error function to smooth the position.
    test_results["erf_signal"] = test_results['Predictions'].apply(erf)
    test_results["scaled_signal"] =
        test_results['Predictions'].shift(2).rolling(trade_horizon).sum() /
        trade_horizon
    test_results["scaled_signal_erf"] =
        test_results['erf_signal'].shift(2).rolling(trade_horizon).sum() /
        trade_horizon
```

4.5.1 Assessing Model Performance

4.5.1.1 Mean Squared Error The mean squared error (MSE) is used to measure the prediction accuracy of the model. It looks at the average distance of the predicted value from the actual value.

$$MeanSquaredError = \frac{1}{N}\sum_{i=1}^{n}(y_i - F(x_i))^2 \tag{16}$$

where y_i is the actual value and $F(x_i)$ is the predicted value for the i_{th} observation. Normally it is best practice to choose the model which has the lowest test MSE.

4.5.1.2 Precision and Recall Another method of model validation is to look at the precision and recall of the model [35], whose values can be calculated from a confusion matrix, which shows the number of correct and incorrect classifications per class. Table 3 shows the confusion matrix for a two class classification problem.

Table 3: Confusion Matrix

Actual (row) Predicted (col)	Buy	Sell
Buy	True Positive	False Negative
Sell	False Positive	True Negative

The confusion matrix allows us to calculate the precision and recall of the model.

$$Precision = \frac{TruePositive}{TruePositive + FalsePositive} \tag{17}$$

$$Recall = \frac{TruePositive}{TruePositive + FalseNegatives} \tag{18}$$

The precision captures the correct positive signals (e.g. in this case the correct number of buy signals as a percentage of all buy signals) that were actually triggered. Recall measures model accuracy as a function of the total number of positives in the data set itself. These statisitcs are important to analyse when assessing trading models as certain periods can be heavily biased towards one class, for example in the case of a financial asset which has a strong trend will by design have a higher occurrence of one class vs another.

4.5.1.3 Deflated Sharpe Ratio Given the inherent issues of overfitting in finance ([8], [4]), it is also important to analyse the overall Sharpe Ratio [55] which measures the return of a trading model per unit of risk (or volatility of returns).
High Sharpe ratios can be obtained by a genuinely good prediction model or through pure overfitting/selection bias, hence it is useful to deflate the Sharpe Ratio of every strategy tested to take into account the number of model tests as well as the skew of each strategy's distribution [56]. The Deflated Sharpe Ratio is a test statistic which can outline the probability of having found a true positive. It is defined below as

$$\widehat{DSR} = \left[\frac{\widehat{SR} - \widehat{SR_0}\sqrt{T-1}}{\sqrt{1 - \gamma_3 \widehat{SR} + \frac{\gamma_4 - 1}{4} * \widehat{SR}^2}} \right] \tag{19}$$

where $\widehat{SR}_0 = \sqrt{V\left[\widehat{SR}_n\right]\left((1-\gamma)\, Z^{-1}\left[1 - \frac{1}{N}\right] + \gamma Z - 1\left[1 - \frac{1}{N}\exp^{(}-1)\right]\right)}$ and $\widehat{SR}$ is the estimated Sharpe Ratio to be tested, T is the sample length, γ_3 is the skewness of the returns distribution and γ_4 is the kurtosis of the selected strategy, $V\left[\widehat{SR}_n\right]$ is the Sharpe Ratio variance across the number of strategies tested, N is the number of independent trials of strategies run (.i.e. the number of models tested) and Z is the cumulative function of the Gaussian distribution[11].

[11] Further information and theoretical proofs of the deflated sharpe are available from [56]

5 Project Design and Implementation

This section outlines the specific steps taken to create each model and provides some justification for the selection of various model parameters and specifications used. Various Python packages such as Sci-Kit Learn and Keras were used to create and test the classification success rates of each type of model. Fischer et al. made use of Keras (on top of Google's Tensorflow) to implement Recurrent Neural Networks (RNN) [22] while using Sci-Kit Learn for logistic regression, SVM and Random Forest methods.
One issue encountered was that the length of time needed to train the LSTM meant that training multiple variations across parameters was challenging from a time perspective.

5.1 Model Architecture

Three model architectures were used in this project.

- Recurrent Neural Networks: Long Short Term Memory Models
- Decision Trees: Random Forests.
- Support Vector Machines: both classification and regression methods.

One novel approach in selecting the target vector (.i.e. the future return of the asset) was to used a risk adjusted return and predict the risk adjusted return as opposed to the binary return [23], this ensures that the trend model can also estimate the positioning size and negates the need for a separate trade risk adjustment to be made.

During the course of the project, it became clear that using historical price data to train the model produced erratic performance in the testing period across all three models and various training and testing phases. Perhaps this was due to the nature of the underlying security being EURUSD (one of the most highly traded instruments in finance and thus was highly efficient) or that the model tended to over fit the training data (even when using techniques such as drop out rates and tree pruning was performed).

5.1.1 Trading Model Methodology

The failure of the model to learn robust profitable trading patterns using its own price data was one risks cited in the proposal which came to fruition. The solution to this problem was to switch the model from one which trained on the actual price series to one which trained on randomly generated price data which had a built in a trend component. In effect it was necessary to go back to basics and understand what the goal of the model and its features ultimately was, trend prediction.

If the series we are trying to predict has only exhibited trend in patches across the full time frame of available data, then the model is likely unable to pick this trend up over the full sample, as it ends up training on conflicting patterns. Figure 6 tries to illustrate this point, showing a sucessful training and test period for EURUSD, and also an unsuccessful period , where the training occurs on a period without any trend, while the test period is then a regime of strong trend.This type of issue emerged when training using K fold cross validation. In some instances

the training data set happened to be during regimes of strong upward or downward trends, and subsequently the model would perform well in the test phase[12].

Figure 6: Do you Feel Lucky? Model Performance Impacted by the Selection of the Training Period

Thus for the model to be able to capture trends, it needed to be trained to *recognise*a trend, therefore, a decision was made to switch the training data away from the historical price series data and onto randomly generated trending data series which could replicate trends in asset price series. Of course when training on randomly generated data one must also include some noise term such that we create series which mimics financial asset prices. Hence the trending data was injected with a noise term. Training on randomly generated data has two very appealing traits, one it reduces the chances of overfitting the data, however this is reduced but not removed as there is a chance the randomly generated data series happens by chance to mimic closely the testing phase of the data (hence training of various iterations of the randomly generated data is important). Secondly, it opens up the full live price series data to exist as the test dataset. By training the model to recognise trends, we can now test on the full 120,000 rows of EURUSD price data.

The randomly generated price series was created using the following formula.

$$X_t = X_{t-1} * \alpha + \epsilon_{t-1} \tag{20}$$

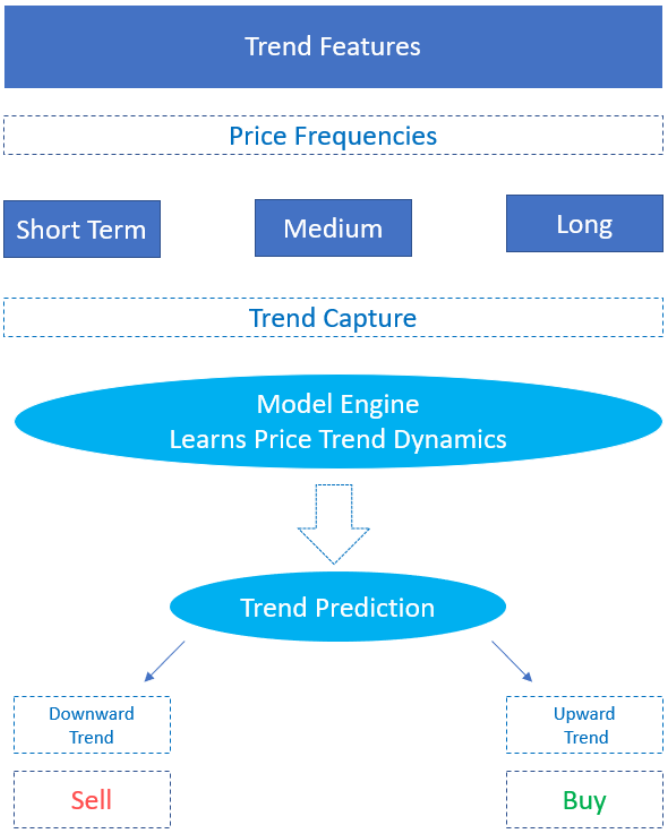
where X_t is the simulated price at time t , α is a parameter to control the strength of the trend and ϵ is a randomly distributed i.i.d error term used to simulate the noise of financial time series.

The trend capture model (model one) is described in the below diagram 7. The figure outlines how the underlying features feed into the trend model and also how each selection of features are tailored to capture short, medium and long term trend frequencies. Model one handles the signal generation process, by measuring the underlying relationship between the features and the target return variable. As mentioned in 1, the features were created from price based technical

[12]The results of various model performance is covered in section 6

signals, such as deviations from exponential moving averages across both short, medium and long frequencies. This purely price based factor model is used to capture trend only, hence we feed in only factors which are applicable to signaling a trend in either direction.

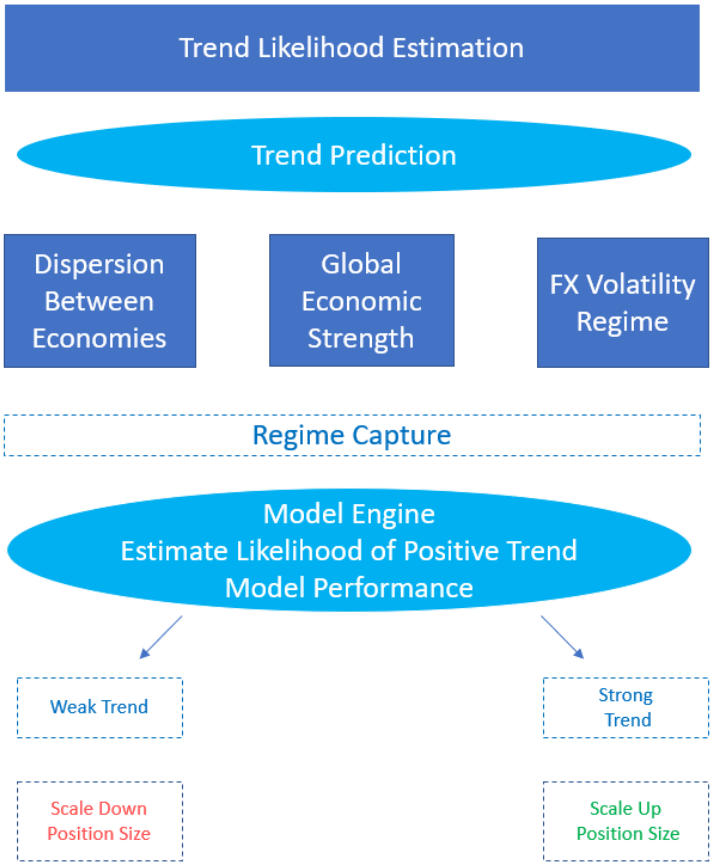
Figure 7: Model Architecture : Trend Capture



The overall goal of model one is to use machine learning models to identify the pattern and non-linearities inherent in trending prices which more simple linear based models fail to capture. The type of model also varied from using regression methods to estimate a continuous valued 24 hour period ahead and also a more simple classification on whether the price was predicted to move higher or lower. This one day ahead trend prediction was then used as the input into model two (trend estimation, 8).

The second model (figure 8), then tries to capture the prevailing market regime, and uses much slower moving fundamental data as features (as described in 2) in order to assist in position sizing of the trend model, by learning when historical fundamental features have tended to coincide with the positive performance of the trend strategy. This work on the intuitive reasoning behind why a currency exhibits trending behaviour, one process of thought is that economic dispersion between countries [57] can lead to large trends between currencies as the capital is pulled towards the currency of the economically stronger country. Decision delays by different segments of investors can also influence the slow grind of a higher price, while central bank monetary policy can also lead to large deviations between different countries currencies. Central bank policy can often be driven by economic data and thus looking at economic data we maybe able to identify regimes which are conducive to trending prices series. We also include a volatility feature as well as global and G10 economic performance, which can act as a potential harbinger of further increases in volatility when trend is driven by factors such as risk appetite (where investors flock back to safe haven assets). The target for this model is the performance of the trend model one historically over the testing period. Thus we combine two different models in order to incase the chances of out of sample model generalisation.

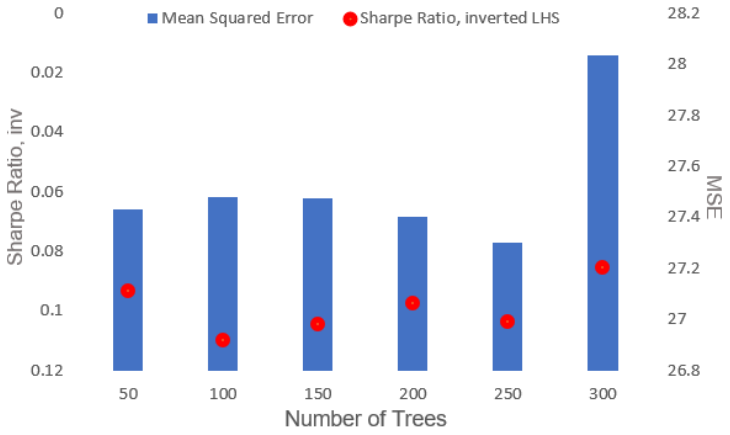
Figure 8: Model 2: Trend Estimation



5.2 Machine Leaning Model Parameter Specifications

As mentioned previously, three different machine learning models were used in the project. The testing and training datasets remained standardised across each model so that comparison of results would be as accurate as possible. The python package Sci-Kit was used to build both the random forest and SVC (SVR) models. Parameter variation was performed across a different number of decision trees and from that analysis, 200 trees where used in the final model phase as it was relatively quick to train and also provided good results (**??**). When looking at the results of batch training and testing, we can see that adding more trees to the random forest model did not necessarily add more prediction power. Figure **??** shows the mean squared error scores and the sharpe ratio [13] of each model trained across various number of trees and using a 50/50 train and test split. The maximum number of features to be selected at each node (i.e. the max features parameter) was kept low to reduce the correlation between trees and increase model generalisation.

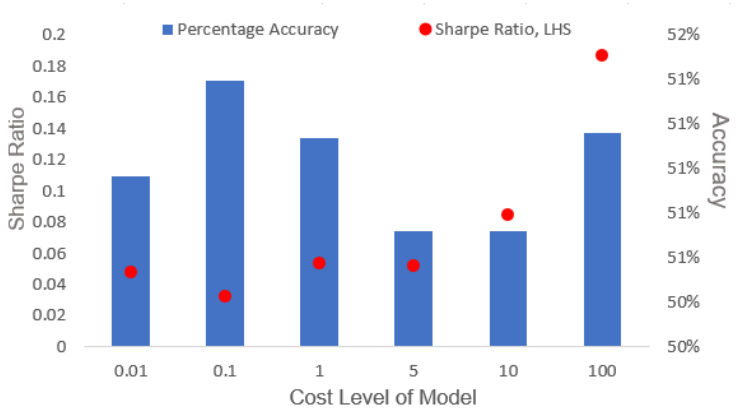
Figure 9: Random Forest Classifier Performance Across Various Number of Trees



Similarly for the Support Vector regression/classifer models used, a review of the appropriate cost to use was performed. The kernel used was the radial basis kernel to try to capture the non linear patterns in the price series. Interestingly, across various different cost levels, it was noted that higher costs levels tended to have slightly higher accuracy scores which also tended to translate into higher Sharpe ratios in the test period 10. The test period of the accuracy scores and Sharpe ratios shown here was from October 2014 until December 2017 (the period of data post 2017 was truncated at the start of the project to be used for the final blind model testing). Overall, the fact that the higher cost allowances generally performed better in the test period shows the value of not over fitting the training model when using financial time series. The recurrent neural network is one model which is very powerful in terms of the complexity and the nature of problems it can in theory solve, however the increased complexity also comes with a loss of generalisation given the propensity to over fit the data. While this in mind, the following model architecture was chosen to train and test the model, an initial first layer of 32 nodes followed by a layer of 16 nodes where the soft max activation function was used, the

[13] assuming a risk free rate of 0%

Figure 10: Support Vector Classifier Performance Across Various Cost Levels



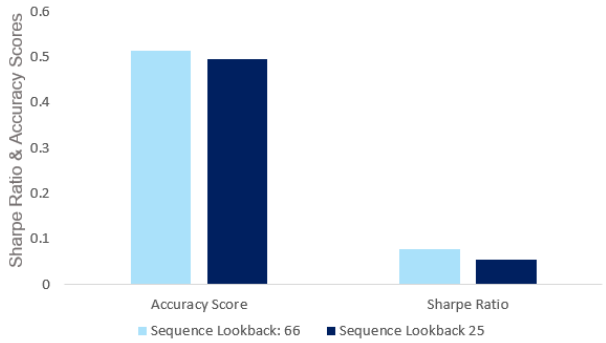
model adds a dense layer of eight nodes with a hyperbolic tangent activation function and a final output node which uses a linear activation function. The main parameter which was varied in order to assess model performance was the look back sequence and the number of epochs used to train the model. Generally as the project progressed , the higher the number of training epochs and look back sequences the better the model performance, increasing the epoch and look back sequence size noticeably increased the computational time spent training the model.

