University of Oxford, Exeter College

Machine Learning Methods for Financial Forecasting: Application to the S&P 500.

Babak Mahdavi Damghani
babak.mahdavidamghani@exeter.ox.ac.uk
candidate number:
48008

Supervisors:
Dr. Patrick McSharry: mcsharry@robots.ox.ac.uk
&
Dr. Vasile Palade: Vasile.Palade@comlab.ox.ac.uk

September 1st, 2006

Thesis submitted in partial fulfillment of the requirements for the degree of Master of Science in Computer Science.

Abstract

The world of quantitative finance is extremely intricate and the level of complexity has increased over the years. Therefore, Decision Support Systems (DSS) are more and more appreciated in the world of finance. In recent years, machine learning and statistical techniques have been applied to financial problems, and people in the industry have realized that these techniques are extremely powerful and, usually, are competitive with current systems. They are therefore ready to invest on research pertaining or related to Artificial Intelligence and on new mathematical models applied to options pricing, derivatives, currency trading, hedging, etc.
The main purpose of this project was to study the historical values of the S&P500, so as to forecast future values. First, this dissertation presents in relative depth what may influence the movement of the S&P500. Then, it takes a univariate approach and it looks at some potential forecasting techniques so as to have a benchmark as well as get a feel of both what could be the leading indicators of the S&P500 and what methods could be the most appropriate for a multivariate approach. The next step will be to implement the best model with the best set of indicators.
A set of different forecasting techniques were implemented and applied to the forecasting of the S&P500. The obtained results of this study were very encouraging. The final design performs better than the benchmark and outperforms Gately design, a reference in the industry.
It is concluded that implementing this final model could perhaps be a successful investment for at least a short period of time, until the arbitrage opportunity is absorbed, but the methods could be useful for a longer period of time.

Keywords: Random Walk, Neural Networks, Markov Chains, Probabilistic Modeling, Principal Component Analysis, Auto Regressive Integrated Moving Average, Genetic Algorithms, Intelligent Techniques, Quantitative Finance.

Contents

1 Introduction

1.1 Objectives

The purpose of this project is to use historical data in order to make predictions of future values for the S&P500, an unmanaged group of stocks often considered representative for the stock market in general. This index is composed of 400 industrial, 20 transportation, 40 utility, and 40 financial companies [25]. The project will be developed in several steps. First, we will make a literature review about financial forecasting and see what may influence the movement of the S&P500. Then, we will take a univariate approach and look at some potential forecasting techniques. This will allow us to get a benchmark as well as a feel of what methods could be the most appropriate for a multivariate approach. The final step will be to implement the best model with the best set of indicators. To summarize, we will study the following function:

$$f(L^{x_1}I1_{s:e}, L^{x_2}I2_{s:e}, \dots, L^{x_n}In_{s:e}) = \hat{P}_{t+1}$$

where:

- L is the lag operator such that: $L^{x_n}In_{s:t} = In_{s-x_n:t-x_n}$.
- Ii is one potential Leading Indicator.
- $\hat{P}_{t+1}$ is the forecasted value of the S&P500 at time t+1.
- f is a trained function using Neural Networks, Markov Chains, Genetic Algorithm, ARIMA or PCA methods.
- $x_n, s, e \in \mathbb{N}^*$ and $x_n < s < t$.

Note that it is very important that one partitions the available data so as to have different training and testing parts [1, 35]. The main reason is that many advanced techniques can perfectly model a function given a data set and a target. The design and the trained function are only considered efficient if they work for new independent data. Also, because the market is very dynamic, in order to keep the system updated as much as possible, we will dynamically train and forecast the data at each time step (Figure 1). To be more precise, we want to be able to test for as many data as possible without underestimating the training process. The market changes very rapidly, so that arbitrage opportunities should be quickly localized and investment bankers quickly take adequate measures, in order to make the most out of a concrete situation. Consequently, we not only want to have an initial training set to check the initial test data, but also the model has to be retrained every step so as to have the best current model for every new test (Figure 1).