```
# Initialise the model the Recurrent Neural Network model
model = Sequential()
# add a layer of 32 nodes, where the sequences across each observation are returned
model.add(LSTM(first_layer,batch_input_shape = (None,look_back,no_features),
    return_sequences = True))
# second layer has half the nodes of the previous layer and uses a softmax
    activation function.
model.add(LSTM(second_layer, return_sequences = False, activation="softmax"))
model.add(Dense(8, activation = "tanh"))
model.add(Dense(1, activation = "linear"))
# finally the model is complied using the MSE and the adam optimser
model.compile(loss = "mean_absolute_error", optimizer="adam", metrics =
    ['accuracy'])
```

Figure 11 , shows the accuracy score and the sharpe ratio (in the testing period) for look back of 25 periods and 66 periods, the 66 period look back outperformed the shorter horizon, possibly as this longer sequence allowed the model to place current price action in the context of a longer window, similar to a human day traders using recent history to base his trading decision on when identifying a trend in the price.

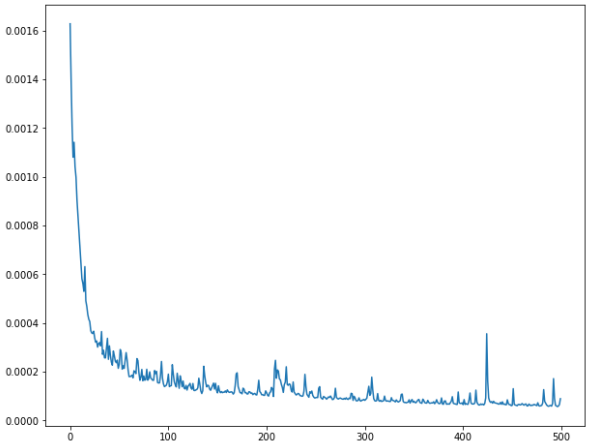
Figure 12, shows the accuracy of the LSTM model on the training data for a given number of epochs. We can see a steady downward slope which initially sees vast improvements in the accuracy scores before it finally settles at a stable if slightly jumpy equilibrium. The training data in this instance was the randomly generated data to mimic an asset price trend, so in this training cycle , the model is learning to understand the features and weights which define a

Figure 11: LSTM Model Performance Across Lookback Sequences



trend in the data. It is this learning process which can then start to recognise trends building in EURUSD in the testing phase.

Figure 12: LSTM Model Accuracy Over 500 Epochs



6 Results, Analysis and Discussion

The initial model training and testing on the EURUSD price series performed poorly, perhaps due to the model overfitting the problem or perhaps due to the different training periods not accurately representing the test data of the model. In figure 13, we show a scatter plot of the annualised return of trading EURUSD across various machine learning models, train and test periods and model parameters. An acceptable trading strategy needs to be robust and should show clustering of the points despite small tweaks to the model parameters (i.e. showing that varying the strategy parameters does not overly impact the prediction accuracy). In figure 13, we would expect to see a clustering of points near the top left of the chart (which would indicate an area of desirable return per unit of risk taken). However, we can see points scattered across the chart, suggesting there is a severe lack of robustness in the underlying dynamics of the trading model.

Figure 13: Annualised Returns (Y-axis) vs Volatility (X-axis) Across Various Strategies

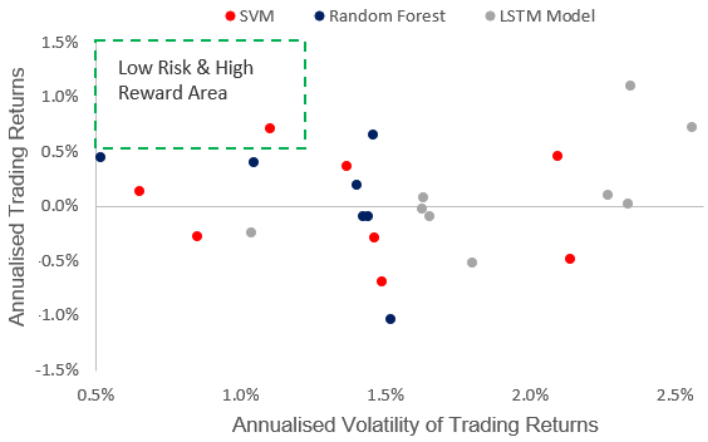
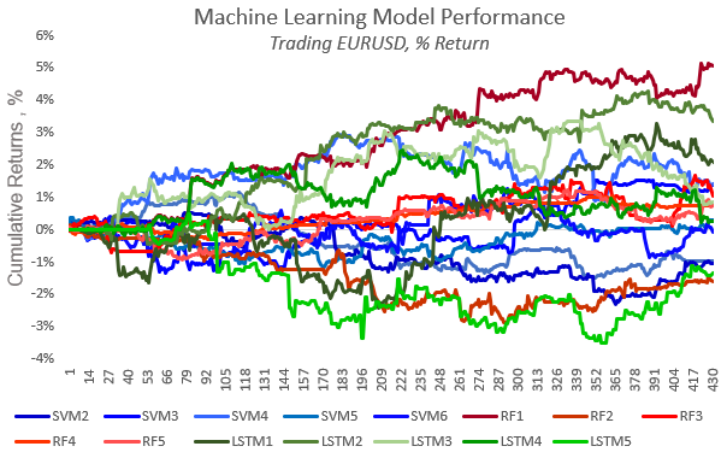


Figure 14 shows the cumulative return profile for a host of different machine learning trading models across support vector machines, recurrent neural networks and random forests. The returns shown in 14 display the model performance for various different training and testing periods. The returns are calculated from the backtester outlined in section 4.5. The returns shown on the y -axis are the cumulative log returns of each specific model when trading EURUSD, over randomly sampled 400 day periods. The performance is erratic and scattered across winning and losing strategies. It was due to this under performance and lack of robustness which led to the need to look at alternative model architectures when training the model.

6.0.0.1 Potential Reasons for the Underperformance

The erratic performance of the machine learning models when training on the price data may be due to the features used or due to the specification of each underlying model. However, given the justification for the features used in the model as outlined in 4.3.3, one should be wary of using tweaked technical features solely because they perform better, as that is blatant overfitting. One reason for the erratic performance of each model is that the training period may have occurred in periods where the currency was in a regime of mean reversion or weak trend.

Figure 14: Erratic Returns of Machine Learning Based Trading Models



This could inhibit the ability of the model to recognise trends. While if the model had trained on a trending price series, then it could learn the patterns associated with the auto correlation in the price, and trade this profitably in the testing period.
The poor performance of the various models on the live price data in capturing trend was disappointing, however, a decision was made to refrain from running an exhaustive grid search optimisation on the data as this would lead to overfitting of the problem. When one runs multiple simulations across different parameters (especially in finance) then it is inevitable that a profitable strategy is found by luck (similar to a coherent novel being written by a billion monkeys typing randomly on a typewriter). Given the inherent non-stationarity of financial timeseries, it is vital to ensure the model performs in a stable way across a variety of parameters. This rules out extensively applying complex transformations or optimisations on the features as the probability of having selected the "best" parameters by pure chance is high. In financial trading, model robustness is key.
As pointed out by Mclean et al. [58], financial model performance can differ vastly prior to publication than after it[14], whether due to over fitting or post publication anomaly decay [7], this means we must be extra vigilante when assessing model performance.

Due to this lack of stability in the performance, a different approach was needed. The new approach was to switch the training periods away from the actual price series and towards randomly generated trending data. Using randomly generated price data allows us to calibrate the strength of trend (and the significance of the noise) when training each model on how to recognise trending price series. The goal here is to allow the model to recognise the dynamics and patterns of a trending price series (as the goal of model one is to capture trend only), and thus removes the risk that the training period of the live price series lacks any sustainable trend. The number of data points used to train the models was 50,000 for the Support Vector Machines and Random Forests, while 4000 points were used for the LSTM (in the interest of reducing the time spent training the model). The test data set consisted of the full history of EURUSD data

[14]Research suggested a 26% fall in performance out-of-sample and 58% decline post publication

which equated to over 115,000 data points[15]. The 24 hour ahead percentage return of EURUSD was the target vector.

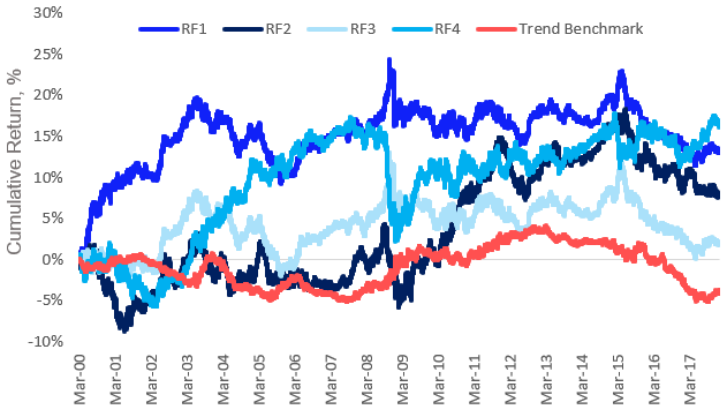
6.1 Performance of Model One: Trend Capture

This section outlines how various machine learning models performed when trying to predict EURUSD direction having been trained only on randomly generated trending price series. Each of the models here were trained as binary classifiers such that precision and recall could be calculated but also due to the fact that the returns tended to be more robust than when compared to models which predicted continuous variables. This is possibly due to the fact that the regression models tended to be skewed by large values that can occur in financial time series, which meant that when a classifier of the move (up or down) was used, this helped to dampen the volatility of the signal and removed the impact of outliers on the models ability to learn the patterns of the model.

6.1.1 Decision Trees: Random Forest

The first model tested after training on the randomly generated trend was the random forest machine learning model. Figure 15 shows the cumulative log return of trading EURUSD across various iterations of the models trained on different randomly generated datasets. The models below used 200 trees and a maximum of four features when selecting node splits. The testing period here is the full dataset which spans from the start of 2002 to the end of 2017. This testing period covers various different market regimes, such as the Great Financial Crisis, unorthodox central bank policies from 2010 onwards and the end of the commodity super cycle. The model features used were outlined previously in 1. In terms of results, the returns are still somewhat

Figure 15: Cumulative Returns: Random Forest Based Trading Model



scattered, however the performance on average is positive. The red line in the chart denotes the performance of a simple linear benchmark strategy of trading in the direction of the price trend over the last 24 hours, we use this 24 hour period as it matches the trade horizon that the model used as its target variable. Thus the benchmark linear strategy predicts the next 24

[15]Some data is lost due to the rolling standardisation lookback window

hours ahead currency move by looking at what occurred in the previous 24 hours. In addition to

Figure 16: Performance Statistics: Random Forest Based Trading Model

	Benchmark	RF1	RF2	RF3	RF4
Annual Return	-0.21%	0.74%	0.44%	0.11%	0.91%
Annual Volatility	1.75%	3.21%	3.83%	3.15%	3.77%
Sharpe	-0.12	0.23	0.12	0.03	0.24
Max	0.56%	0.74%	0.81%	0.73%	0.75%
Min	-0.74%	-0.80%	-0.94%	-0.80%	-0.84%
Accuracy (Hit Ratio)	39%	41%	44%	43%	44%
Precision	49%	51%	50%	50%	50%
Recall	50%	51%	58%	62%	60%
Mean Squared Error	41%	41%	43%	43%	43%
True Positive	22577	23460	28088	30004	29200
True Negatives	21327	22722	20541	18464	19390
False Positives	23172	22910	27789	30023	29027
False Negatives	22375	22938	20532	18257	19330
Sample Size (N)	111654	111654	111654	111654	111654

the cumulative returns of the strategy, we also shown a break down of each models performance statistics in figure 16, which are broken down to provide the annualised return and volatility, the Sharpe ratio (risk taken to reward received), the max and min return of each model, the mean squared error (MSE) and the precision and recall. We can see that on average the machine learning models do outperform the benchmark in terms of the MSE, accuracy and Sharpe ratios. The maximum daily return of each model was higher than the benchmark while each ML model also performed better in terms of annualised returns. Overall random forest model was able to identify trending patterns and profitably trade them better than the simple linear benchmark.

6.1.2 Support Vector Classifiers

The performance of the support vector classifier was mixed, with some models tending to underperform the simple linear benchmark. As a reminder, the SVC used the radial basis kernael and the train period was 50,000 rows of randomly generated trend data while EURUSD from 2002 until 2017 was the test period. One particular model (labelled SV2 in figure 17,) performed very well up until the final few years of the sample, perhaps as this coincided with a period of mean reverting behaviour in EURUSD. Overall , the SVM model performed somewhat disappointingly over the test full sample.

The performance statistics shown in 18 show the under performance of the models in general, as they show low or negative Sharpe ratios alongside underwhelming accuracy rates. The average mean squared error of the SVM models were inline with the linear benchmark.

Figure 17: Cumulative Returns: Support Vector Classifier Based Trading Model

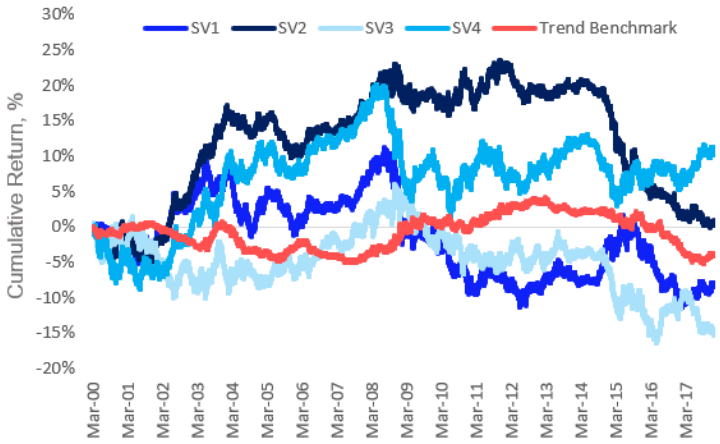


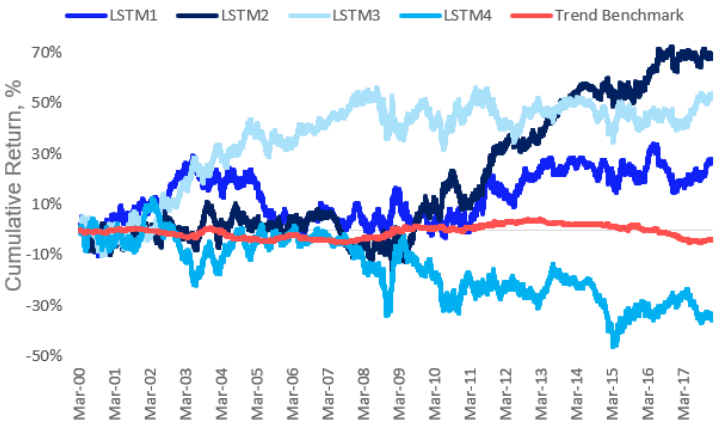
Figure 18: Performance Statistics: Support Vector Classifier Based Trading Model

	Benchmark	SVM1	SVM2	SVM3	SVM4
Annual Return	-0.21%	-0.45%	0.05%	-0.85%	0.63%
Annual Volatility	1.75%	4.12%	3.53%	4.15%	4.48%
Sharpe	-0.12	-0.11	0.01	-0.20	0.14
Max	0.56%	0.73%	0.64%	0.86%	0.81%
Min	-0.74%	-0.84%	-0.84%	-1.04%	-0.86%
Accuracy (Hit Ratio)	41%	59%	60%	20%	80%
Precision	49%	50%	50%	50%	50%
Recall	50%	68%	69%	35%	92%
Mean Squared Error	41%	43%	43%	29%	43%
True Positive	22577	33009	33725	11257	45036
True Negatives	21327	15528	14784	20850	3655
False Positives	23172	32901	33135	11139	44578
False Negatives	22375	15517	15265	21088	3686
Sample Size (N)	111654	111654	111654	111654	111654

6.1.3 Long Short Term Memory Trading Model

The recurrent neural network such as the LSTM provided the most impressive results along with the random forest module, with the majority of models outperforming the linear benchmark 20. All of the LSTM models here used an initial layer of 32 nodes , with an additional layer containing 16 nodes and used the softmax activation function. The length of the lookback sequence here was 90 periods while each model was compiled to use the mean square error as the loss function and the "adam" stochastic optimiser. The training period was run over 500 epoch across 4,000 data points.

Figure 19: Cumulative Returns: Long Short Term Memory Based Trading Model



The performance statistics 20 show that the average return across all models is positive and on a individual basis only model 4 underperformed the linear benchmark in terms of the Sharpe ratio, accuracy, mean squared error and maximum return. Accuracy, precision and recall scores all ranged from 40% to above 50% across all the models, while this is may not seem very good, the fact that the correct predictions outperform the incorrect predictions in terms of percentage return (otherwise the average returns would be negative), then the model does seem to be able to predict the larger moves in the currency. Based on the results, the LSTM models do seem to be best suited to recognising and predicting trends in financial timeseries. The outperformance of the LSTM model suggests that it is important for the model to understand the historical sequence of the data points such that the patterns of trend can sustain long enough for the model to recognise the sequence of points which form part of a trending price series. This result is also seen in [23] where the LSTM model outperformed all other machine learning models. Lim et. al used 100 epochs for training [23], while in this paper 500 epochs were used to train the model on the randomly generated data, with a sequence look back of 90 periods in the testing phase.

6.2 Performance of Model Two: Regime Estimation

This section looks at the performance of the trend estimation model, where we feed in the performance of each trend model (as outlined in 7) and make use of the fundamental features, such as economic growth differentials, historical volatility and the state of global growth to

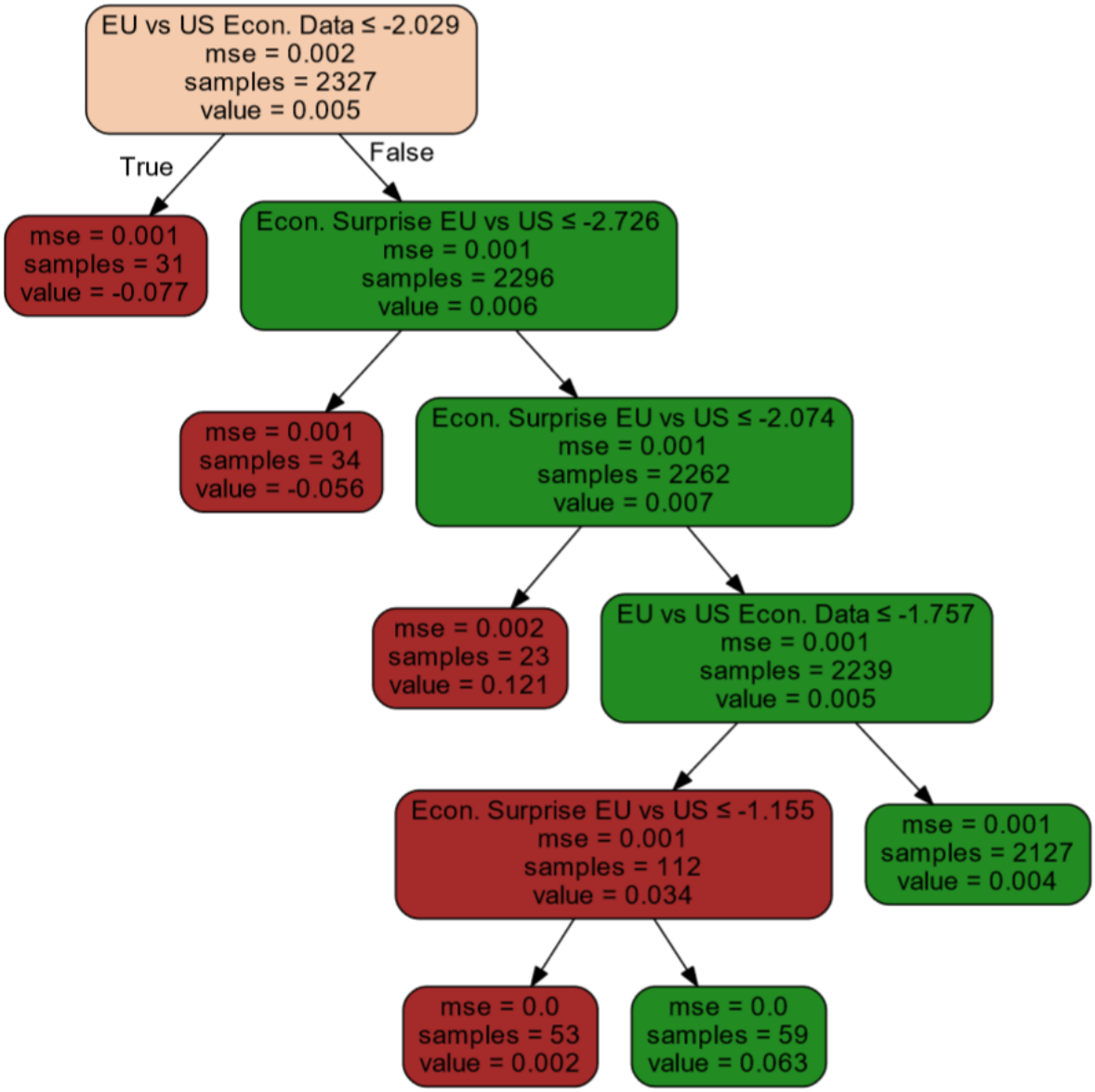
Figure 20: Performance Statistics: Long Short Term Memory Based Trading Model

	Benchmark	LSTM1	LSTM2	LSTM3	LSTM4
Annual Return	-0.21%	1.56%	3.78%	3.00%	-1.99%
Annual Volatility	1.75%	10.15%	10.15%	10.15%	10.14%
Sharpe	-0.12	0.15	0.37	0.30	-0.20
Max	0.56%	1.81%	2.03%	2.25%	2.25%
Min	-0.74%	-2.25%	-2.25%	-2.03%	-1.47%
Accuracy (Hit Ratio)	39%	44%	44%	44%	43%
Precision	49%	50%	51%	50%	50%
Recall	50%	62%	43%	94%	36%
Mean Squared Error	41%	43%	43%	43%	44%
True Positive	22577	30262	20741	45837	17351
True Negatives	21327	18497	28052	3020	30744
False Positives	23172	29702	20147	45169	17387
False Negatives	22375	18490	28011	2896	31332
Sample Size (N)	111654	111654	111654	111654	111654

identify periods which can be conducive to the positive performance of trend strategies. The approach taken in this stage of the trading process was split the dataset into one batch training which consisted of the period from 2002 until 2010 and a test period from 2010 until the end of 2017. The model chosen for trend estimation was a simple random forest decision tree as this type of model does not require a large training dataset and can provide important information about the importance of each feature via the variable importance plots. A simplified tree structure is shown for one of the trend models in 21. In this instance, only three features were used, global economic data momentum, the economic performance differential between the United States and Euro Area and the Economic surprise differential between the US and Euro Area [16]. The decision tree structure of the first split shows that when the economic performance differential between Euro Area and the US has been below -2 standard deviations, then the trend model has under performed. Hence we can see that the goal of the trend estimation model is to align the fundamental factors with periods that the trend capture model has performed well, allowing the second trend estimation model to switch off the trend model when the regime is not conducive to the performance of trend (based on those historical patterns between the trend model performance and the fundamental features).

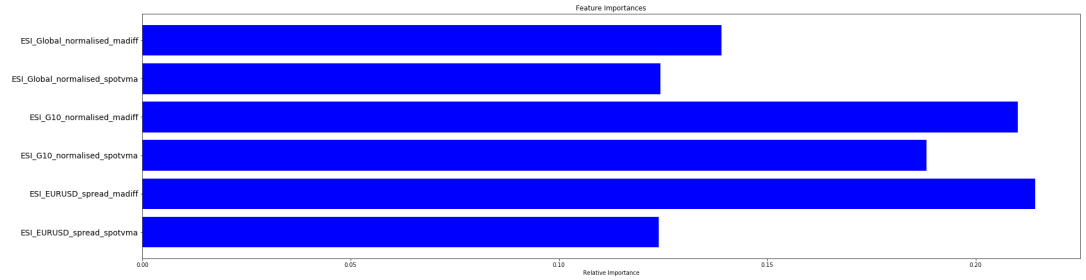
[16] An economic surprise measures the difference between the actual economic release value and the economists consensus expectation

Figure 21: Decision Tree Structure Example of the Trend Estimation Model



When running the trend estimation model, given the high number of possible features (and features combinations) across various different lookback horizons, a variable importance plot was created to try to understand which features may have the most explanatory power. A variable importance plot is created by shuffling the features and running the decision tree model each time without one feature to try to uncover how much value that feature adds to the overall prediction ability of the model.
This feature selection study occurred in the training period only. In this case, one trains using daily data[17] and hence we look to reduce the model dimensionality by selecting a few sub components of the most interesting features. The variable importance chart is shown below in figure 22.

Figure 22: Variable Importance Plots of the Features During Training Period



We can see that certain features do trend to outperform other features in the level of their predictability. This this allows us to reduce the dimensionality of the model as some of the features are likely redundant. The trend estimation model predicted the performance the trend model one month ahead. Figure 23, shows the performance of a selection of trend models from each of the three models (RF, SVM, LSTM) which have been fed into the trend estimation model and trained on how each individual model has performed given the economic, macro risk appetite and volatility features shown above.