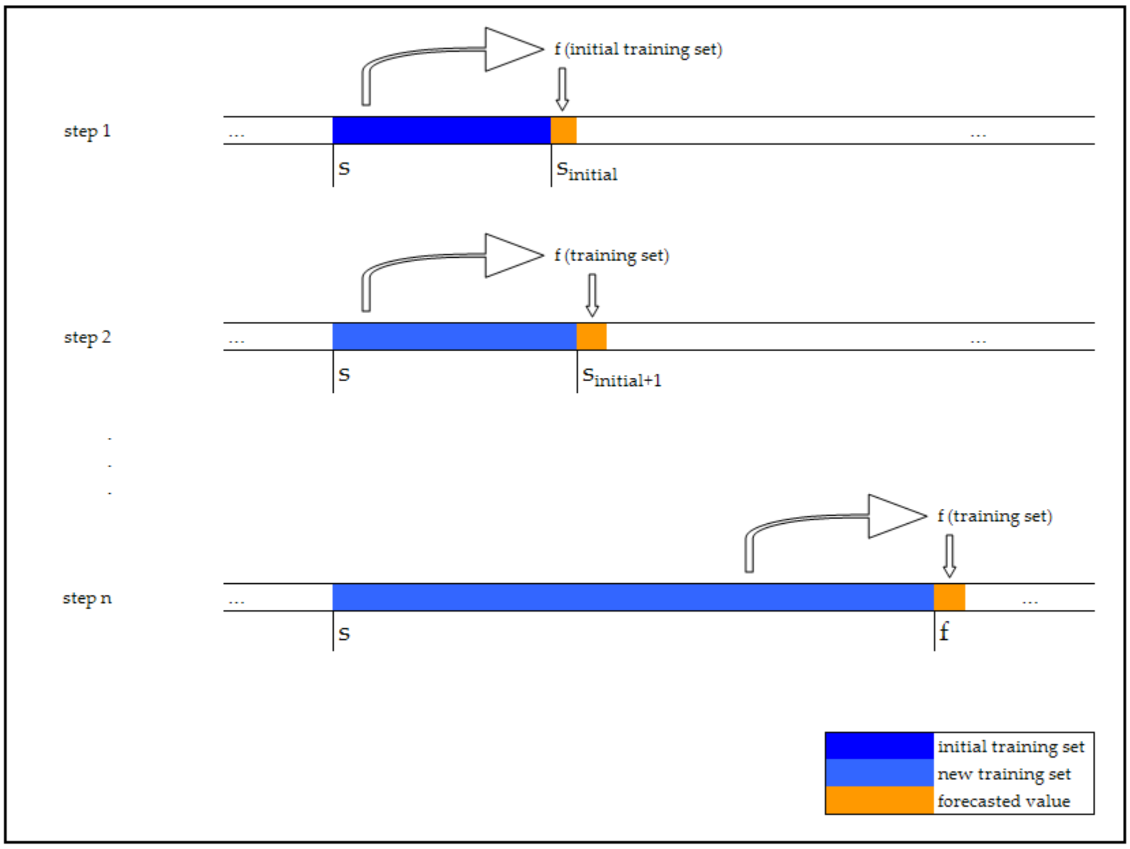


Figure 1: This Figure represents how the f function is trained and how the system will dynamically check for its performance.

1.2 Background Information on Forecasting the Market

In this dissertation, whenever we will mention market, we really mean the stock market and more precisely the stock market in the US symbolized by the S&P500. The US economy is a good indicator of the world economy in general, and the S&P500 is well known to be the index that is the most representative in the US. Therefore, using the S&P500 becomes an obvious choice because of several reasons. First, unlike equity (stock of companies, like IBM, Microsoft or Starbucks), the pricing of the stock is not determined by price to earning ratio or the company performance. In this case, the equity price fluctuates depending on the the actual performance with respect of the expected one. The S&P500 is different in the sense that its fluctuations are more determined by macroeconomic dynamics, and therefore making a profit out of it does not require ”internal” information like it would be the case for an employee who knows that the company's performance is better than expected and, unethically, buys stocks before the company releases that information to the public. The fluctuation of the S&P500 tries to replicate a portfolio of all the stocks in the market. The purpose of the project was more with respect to the involvement of machine learning techniques rather than to study in depth what impacts the value of the S&P500. So, we will not spend much time on the deep