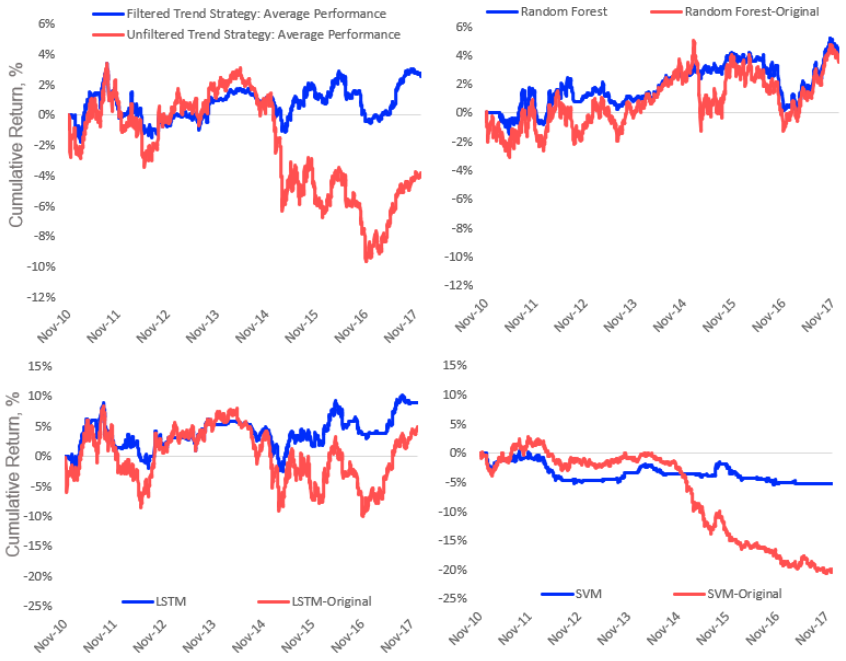
Figure 23, shows the performance statistics of the models once having been filtered by the trend. The testing period ran from December 2010 until December 2017. The red lines denote the average cumulative performance of all the different trend models created using a particular machine learning model. The blue lines denote the average performance across the models tested of the same strategies when they have been filtered by the trend estimation model. The trend estimation model would reduce the exposure of the trend model to zero if it had predicted a loss in the strategy over the next one month.

The performance statistics on average across all three models once having been filtered by the trend estimation model are shown in **??**, on average across all the models, applying trend estimation either increased the average return or reduced a trading strategies' maximum drawdowns (a trading models high to low cumulative losses).

Overall, the two models do tend to work well together and there does seem to be some value in diversifying the decision making process across different models with different specific

[17] As fundamental economic data is only available on a daily time frequency

Figure 23: Performance Model 2 Trend Estimation Across Each ML Model Trading Model



objectives. Future work in this area could involve a method of transfer learning , where the model trains on the randomly generated data first to recognise the basics of trend, and then train on selected assets which have exhibited trending behaviour so that the model has experienced real world data, finally once these patterns have been learned the trend model could then be tested on the an asset price where the tredn capture model would switch that exposure on and off as the regime changes.

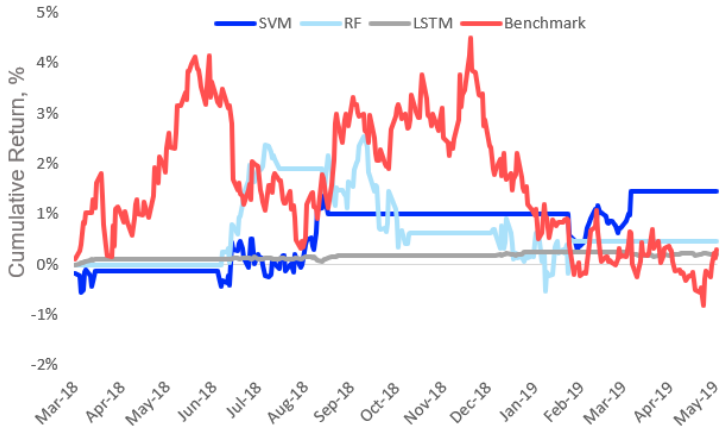
Figure 24: Performance Statistics Model 2 Trend Estimation Average Performance Per Trading Model

	LSTM	RF	SVM
Annual Return	1.25%	0.56%	-0.72%
Annual Volatility	6.80%	2.31%	1.90%
Average Sharpe	0.19	0.19	-0.37
Max	2.8%	0.8%	1.0%
Min	-3.4%	-1.0%	-1.0%
Accuracy (Hit Ratio)	51.7%	50.2%	48.4%
Max Gain/Loss Ratio	0.12	0.04	0.52
Drawdown	-19.3%	-5.9%	-6.8%
Calmar Ratio	-0.07	-0.10	0.11
Mean Squared Error	48.3%	49.6%	50.7%
Deflated Sharpe Test Statistic	47%	30%	3%

6.3 Performance on Blind Test Data

In this section, we look at the performance of the finalised models and model architectures on a fully unseen set of EURUSD data, this dataset was only used in the last stage of the project to provide an indication of how the model may have performed if it had hypothetically been created at the end of 2017 (when the test data ended). We show the performance of the trend model combined with the trend estimation model. The linear benchmark is shown in red (once again, this is the performance of trading in the direction of the previous 24 hour trend in the currency, 25). The model is once again trained on the randomly generated data and

Figure 25: Trading System Performance Fully Unseen Live Data



then tested on EURUSD price from February 2018 until March 2019 using the same models and specifications as outlined earlier. One clear takeaway is that the machine learning models were much less volatile than the benchmark, and two of the machine learning models outperformed the linear benchmark over the full withheld testing period. While there is not enough data to draw statistical conclusions from the blind test results, there is evidence that the model architecture proposed in this project can indeed be used in the domain of currency trading. The fact that the trend estimation model was able to identity the lack of sustained trend allowed the support vector machine to trade trending price behaviour only when the probability of trend working was high.

The model statistics are shown in 26, the accuracy rates and mean squared error percentages of the machine learning trend models after applying the model two trend estimation outperforms the linear benchmark for all models. Each machine learning model also outperforms its linear bench mark in terms of the mean annualised returns with lower standard deviation of returns and a vastly reduced drawdown profile.

Figure 26: Trading System Performance Fully Unseen Live Data

	Benchmark	SVM	RF	LSTM
Average Return	0.16%	0.97%	0.38%	0.15%
Average Volatility	4.79%	1.82%	2.70%	0.16%
Sharpe	0.03	0.53	0.14	0.89
Max	0.90%	0.86%	0.76%	0.04%
Min	-1.11%	-0.49%	-0.83%	-0.04%
Accuracy (Hit Ratio)	47%	52%	53%	58%
Max Gain/Loss Ratio	0.80	1.75	0.92	1.04
Drawdown	-5.32%	-1.10%	-3.09%	-0.09%
Mean Squared Error	53%	48%	47%	43%
Calmar Ratio	0.0	-0.9	-0.1	-1.6

7 Conclusions

The goal of this project was to assess the applicability of machine learning algorithms in the domain of currency trading. This was done by exploring the most appropriate way in which to use machine learning models to predict currency moves. Initially training the model using historical price data did not perform consistently across different models or features. This is in some way not overly surprising given the inherent heteroskedasticity and efficiency of financial markets. One of the main takeaways of the paper is that we cannot expect machine learning algorithms to perform the heavy lifting of price prediction alone. Financial trading is a multi faceted problem and requires the person developing the trading system to have a deep understanding of the subject domain in order to select and find underlying intuitive reasons to justify the model features or parameter specifications.

Creating robust models is key to the longevity of a trading model. Overly complicated algorithms will not be able to withstand the test of time in a trading system as they will have over fitted recent history. To try to avoid some of these pitfalls, this paper delegated trading responsibilities to two separate models. Thereby helping to simplify the problem into two distinct problems, one of trend capture and the second of regime identification. Asking the machine learning model to work on one task (identify trend) allows us to simplify the model structure and narrow the number of features used. Predicting trend as opposed to price removes some of the overfitting risk as we are not trying to predict future return based on a host of features, but only find more efficient and accurate ways to capture price momentum. Machine learning models may outperform linear trend models by being able to identify the non-linearities which may help in the early identification of a trend (either beginning or ending). Selecting this trend capture model also allowed us to train the model on randomly generated trending data, which in future could be expanded to allow the model to include transfer learning algorithms which train on both random data and live prices before proceeding to the test stage.

In the second stage of the trading architecture (regime estimation), a random forest model was used to try to understand the complexities between macro-economic data and currency trends. The results presented showcased that the second model was able to identify regimes conducive to the positive performance of trend strategies. Future work in this area could look at adding additional factors such as a mean reversion model to the trend strategy such that the regime estimation model switches between trend and mean reversion based on the probability of a certain regime occurring. Given the simplicity of the model architecture, this type of framework could be scaled up for a multitude of different models which capture various behaviours and various regimes across different asset classes. This project could also be expanded to include new features measuring investor sentiment, positioning, market trading conditions and seasonality. Adding diverse strategies to the model architecture would translate into a higher probability of generalising into the future regardless of the regime.

Overall, the project outlined one way in which machine learning can be used in the domain of financial trading and showcased how separating out duties to two different models can assist in adding robustness to the trading returns. The results showed promising signs of out of sample success, where the returns were displayed for a host of different strategies and not just the best one. Machine Learning and artificially intelligent systems will likely see their role in financial markets increase over the course of the next decade as the industry evolves, however in terms

of trading algorithms, machine learning methods will remain just one piece of a very large and complex puzzle. There is potential for machine learning methods to outperform standard econometric models however this will still need to be combined with expert domain knowledge, risk management systems, a robust testing framework and data which holds informational value. Above all however, purveyors of the art of financial machine learning will need to remain humble, acknowledge that past performance is rarely an exact guide to the future and that success relies on diversification, hard work and unfortunately...luck.

References

[1] S. Bookman, "15 Supercomputers Less Powerful Than Your Phone." https://www.theclever.com/15-huge-supercomputers-that-were-less-powerful-than-your-smartphone/, 2015. [Online; accessed 02-April-2019].

[2] C. A. MacK, "Fifty years of Moore's law," in *IEEE Transactions on Semiconductor Manufacturing*, 2011.

[3] R. Rebonato, "Asset Management: A Systematic Approach to Factor Investing," *Quantitative Finance*, vol. 17, no. 5, pp. 1–4, 2017.

[4] M. Lopez de Prado, "The 10 Reasons Most Machine Learning Funds Fail," *Journal of Portfolio Management*, 2018.

[5] B. Lebaron, "Chaos and Nonlinear Forecastability in Economics and Finance," *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 1994.

[6] D. H. Bailey, J. Borwein, M. Lopez de Prado, and Q. J. Zhu, "Pseudo-Mathematics and Financial Charlatanism: The Effects of Backtest Overfitting on Out-of-Sample Performance," *American Mathematical Society*, vol. 61, no. 5, pp. 458–471, 2013.

[7] S. M. Bartram, L. Djuranovik, and A. Garratt, "Currency Anomalies," *SSRN Electronic Journal*, 2019.

[8] R. D. Arnott, C. R. Harvey, and H. Markowitz, "A Backtesting Protocol in the Era of Machine Learning," *SSRN Electronic Journal*, 2018.

[9] S. Camargo, S. M. Queirós, and C. Anteneodo, "Bridging stylized facts in finance and data non-stationarities," *European Physical Journal B*, vol. 86, no. 4, 2013.

[10] N. Kaissar, "Hedge Funds have a Performance Problem." https://www.bloomberg.com/opinion/articles/2016-03-24/hedge-funds-have-a-performance-problem, 2016. [Online; accessed 04-April-2019].

[11] A. Corhay and A. T. Rad, "Conditional heteroskedasticity adjusted market model and an event study," *Quarterly Review of Economics and Finance*, 1996.

[12] M. Batwick, "How missing out on 25 days in the stock market over 45 years costs you dearly." https://www.marketwatch.com/story/how-missing-out-on-25-days-in-the-stock-market-over-45-years-costs-you-dearly-2016-01-25, 2016. [Online; accessed 31-March-2019].

[13] N. N. Taleb, "Black Swans and the domains of statistics," *American Statistician*, 2007.

[14] M. Greenwood, "In Persuit of Positive Skew." https://www.merian.com/gb/en/institutional/insights/in-pursuit-of-positive-skew/, 2017. [Online; accessed 11-September-2019].

[15] W. Huang, Y. Nakamori, and S. Y. Wang, "Forecasting stock market movement direction with support vector machine," *Computers and Operations Research*, 2005.

[16] S. Shen, H. Jiang, and T. Zhang, "Stock market forecasting using machine learning algorithms," 2012.

[17] S. Wang and W. Shang, "Forecasting direction of China security index 300 movement with least squares support vector machine," in *Procedia Computer Science*, 2014.

[18] B. Widrow and M. A. Lehr, "30 Years of Adaptive Neural Networks: Perceptron, Madaline, and Backpropagation," *Proceedings of the IEEE*, 1990.

[19] F. Rosenblatt, "The perceptron: A probabilistic model for information storage and organization in the brain," *Psychological Review*, 1958.

[20] V. Kůrková, "Kolmogorov's theorem and multilayer neural networks," *Neural Networks*, 1992.

[21] S. Hochreiter and J. Schmidhuber, "Long Short-Term Memory," *Neural Computation*, 1997.

[22] T. Fischer and C. Krauss, "Deep learning with long short-term memory networks for financial market predictions," *European Journal of Operational Research*, 2018.

[23] B. Lim, S. Zohren, and S. Roberts, "Enhancing Time Series Momentum Strategies Using Deep Neural Networks," *SSRN Electronic Journal*, 2019.

[24] D. Kucharczyk, J. Osterrieder, S. Rudolf, and D. Wittwer, "Neural Networks and Arbitrage in the VIX – A Deep Learning Approach for the VIX," *SSRN Electronic Journal*, 2019.

[25] P. Czekalski, M. Niezabitowski, and R. Styblinski, "ANN for FOREX forecasting and trading," in *Proceedings - 2015 20th International Conference on Control Systems and Computer Science, CSCS 2015*, 2015.

[26] K. Joarder and R. A. Sarker, "Forecasting of currency exchange rates using ANN: A case study," in *Proceedings of 2003 International Conference on Neural Networks and Signal Processing, ICNNSP'03*, 2003.

[27] H. Gunduz, Y. Yaslan, and Z. Cataltepe, "Intraday prediction of Borsa Istanbul using convolutional neural networks and feature correlations," *Knowledge-Based Systems*, vol. 137, pp. 138–148, 2017.

[28] K.-j. Kim, "Predicting Stock Price Direction using Support Vector Machines," *Neurocomputing*, 2003.

[29] M. Kumar and M. Thenmozhi, "Forecasting Stock Index Movement: A Comparison of Support Vector Machines and Random Forest." 2006.

[30] K. J. Kim and I. Han, "Genetic algorithms approach to feature discretization in artificial neural networks for the prediction of stock price index," *Expert Systems with Applications*, 2000.

[31] C. Cortes and V. Vapnik, "Support-Vector Networks," *Machine Learning*, 1995.

[32] D. Wilimitis, "The Kernel Trick in Support Vector Classification." `https://towardsdatascience.com/the-kernel-trick-c98cdbcaeb3f`, 2018. [Online; accessed 07-April-2019].

[33] G. James, D. Witten, T. Hastie, and R. Tibshirani, *An Introduction to Statistical Learning*. 2013.

[34] Y. Kara, M. Acar Boyacioglu, and Ö. K. Baykan, "Predicting direction of stock price index movement using artificial neural networks and support vector machines: The sample of the Istanbul Stock Exchange," *Expert Systems with Applications*, 2011.

[35] J. Patel, S. Shah, P. Thakkar, and K. Kotecha, "Predicting stock and stock price index movement using Trend Deterministic Data Preparation and machine learning techniques," *Expert Systems with Applications*, 2015.

[36] B. Gui, X. Wei, Q. Shen, J. Qi, and L. Guo, "Financial time series forecasting using support vector machine," in *Proceedings - 2014 10th International Conference on Computational Intelligence and Security, CIS 2014*, 2015.

[37] J. Kamruzzaman, R. Sarker, and I. Ahmad, "SVM based models for predicting foreign currency exchange rates," 2004.

[38] V. Podgorelec and M. Zorman, "Decision Tree Learning," in *Encyclopedia of Complexity and Systems Science*, 2015.

[39] L. Breiman, "Random Forrests," *Machine learning*, 2001.

[40] R. Genuer, "Variance reduction in purely random forests," *Journal of Nonparametric Statistics*, 2012.

[41] S. P. Chatzis, V. Siakoulis, A. Petropoulos, E. Stavroulakis, and N. Vlachogiannakis, "Forecasting stock market crisis events using deep and statistical machine learning techniques," *Expert Systems with Applications*, 2018.

[42] W. McKinney, *Python for Data Analysis: Data Wrangling with Pandas, NumPy, and IPython*. 1976.

[43] S. Van Der Walt, S. C. Colbert, and G. Varoquaux, "The NumPy array: A structure for efficient numerical computation," *Computing in Science and Engineering*, 2011.

[44] D. R. Tobergte and S. Curtis, "Learning SciPy for Numerical and Scientific Computing Second Edition," *Journal of Chemical Information and Modeling*, 2013.

[45] A. Géron, *Hands-On Machine Learning with Scikit-Learn and TensorFlow*. 2017.

[46] M. A. Wood, *Python and Matplotlib Essentials for Scientists and Engineers*. 2015.

[47] M. I. Reiff, "Pandas," *Journal of Developmental and Behavioral Pediatrics*, 2002.

[48] F. Chollet, "Keras: The Python Deep Learning library," 2015.

[49] I. Filippou and M. P. Taylor, "Common Macro Factors and Currency Premia," *Journal of Financial and Quantitative Analysis*, 2017.

[50] B. Yoann, "The Truth Nobody Wants to Tell You About AI for Trading." `https://hackernoon.com/https-medium\-com-supernova-the-truth-nobody-wants-to-tell-you-about-ai-for-trading-5d29a297ee93`, 2019. [Online; accessed 31-March-2019].

[51] M. Dahlquist and H. Hasseltoft, "Economic Momentum and Currency Returns." 2015.

[52] T. J. Moskowitz, Y. H. Ooi, and L. H. Pedersen, "Time series momentum," *Journal of Financial Economics*, 2012.

[53] B. Döme, "Efficiency and Return Predictability." 2008.

[54] G. Abreu, R. Neves, and N. Horta, "Currency exchange prediction using machine learning, genetic algorithms and technical analysis," 2018.

[55] W. F. Sharpe, "The Sharpe Ratio," *The Journal of Portfolio Management*, 2009.

[56] D. H. Bailey and M. Lopez de Prado, "The Deflated Sharpe Ratio: Correcting for Selection Bias, Backtest Overfitting and Non-Normality," *SSRN*, 2014.

[57] E. Sciulli, "Is There Any Alpha Left in FX." `https://www.man.com/maninstitute/is-there-any-alpha-left-in-FX`, 2015. [Online; accessed 10-September-2019].

[58] R. D. Mclean and J. Pontiff, "Does Academic Research Destroy Stock Return Predictability?," *Journal of Finance*, 2016.

A Appendix

A.1 List of Data Files Included in Project

Table 4: Data Files Included in Report

File Name	Desc.
eurusd_hour.csv	EURUSD Hourly Price
CurrencyData.csv	G10FX Hourly Price
EDI.csv	Econ Data
ESI.csv	Econ Data
CcyrandomTrend.csv	Randomly Generated Data
CcyrandomTrendLSTM.csv	Randomly Generated Data
CcyDataLONGER.csv	Data with Features
ccyDataBLIND_STFEATURES.csv	Withheld Data Features
trendestimation.csv	Data for Trend Estimation
trendestimation_manual.xlsx	Data for Trend Estimation
trendestimation_manual_BLIND.xlsx	Withheld Data, for Trend Estimation
Signals.csv	Raw Signals for Trend Estimation
Signals_blindData.csv	Withheld Data, Raw Signals for Trend Estimation
Signals_FullsamplewBlind.csv	Full Sample Signals For Trend Estimation

A.2 Code Used in the Project

Listing 1: Code to Generate Random Trending Data