reasons on why a certain indicator impacts the value of the S&P500, and we will take as given that the following indicators somehow impact, with a certain lag, the value of the S&P500:
GDP, S&P500 itself, Investors Sentiment, Expectation (COT), Average Directional indeX (ADX), Moving Average (MACD) , Volume traded, Money Supply (M1), Consumer Price Index (CPI), Wholesale Price Index (WPI), Tourist Arrival, Stock Price Index, Total Imports, Interest rate, Electric Energy consumption, Exchange rate, Average workweek (manufacturing), Unemployment, New orders for consumer goods, Building permits, Real M2, Bond Return (Yield), International Indices, US Indices, Inflation report, Calendar anomalies, Gold Futures Price, Reaction of Financial Markets to economic data, Central bank policy, The age wave (demographics), Transaction Costs, Democrats and Republicans (politics), S&P 500 Futures, Increasing favorable tax factors for equities, The Price-to-Earning Ratio, Volatility, Vendor performance, Plant and equipment orders, Wacker Drive at the Chicago Mercantile Exchange (WDCME), Change in unfilled durable orders, Sensitive material prices, Number of business incorporations and Bias within the Definition of the S&P 500 [15, 27].
Unfortunately, even given all this information, according to the researches done by Eric Troseth [32], most financial advisers say that it is almost impossible to forecast the stock market. According to the International Monetary Fund (IMF), out of 72 recessions worldwide in the 1990s, economists foresaw just eight in the year before they happened [7], which gives a success rate of 11%. Other economists, such as David Stamp, explain how forecasts are fundamentally flawed because they do not take into account "external events" such as the suicide attacks on the World Trade Centre or the collapse of the Berlin Wall [29]. However, Min Qi [24] mentions that it is commonly agreed among economists that business cycles are asymmetric and cannot be adequately accommodated by linear constant parameter single-index models. Neural Networks are a class of flexible nonlinear models. Given enough data, they can approximate any function arbitrarily close. Because there is little a priori knowledge about the true underlying function that relates financial, economic and composite indicators to the probability of future recessions, the NN models are an ideal choice for modeling these relationships [24].

1.3 Structure of the Dissertation

Giving few comments on the structure of the dissertation might help the reader understand the project better as well as jump from section to section without loosing any information that would be essential in understanding the big concepts. The first chapter introduces the topic and the objectives of the project. Since the project is a great deal of data manipulation, the second chapter aims at giving the details about the data used. We will complete our understanding of these leading indicators as we progress throughout this dissertation. The third chapter gives us a first exposure to some of the machine learning techniques available. This chapter helped choosing what machine learning technique would be optimal to a multivariate (more leading indicators, therefore more complex) approach in the fourth chapter. This fourth chapter tries three designs: a naive approach, an approach that has been successful in the past (Gately) and our own final design. The fifth chapter presents the techniques used to analyze the obtained results. The sixth chapter presents these results, and all the codes developed throughout the project are presented in the appendix. Note that at the end of every section or subsection, the codes involved in the relevant program

`will be mentioned with the following font style.`

2 The Data

2.1 What is an Economic Indicator?

Every time series system requires data in order to make a simple forecast. The data that we will use for forecasting is delimited into three main categories:

- Leading indicators lead the economy or lead what ever we are trying to forecast in our case the S&P 500.
- Lagging indicators happen "after". This will be irrelevant for us as we are trying to forecast, we will disregard that type of indicator.
- Coincident indicators happen "in the same time". Coincident indicators will be used in our case to correct for any perturbation in our model. For example if we use several leading indicator as a mean of forecasting, we will also input an indicator such as, news for example, in cases where external events might come and perturb the learning process for our forecast.

2.2 Literature Availability

After extensive research it is our general understanding that leading indicators for the fluctuation of the S&P 500 are hard to find. And this is mainly due for two reasons:
- Absurdity of the idea: Most of the scholars that have a good understanding of the stock market are often supporter of the Random Walk: "A random walk is one in which future steps or directions cannot be predicted on the basis of past actions. When the term is applied to the stock market, it means that short-run changes in stock prices cannot be predicted. Investment advisory services, earnings predictions, and complicated chart patterns are useless. On Wall Street, the term "random walk" is an obscenity. It is an epithet coined by the academic world and hurled insultingly at the professional soothsayers. Taken to its logical extreme, it means that a blindfolded monkey throwing darts at a newspaper's financial pages could select a portfolio that would do just as well as one carefully selected by the experts." [19]. Few are the ones who take their chance in writing a scholarly article that would be detailed enough for us to use. We will never find an article that would state that for example: "a three percent increase of the money supply impact negatively the S&P 500 five months from now by 8 percent". This is actually understandable because such a statement is for now impossible to prove or empirically verified because of the huge noise in the historical data.
- Market Efficiency [28] is the relation between stock prices and information available to investors indicating whether it is possible to beat the market; if a market is efficient, it is

not possible, except by luck. The second reason for this lack of information comes from the fact that the only people that believe in the predictability of the stock market and that have done research on this field and claim that they have had possible decent prediction will not reveal there full work because they are afraid of market efficiency. To emphasis, if there full work were to be published (with the leading indicators and the exact technique to get possible results) other people would use that "arbitrage" information to buy futures on the S&P 500 which would ultimately cancel out the work done by the ANN and we would be back to the Random Walk.
We have been however able to find enough literature to pursue our research but will ultimately have to "shift" our data from left to right so has to capture the most updated lag between the S&P500 and its leading indicators.