```
'''
Author: Ed Gill
This is a process to return a randomly generated series of numbers.
change alpha from -0.2 to 0.2 to move from mean reversion to strong trend.
'''

import numpy as np
import pandas as pd
from create_model_features import trends_features
def generate_trend(n_samples, alpha, sigma):
    '''

    :return: Generate a trend
    '''
   # ( range from -0.2 to 0.2 to move from mean reversion to strong trend
    trend_param = (1 / (1 - (alpha** 3)))
    x = w = np.random.normal(size=n_samples)*sigma
    for t in range(n_samples):
        x[t] = trend_param*x[t - 1] + w[t]
    # return the trend file as a a dataframe
    trendy_ts = pd.DataFrame(x, columns = ["trend"])
    return trendy_ts
```

```
def get_trendy_data(n_samples,trend_strength,pct_stdev,CCY_COL, short, medium, long,
    longest, medium_multiplier,long_multplier):
    '''
    Takes the trendy series and gets the model features
    :return:
    '''
    trendy_df = generate_trend(n_samples, trend_strength, pct_stdev)
    ccy_data = trends_features(trendy_df, CCY_COL, short, medium, long, longest,
        medium_multiplier, long_multplier)
    ccy_data['CCY'] = trendy_df['trend']
    # need to replace log ret with simpel return
    ccy_data['logret'] = trendy_df['CCY'] - trendy_df['CCY'].shift(1)
    return ccy_data.replace(np.nan, 0)
```

Listing 2: Code to Create Model Features

```
'''
Author: Ed Gill
This file will create the model features for the trend model.

'''
import datetime
import numpy as np

short = 5
medium = 15
long = 55
longest = 100

def trends_features(data,CCY_COL, short, medium, long, longest,
    medium_multiplier,long_multplier):
    '''
    This function will calculate each trend feature and return a dataframe
    :param data: data frame with all the columns as standard
    :param short:
    :param medium:
    :param long:
    :param longest:
    :param medium_multiplyer: This will cycle through the number of peridos that the
        medium term window uses. .i.e 24 would make everything a day
    :param long_multiplyer: at 120, long mult means we look in terms of weeks. each
        period is now one business week, 24*5
    :return:
    '''
    if CCY_COL not in list(data.columns):
        print("Column name not in the dataframe")
        return data
    data["logret"] = np.log(data[CCY_COL]) - np.log(data[CCY_COL].shift(1))
    # TODO: Should this be an EMA or simple average? Using EWMA now as we
    # overweight recent history
    # HF = high frequency
```

```
data['HF_short'] = data[CCY_COL].ewm(short).mean()
data['HF_medium'] = data[CCY_COL].ewm(medium).mean()
data['HF_long'] = data[CCY_COL].ewm(long).mean()
data['HF_longest'] = data[CCY_COL].ewm(longest).mean()
# differences to spot
data['spot_v_HF_short'] = data[CCY_COL] - data['HF_short']
data['spot_v_HF_medium'] = data[CCY_COL] - data['HF_medium']
data['spot_v_HF_long'] = data[CCY_COL] - data['HF_long']
data['spot_v_HF_longest'] = data[CCY_COL] - data['HF_longest']

# medium frequency factors, multiplyer allows us to scale up the lookback as
    needed.
# days to weeks
data['MF_short'] = data[CCY_COL].ewm(short*medium_multiplier).mean()
data['MF_medium'] = data[CCY_COL].ewm(medium*medium_multiplier).mean()
data['MF_long'] = data[CCY_COL].ewm(long*medium_multiplier).mean()
data['MF_longest'] = data[CCY_COL].ewm(longest*medium_multiplier).mean()
# differences to spot
# to measure relative momentum
data['spot_v_MF_short'] = data[CCY_COL] - data['MF_short']
data['spot_v_MF_medium'] = data[CCY_COL] - data['MF_medium']
data['spot_v_MF_long'] = data[CCY_COL] - data['MF_long']
data['spot_v_MF_longest'] = data[CCY_COL] - data['MF_longest']
# long term factors
# weeks to months
data['LF_short'] = data[CCY_COL].ewm(short*long_multplier).mean()
data['LF_medium'] = data[CCY_COL].ewm(medium*long_multplier).mean()
data['LF_long'] = data[CCY_COL].ewm(long*long_multplier).mean()
data['LF_longest'] = data[CCY_COL].ewm(longest*long_multplier).mean()
# differences to spot
# to measure relative momentum
data['spot_v_LF_short'] = data[CCY_COL] - data['LF_short']
data['spot_v_LF_medium'] = data[CCY_COL] - data['LF_medium']
data['spot_v_LF_long'] = data[CCY_COL] - data['LF_long']
data['spot_v_LF_longest'] = data[CCY_COL] - data['LF_longest']

# average of both spot distance and each ema distance
# take simple average of the divergences at each time frame
data['spot_v_HF'] = (data['spot_v_HF_short'] + data['spot_v_HF_medium'] +
    data['spot_v_HF_long'] + data['spot_v_HF_longest'])/4
data['spot_v_MF'] = (data['spot_v_MF_short'] + data['spot_v_MF_medium'] +
    data['spot_v_MF_long'] + data['spot_v_MF_longest'])/4
data['spot_v_LF'] = (data['spot_v_LF_short'] + data['spot_v_LF_medium'] +
    data['spot_v_LF_long'] + data['spot_v_LF_longest'])/4
#differences to each ema
# This can capture the divergences between the EMAs, which allows us to grasp the
    speed of the move
data['HF_ema_diff'] = (data['HF_short']-data['HF_medium']) +
    (data['HF_medium']-data['HF_long']) + (data['HF_long']-data['HF_longest'])
data['MF_ema_diff'] = (data['MF_short']-data['MF_medium']) +
    (data['MF_medium']-data['MF_long']) + (data['MF_long']-data['MF_longest'])
data['LF_ema_diff'] = (data['LF_short']-data['LF_medium']) +
    (data['LF_medium']-data['LF_long']) + (data['LF_long']-data['LF_longest'])
```

```
    return data

def add_timezones(data):
    '''
    Timezones can exhibit trendy or mean reverting behaviour, Hence we can add this as
        a feature to the model.
    :return: A data frame where we have added the timezones
    '''
    # Add in hourly feature times. Think this is important as there can be certain
        patterns that occur into and out
    # of these time frames
    # London and NY liquid hours
    data['LDN'] = 0
    data['NY'] = 0
    data['Asia'] = 0
    # adding in timezone changes
    data['LDN'].loc[(data["timestamp"] >= datetime.time(7,0)) & (data["timestamp"] <=
        datetime.time(12,0))] = 1
    data['LDN'].loc[(data["timestamp"] >= datetime.time(13,0)) & (data["timestamp"] <=
        datetime.time(17,0))] = 0.5
    data['NY'].loc[(data["timestamp"] >= datetime.time(13,0)) & (data["timestamp"] <=
        datetime.time(17,0))] = 0.5
    data['NY'].loc[(data["timestamp"] >= datetime.time(18,0)) & (data["timestamp"] <=
        datetime.time(22,0))] = 1
    data['Asia'].loc[(data["timestamp"] >= datetime.time(23,0))] = 1
    data['Asia'].loc[(data["timestamp"] <= datetime.time(6,0))] = 1
    return data
```

Listing 3: Code Containing Model Functions

```
'''
author: Ed Gill

This file contains the neccessary modules for creating the training and testing files
    for the machine learnign algorithm.
'''

# import neccessary modules to perpare the data for entry to ML model.
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix
from scipy.stats import norm
from sklearn.decomposition import PCA


def create_train_test_file(data_file, data_size, test_split,
    test_buffer,concat_results):
    '''
    This module will create the training and testing files to be used in the ML models.
    :param data_file: DF with price data
```

```
    param: data_size: size of the training perios
    param: test_split: size of the testing period
    param: test_buffer: the time between training and testing, to ensure no data
        leakage from train to test.
    param: concat_results: If True, we then use a walk forward type methodology for
        training and testing.
    :return: training and testing data fils.
    '''
    # Check we have enough data to train with compared to the data size
    if data_size > data_file.shape[0]:
        # Overwrite the data_length to be 90% f file, with remaining 10% as train
        data_size = int(data_file.shape[0]*0.9)
        # adding a buffer of 5 forward steps before we start trading on test data
        test_size = data_file.shape[0] - (data_size + test_buffer)
    else:
        # if a percentage test split is provided , then transform this to actual no.
            of points.
        if test_split <= 1:
            test_size = int(data_size*test_split)
        else:
            # if a whole number, this is accepted as the number of test points to use.
            test_size = test_split
    # training size is the first x data points and the test data is appended onto the
        train data
    # such that we use a walk forward testing framework
    if concat_results:
        # provide data up till the test data zone
        train_data = int(data_file.shape[0]) - (test_size + test_buffer)
        train_original = data_file.iloc[:train_data, :].reset_index(drop = True)
        # provide data in the last x points of test data available
        test_original = data_file.iloc[-test_size:, :].reset_index(drop = True)
    else:
        train_original = data_file.iloc[:int(data_size), :].reset_index(drop= True)
        test_original = data_file.iloc[int(data_size) + test_buffer: (int(data_size) +
            int(test_size)), :].reset_index(drop= True)
    # return two separate data files for training/testing
    return train_original, test_original

def standardise_data(dataset, full_cols, standardised_cols,window):
    '''
    This function computes the standardised returns on a rolling basis looking
        backwards over the window specified.
    This is the most realistic in terms of a trading strategy and also means the test
        data is standardised on the correct basis using the
    latest data available at each timestep.
    :param dataframe:
    :param cols:
    :return: standardised dataframe of values
    '''
    train_standardised =
        dataset[standardised_cols].subtract(dataset[standardised_cols].rolling(window).mean())
    train_standardised =
        train_standardised.divide(dataset[standardised_cols].rolling(window).std())
```

```
    # we will only return the data which is outide the initial window standardisation
        period
    # add non standardised features
    for feature in full_cols:
        if feature not in standardised_cols:
            train_standardised[feature] = dataset[feature]
    # This function now returns the neccessary file with both standard and non
        standardised columns.
    return train_standardised.loc[window:,:]

def calculate_target(data_df,trade_horizon,use_risk_adjusted):
    '''
    Take the raw dataseries of log returns.
    :param data_df:
    :param horizon:
    :param use_risk_adjusted:
    :return: return the risk adjusted return or the raw percent ahead return
    '''
    # Using a shift = 2 so that the forward return starts from exactly the next future
        time step.
    if use_risk_adjusted:
        return
            data_df['logret'].iloc[::-1].shift(2).rolling(trade_horizon).sum().values[::-1]/data_df['lo
    else:
        return
            data_df['logret'].iloc[::-1].shift(2).rolling(trade_horizon).sum().values[::-1]



def create_dataset(dataset, populate_target, look_back, test):
    '''
    This creates the data for passing to the LSTM module
    :param dataset:
    :param populate_target:
    :param look_back:
    :return:
    '''
    dataX, dataY, target_dates = [], [], []
    for i in range(len(dataset) - look_back + 1):
        # this takes the very last col as the target
        a = dataset[i:(i + look_back), :-1]
        dataX.append(a)
        # this code assumes that the target vector is the very last col.
        dataY.append(dataset[i + look_back - 1, -1])
        if populate_target:
            target_dates.append(test['Date'].loc[i + look_back - 1])
    return np.array(dataX), np.array(dataY), target_dates

def signal(output, thold):
    '''
    :param x: Create a signal from the predicted softmax activation output
    :return: signal to trade
    '''
```

```
    if output >= thold:
        return 1
    elif output <= (1-thold):
        return -1
    else:
        return 0

def get_accuracy(predicted, test_target ):
    '''
    :return: the prediction accuracy of our model
    '''
    return accuracy_score(test_target, predicted)

def get_scaled_returns():
    '''
    This file will scale exposure based on the next 24 hour ahead prediction
    :return:
    '''
    pass

def erf(row_value):
    ''''
    This applies the error function to smoooth the risk adjusted return prediction
    mapps all numbers from -1 to + 1
    '''
    return (2*(1/(1 + np.exp(-row_value)))-1)

def get_total_data_needed(test_split, data_size,test_buffer):
    '''

    :param test_split:
    :return: the data size needed for all computations
    '''
    if test_split <1 :
        return int(data_size*(1 + test_split)) + test_buffer
    else:
        return int(data_size + test_split + test_buffer)

def get_pca_features(train,test, features_to_standardise, use_pca):
    '''
    This file outputs the PCA vectors of the model to the number of features needed.
    :param data_file:
    :param model_features:
    :param output_feature:
    :return:
    '''
    pca = PCA(n_components=use_pca)
    # find the PCs
    pca = pca.fit(train[features_to_standardise])
    pca_train = pca.transform(train[features_to_standardise])
    pca_test = pca.transform(test[features_to_standardise])
    labels = ['PC%s' % i for i in range(1, use_pca + 1)]
    # add the pc values to the train and test model
```

```
    pc_number = 0
    for label in labels:
        train[label] = pd.DataFrame(pca_train[:,pc_number])
        test[label] = pd.DataFrame(pca_test[:, pc_number])
        pc_number += 1
    # Find the variance explained within each PC
    var_exp = pca.explained_variance_ratio_
    # return train , test and var explained of the pca
    return train, test, var_exp

def max_exp_sharpe(avg_sharpe, var_sharpe, ntrials):
    '''
    Link to code is from -->
        https://gmarti.gitlab.io/qfin/2018/05/30/deflated-sharpe-ratio.html
    :param mean_sharpe:
    :param var_sharpe:
    :param nb_trials:
    :return:
    '''
    gamma = 0.5772156649015328606
    e = np.exp(1)
    return avg_sharpe + np.sqrt(var_sharpe) * (
        (1 - gamma) * norm.ppf(1 - 1 / ntrials) + gamma * norm.ppf(1 - 1 / (ntrials *
            e)))

def dsr(expected_sharpe,sharpe_var,ntrials,sample_length,skew,kurtosis):
    '''
    Calculate the deflated sharpe
    :param expected_sharpe:
    :param sharpe_var:
    :param ntrials:
    :param sample_length:
    :param skew:
    :param kurtosis:
    :return:
    '''
    SR_zero = max_exp_sharpe(0, sharpe_var, ntrials)

    return norm.cdf(((expected_sharpe - SR_zero) * np.sqrt(sample_length - 1))
                    / np.sqrt(1 - skew * expected_sharpe + ((kurtosis - 1) / 4) *
                        expected_sharpe ** 2))


def main():
    pass

if __name__ == "__main__":
    main()
```

Listing 4: Code to Set Parameters of the Models

```
''''
```

```
This file runs the decision tree module and then returns the train test results'''
# import nsccessary modules
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
import datetime
from sklearn import tree
from model_functions import get_accuracy, erf, standardise_data, calculate_target
from sklearn.ensemble import RandomForestClassifier, RandomForestRegressor
from sklearn import metrics, svm

def decision_tree(train, test,use_classifier, use_risk_adjusted,ntree, max_features,
    max_depth):
    # take the
    X = train.iloc[:, :-1]
    X_test = test.iloc[:, :-1]
    if use_classifier:
        Y = train["target"].apply(np.sign)
        Y_test = test["target"].apply(np.sign)
    else:
        # using risk adjusted return- oontinous value
        Y = train["target"]
        Y_test = test["target"]
    # clean the data and nan values
    X = X.replace(np.nan, 0)
    Y = Y.replace(np.nan, 0)
    Y = Y.replace(np.inf, 0)
    X_test = X_test.replace(np.nan, 0)
    Y_test = Y_test.replace(np.nan, 0)
    Y_test = Y_test.replace(np.inf, 0)
    if use_classifier:
        RF = RandomForestClassifier(n_estimators=ntree, max_features=
            max_features,max_depth = max_depth, verbose=0)
        # clf = tree.DecisionTreeClassifier(max_leaf_nodes = 6, max_depth = 8)
    else:
        # ass in code for regresion classifier
        RF = RandomForestRegressor(n_estimators=ntree, max_features= max_features,
            max_depth = max_depth,verbose=0)
    RF.fit(X, Y)
    # run training on the test data
    results = RF.predict(X_test)
    # The % threshold needed to trigger a signal either way
    if use_risk_adjusted:
        acc_score = metrics.mean_squared_error(Y_test, results) # cant get a
            classifier, so need to print simple mse
    else:
        acc_score = get_accuracy(Y_test, results)
    return (results, acc_score)

def backtester(results, test, trade_horizon):
    '''
    Calculate the returns of the trading strategy and store the results in the test
        file.
```

```
    :param results: the trading signals generated by the model.
    :param test: this contains the test data
    :param trade_horizon: this is the length of time of the trading signal prediction
    :return: A dataframe which has the test data and also the results of the trading
        strategy in pct terms of log returns.
    '''
    # This needs to change to handle the change in the target
    predictions = pd.DataFrame({"Date": test['Date'], "Predictions": results})
    test_results = pd.merge(test, predictions, how="left", on="Date").fillna(0)
    # calculate the returns of the signal
    test_results["erf_signal"] = test_results['Predictions'].apply(erf)
    test_results["scaled_signal"] =
        test_results['Predictions'].shift(2).rolling(trade_horizon).sum() /
        trade_horizon
    test_results["scaled_signal_erf"] =
        test_results['erf_signal'].shift(2).rolling(trade_horizon).sum() /
        trade_horizon
    # no shift needed as we have already done that in previous step
    test_results['strat_returns'] = test_results['logret'] *
        test_results['scaled_signal']
    test_results['strat_returns_sum'] = test_results['strat_returns'].cumsum()
    # calculate returns for the error function
    test_results['strat_returns_erf'] = test_results['logret'] *
        test_results['scaled_signal_erf']
    test_results['strat_returns_sum_erf'] = test_results['strat_returns_erf'].cumsum()
    strat_return = test_results['strat_returns'].sum()
    # provide the information ratio on an annualised basis.
    information_ratio = (test_results['strat_returns'].mean() * 260) /
        (test_results['strat_returns'].std() * np.sqrt(260))
    return (test_results, strat_return, information_ratio)

def update_performance(data_size,ntree, acc_score , information_ratio, run_time,
    train_date, test_date, performance_store):
    # Store the data as needed
    performance_store['data_size'].append(data_size)
    performance_store['ntree'].append(ntree)
    performance_store['Accuracy_Score'].append(acc_score)
    performance_store['Info_Ratio'].append(information_ratio)
    performance_store['run_time'].append(run_time)
    performance_store['train_date_st'].append(train_date)
    performance_store['test_date_st'].append(test_date)
    return pd.DataFrame(performance_store)

def initialise_process(file_location, trade_horizon, window, use_risk_adjusted,
    use_pca,use_random_train_data ):
    '''
    This re freshes the whole data set as needed by the ipython process
    this is the function to modify if you want different features in the model.
    :return: data_normed df with standardised values and model features to use
    '''
    data_file = pd.read_csv(file_location) #
        pd.read_csv(r"/storage/eurusd_train_normed.csv")
    data_file = data_file.replace(np.nan, 0)
```

```
    ############################ Set Model Paramaters ############################
    # this looks back over a set period as the memory for the LSTM
    model_features = ["spot_v_HF", "spot_v_MF", "spot_v_LF", "HF_ema_diff",
                      "MF_ema_diff", "LF_ema_diff", "target"] # "LDN", "NY", "Asia"
                          removed, # target must be kept at the end
    ################### Standardise Entire Dataset using rolling lookback windows
        ###############
    features_to_standardise = ["spot_v_HF", "spot_v_MF", "spot_v_LF", "HF_ema_diff",
                               "MF_ema_diff", "LF_ema_diff"]
    ###### Set Targets ##############
    data_file["target"] = calculate_target(data_file, trade_horizon, use_risk_adjusted)
    # Remove infinity from the values, division by 0
    data_file["target"] = data_file["target"].replace(np.inf, 0)
    data_file['target'] = data_file["target"].replace(-np.inf, 0)
    data_file['target'] = data_file["target"].replace(np.nan, 0)
    # roughly 3 yrs of data slightly less actually
    data_normed = standardise_data(data_file, model_features, features_to_standardise,
        window)
    # add extra features non standardised, check we are using random or non random data
    if not use_random_train_data:
        data_normed['Date'] = data_file['Date'].iloc[window:]
    data_normed['CCY'] = data_file['CCY'].iloc[window:]
    data_normed['logret'] = data_file['logret'].iloc[window:]
    if use_pca > 0:
        # if we are using pca features, then model features need only to be PC1 and
            PC2 etc plus the target
        model_features = ['PC%s' %i for i in range(1,use_pca+1)]
        model_features.append("target")
    return data_normed.reset_index(drop = True), model_features,
        features_to_standardise

def run_svm_model(train, test,use_classifier, use_risk_adjusted,kernel,cost):
    # This duplicates alot of code in the Dec tree module, so maybe think about
        removing these duplications
    X = train.iloc[:, :-1]
    X_test = test.iloc[:, :-1]
    if use_classifier:
        Y = train["target"].apply(np.sign)
        Y_test = test["target"].apply(np.sign)
    else:
        # using risk adjusted return- continous value
        Y = train["target"]
        Y_test = test["target"]
    # clean the data and nan values
    X = X.replace(np.nan, 0)
    Y = Y.replace(np.nan, 0)
    Y = Y.replace(np.inf, 0)
    X_test = X_test.replace(np.nan, 0)
    Y_test = Y_test.replace(np.nan, 0)
    Y_test = Y_test.replace(np.inf, 0)
    if use_classifier:
        SVM = svm.SVC(kernel=kernel, C = cost)
        # clf = tree.DecisionTreeClassifier(max_leaf_nodes = 6, max_depth = 8)
```

```
    else:
        # ass in code for regresion classifier
        SVM = svm.SVR(kernel= kernel, C = cost)
    SVM.fit(X, Y)
    # run training on the test data
    results = SVM.predict(X_test)
    # The % threshold needed to trigger a signal either way
    if use_risk_adjusted:
        acc_score = metrics.mean_squared_error(Y_test, results) # cant get a
            classifier, so need to print simple mse
    else:
        acc_score = get_accuracy(Y_test, results)
    return (results, acc_score)

def set_params_random_forests():
    '''
    This is the control center for all the params that need to be set in the RF modules
    :return: return all params as they have been set here
    '''
    ############################ Set Model Paramaters ##############################
    param_dict = {"ntrees" : [200], "max_features" : 4, "test_buffer" : 5, "max_depth"
        : 30 , "data_size" : 15000 ,
                  "concat_results" : False, "test_split" : 0.25, "thold" : 0.55,
                      "window" : 1000, "trade_horizon" : 24,
                  "use_risk_adjusted" : True , "use_binary" : False, "use_classifier" :
                      True, "use_pca" : 0,
                  "use_separated_chunk" : False, "use_random_train_data" : True,
                      "use_RF": True}
    # this looks back over a set period as the memory for the LSTM
      # [i for i in range(25,301,25)] # [21, 66]
    # if running pca, max features can only be same or less than the full total of
        features
    return param_dict

def set_params_svm():
    '''
    Additional params only applicable to the svm code.
    :return:
    '''
    svm_dict = {'kernel':"linear" , "cost": [1]}
    return svm_dict

def set_params_LSTM():
    '''
    Additional params only applicable to the RF code
    :return:
    '''
    return {'EPOCH' : 500, 'first_layer': 32, 'second_layer': 16, 'look_back' : 66 }

def set_params_trend_estimate():
    '''
    Additional params only applicable to the RF code
    :return:
```

```
    '''
    return {'trade_horizon' : 21, 'std_window': 260, 'train_size': 0.92, 'test_split'
        : 1, 'test_buffer': 1 }

def main():
    '''

    :return:
    '''
    pass


if "__main__" == __name__:
        main()
```

Listing 5: Code to Set Parameters for the Keras & Tensor flow modules

```
'''
author: Ed Gill

This file contains the neccessary modules for creating the training and testing files
    for the machine learnign algorithm.
'''

# import neccessary modules to perpare the data for entry to ML model.
from keras.models import Sequential
from keras.layers import LSTM
from keras.layers import Dense
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix
from sklearn.decomposition import PCA

def create_train_test_file(data_file, data_size, test_split,
    test_buffer,concat_results):
    '''
    This module will create the traingin and testing files to be used in the ML RNN
        model.
    :return: training and testing data fils.
    '''
    # use a long term rolling standardisation window

    # How large should the training data be?
    if data_size > data_file.shape[0]:
        # Overwrite the data_length to be 90% f file, with remaining 10% as train
        data_size = int(data_file.shape[0]*0.9)
        # adding a buffer of 5 forward steps before we start trading on test data
        test_size = data_file.shape[0] - (data_size + test_buffer)
    else:
        if test_split < 1:
```

```
            test_size = int(data_size*test_split)
        else:
            # if a whole number, this is accepted as the number of test points to use.
            test_size = test_split
    # training size is the first x data points
    if concat_results:
        # provide data up till the test data zone
        train_data = int(data_file.shape[0]) - (test_size + test_buffer)
        train_original = data_file.iloc[:train_data, :].reset_index(drop= True)
        # provide data in the last x points of test data
        test_original = data_file.iloc[-test_size:, :].reset_index(drop= True)
    else:
        train_original = data_file.iloc[:int(data_size), :].reset_index(drop= True) #
            eurusd_train.iloc[-DATA_SIZE:,:]
        test_original = data_file.iloc[int(data_size) + test_buffer: (int(data_size) +
            int(test_size)), :].reset_index(drop= True)
    return train_original , test_original

def standardise_data(dataset, full_cols, standardised_cols,window):
    '''
    This function computes the standardised returns on a rolling basis backwards.
    This si most realistic in term sof a trading strategy and also means the test data
        is standardised on the correct basis using the
    latest data available at each timestep.
    :param dataframe:
    :param cols:
    :return:
    '''
    train_standardised =
        dataset[standardised_cols].subtract(dataset[standardised_cols].rolling(window).mean())
    train_standardised =
        train_standardised.divide(dataset[standardised_cols].rolling(window).std())
    # we will only return the data which is outide the initial window standardisation
        period
    # add non standardised features
    for feature in full_cols:
        if feature not in standardised_cols:
            train_standardised[feature] = dataset[feature]
    # This function now returns the neccessary file with both standard and non
        standardised columns.
    return train_standardised.loc[window:,:]

def calculate_target(data_df,trade_horizon,use_risk_adjusted):
    '''
    Take the raw dataseries of log returns.
    :param data_df:
    :param horizon:
    :param use_risk_adjusted:
    :return: return the risk adjusted return or the raw percent ahead return
    '''
    # Using a shift = 2 so that the forward return starts from exactly the next future
        time step.
    if use_risk_adjusted:
```

```
        return
            data_df['logret'].iloc[::-1].shift(2).rolling(trade_horizon).sum().values[::-1]/data_df['lo
    else:
        return
            data_df['logret'].iloc[::-1].shift(2).rolling(trade_horizon).sum().values[::-1]



def create_dataset(dataset, populate_target, look_back, test):
    '''
    This creates the data for passing to the LSTM module
    :param dataset:
    :param populate_target:
    :param look_back:
    :return:
    '''
    dataX, dataY, target_dates = [], [], []
    for i in range(len(dataset) - look_back + 1):
        # this takes the very last col as the target
        a = dataset[i:(i + look_back), :-1]
        dataX.append(a)
        # this code assumes that the target vector is the very last col.
        dataY.append(dataset[i + look_back - 1, -1])
        if populate_target:
            target_dates.append(test['Date'].loc[i + look_back - 1])
    return np.array(dataX), np.array(dataY), target_dates

def signal(output, thold):
    '''
    :param x: Create a signal from the predicted softmax activation output
    :return: signal to trade
    '''
    if output >= thold:
        return 1
    elif output <= (1-thold):
        return -1
    else:
        return 0

def get_accuracy(predicted, test_target ):
    '''
    :return: the prediction accuracy of our model
    '''
    true_class = [np.sign(i[0]) for i in test_target]
    return accuracy_score(true_class, predicted)

def get_scaled_returns():
    '''
    This file will scale exposure based on the next 24 hour ahead prediction
    :return:
    '''
    pass
```

```
def get_pca_features(train,test, features_to_standardise, use_pca):
    '''
    This file outputs the PCA vectors of the model to the number of features needed.
    :param data_file:
    :param model_features:
    :param output_feature:
    :return:
    '''
    pca = PCA(n_components=use_pca)
    # find the PCs
    pca = pca.fit(train[features_to_standardise])
    pca_train = pca.transform(train[features_to_standardise])
    pca_test = pca.transform(test[features_to_standardise])
    labels = ['PC%s' % i for i in range(1, use_pca + 1)]
    # add the pc values to the train and test model
    pc_number = 0
    for label in labels:
        train[label] = pd.DataFrame(pca_train[:,pc_number])
        test[label] = pd.DataFrame(pca_test[:, pc_number])
        pc_number += 1
    # Find the variance explained within each PC
    var_exp = pca.explained_variance_ratio_
    # return train , test and var explained of the pca
    return train, test, var_exp

def update_performance(data_size,ntree, acc_score , information_ratio, run_time,
    train_date, test_date, performance_store):
    # Store the data as needed
    performance_store['data_size'].append(data_size)
    performance_store['ntree'].append(ntree)
    performance_store['Accuracy_Score'].append(acc_score)
    performance_store['Info_Ratio'].append(information_ratio)
    performance_store['run_time'].append(run_time)
    performance_store['train_date_st'].append(train_date)
    performance_store['test_date_st'].append(test_date)
    return pd.DataFrame(performance_store)


def set_params_random_forests():
    '''
    This is the control center for all the params that need to be set in the RF modules
    :return: return all params as they have been set here
    '''
    ############################ Set Model Paramaters ##############################
    param_dict = {"ntrees" : [150], "max_features" : 5, "test_buffer" : 5, "max_depth"
        : 30 , "data_size" : 15000 ,
                  "concat_results" : False, "test_split" : 0.25, "thold" : 0.51,
                      "window" : 1000, "trade_horizon" : 24,
                  "use_risk_adjusted" : False , "use_binary" : False, "use_classifier"
                      : False, "use_pca" : 0,
                  "use_separated_chunk" : False, "use_random_train_data" : True}
    # this looks back over a set period as the memory for the LSTM
      # [i for i in range(25,301,25)] # [21, 66]
```

```
    # if running pca, max features can only be same or less than the full total of
        features
    return param_dict


def set_params_LSTM():
    '''
    Additional params only applicable to the RF code
    return:
    '''
    lstm_dict = {'EPOCH' : 500, 'first_layer': 32, 'second_layer': 16, 'look_back' :90
        }
    return lstm_dict

def initialise_process(file_location, trade_horizon, window, use_risk_adjusted,
    use_pca,use_binary, use_random_train_data):
    '''
    This re freshes the whole data set as needed by the ipython process
    this is the function to modify if you want different features in the model.
    :return: data_normed df with standardised values and model features to use
    '''
    data_file = pd.read_csv(file_location) #
        pd.read_csv(r"/storage/eurusd_train_normed.csv")
    data_file = data_file.replace(np.nan, 0)
    ############################## Set Model Paramaters ##############################
    # this looks back over a set period as the memory for the LSTM
    model_features = ["spot_v_HF", "spot_v_MF", "spot_v_LF", "HF_ema_diff",
                      "MF_ema_diff", "LF_ema_diff", "target"] # "LDN", "NY", "Asia"
                          removed, # target must be kept at the end
    ################### Standardise Entire Dataset using rolling lookback windows
        ###############
    features_to_standardise = ["spot_v_HF", "spot_v_MF", "spot_v_LF", "HF_ema_diff",
                               "MF_ema_diff", "LF_ema_diff"]
    ###### Set Targets ##############
    data_file["target"] = calculate_target(data_file, trade_horizon, use_risk_adjusted)
    # Remove infinity from the values, division by 0
    data_file["target"] = data_file["target"].replace(np.inf, 0)
    data_file['target'] = data_file["target"].replace(-np.inf, 0)
    data_file['target'] = data_file["target"].replace(np.nan, 0)
    # make it binary
    if use_binary:
        data_file['target'] = data_file["target"].apply(np.sign)
    # roughly 3 yrs of data slightly less actually
    data_normed = standardise_data(data_file, model_features, features_to_standardise,
        window)
    # add extra features non standardised, check we are using random or non random data
    if not use_random_train_data:
        data_normed['Date'] = data_file['Date'].iloc[window:]
    data_normed['CCY'] = data_file['CCY'].iloc[window:]
    data_normed['logret'] = data_file['logret'].iloc[window:]
    if use_pca > 0:
        # if we are using pca features, then model features need only to be PC1 and
            PC2 etc plus the target
```

```
        model_features = ['PC%s' %i for i in range(1,use_pca+1)]
        model_features.append("target")
    return data_normed.reset_index(drop = True), model_features,
        features_to_standardise

def main():
    pass

if __name__ == "__main__":
    main()
```