2.3 What are the Potential Leading Economic Indicators?

Since the purpose of this dissertation is more the general methods of forecasting rather than the application on the S&P500 itself, we will not spend too much time on this section and will refer to books on the topic such as Stocks For the Long Run by Jeremy Siegel [27] that is quite well written and give a nice overview of the topic as well as other people such as Gately who have done research on forecasting [15]. We will therefore take as given that the following indicators "somehow" lead the stock market:
GDP, Investors Sentiment, Expectation (COT) and related data, Average Directional indeX (ADX), Moving Average (MACD), Volume, Money Supply (M1), Consumer Price Index (CPI), Wholesale Price Index (WPI), Tourist Arrival, Stock Price Index, Total Imports, Interest rate, Electric Energy consumption, Exchange rate, Average workweek (manufacturing), Unemployment, New orders for consumer goods, Building permits, Real M2, Bond Return (Yield) , International Indices , US Indices, Inflation report, Calendar anomalies, Gold Futures Price and finally Astrology & Finance. Please contact us directly via email if you need more details about this area.
For our univarite approach we will use the value of the S&P500 itself with a one day shift and for our multivariate approach we will use the following indicators with variable shift:
- Net Daily Adjusted commitments of large speculators for several indicators in our pool of economical indicators.
- Net Daily Adjusted commitments of large hedgers for several indicators in our pool of economical indicators.
- Net Daily Adjusted commitments of small traders for several indicators in our pool of economical indicators.
- Net Daily for several indicators in our pool of economical indicators.
Our pool of indicators will be composed of: Dow Jones Transportation Index, Dow Jones

Industrial Index, Dow Jones Utilities Index, AMEX Oil Index, Gold and Silver Mining Index, Open S&P 500, High S&P 500, Low S&P 500, Volume S&P 500, DJI moving average 30 days and News. Otherwise, the data for the S&P 500 as well as the other indicators was taken between 13th of April 2006 and 7th of October 1997 downloaded from yahoo finance [14] or purchased from Pinnacle Data Corp. [2].

2.4 What should the Length of the Data be?

Another important factor of our design is the length of the data used. This is actually a key element. Here my concern was that the stock market changes its behavior very often, so that our sample space should not be too big and use data that was irrelevant for our training; but in the same time the data sample should not be too little so that we avoid computation problems. Edgar E. Peters supports this idea that the market changes rapidly and data from longer period in the past is irrelevant [22] and according to Dr. Zandi, University of Pennsylvania: "in order to get viable results with neural networks we will need approximately 15 times as many rows than we have columns [35]. To emphasise, if we were to use two leading indicators in our design for example: Consumers Expectation and Inflation, then we are going to need as many as 30 entry rows in our design. But in order to cope with this problem of having enough data to work with, but in the same time putting less weight on the irrelevant data we can use a forgetting factor. Several techniques already exist and we will use the exponential decay, logarithmic or weighted average techniques depending on the method, in order to create this forgetting factor. We will also see how we will reduce the noise in our data for each of the different methods.

2.5 What should the Lag of the Leading Indicators be?

Gately [15] explains in his book that the optimal lag between S&P and its leading indicators is ten days. Therefore our initial design will be a 10 days lag between the S&P 500 and its indicators. Later if we have time we will write an algorithm that will shift our data with few days up and down to find an optimal design. We will study more in depth this part when we study the time complexity Issue for a multivariate approach.

2.6 How are we going to modify the data?

We will never emphasise enough how important the input to our function is, because, once again, no matter how complex and elegant our system is, we will not learn the right function and therefore will be doomed in having incorrect results if our input is wrong. Therefore, now that we have the right list of indicators with the right shift we want to do several things:

- change it so that we have the return.
- cut the data or create a forgetting factor.
- reduce the noise.
- normalize the data.

• First then, we will modify the historical value of our indexes so as to get the daily return rather than the value of the index on that day in particular that is:

$$\Lambda P_t = \frac{P_t}{P_t + P_{t-1}}$$

where P_t is the value of the S&P500 at time t.
Note that one could instead define ΛP_t to be equal to $P_t - P_{t-1}$ instead but in this case one needs to normalize the data as well as we will see in the last bullet point.

• As for creating a forgetting factor several methods are known and we will apply different methods to different approaches depending on our needs. As we have explained earlier a forgetting factor is also important because the stock market changes its behavior very often, so that our sample space should not be too big and use data that was irrelevant for our training [22]. Few Investment banks claim that data from more than five years ago is irrelevant but it is very difficult pin point this fact using data where the noise is sometimes higher than the signal itself. What we do know though is that the stock market evolves therefore cutting or creating a forgetting factor for our data is primordial. One way is to put a weight to our data so that earlier data have a smaller importance compared to newer ones. For example:

$$Weight_{V_t^i} = log(t)$$

or

$$Weight_{V_t^i} = t$$

or

$$Weight_{V_t^i} = e^t$$

or any other type of continuous function that is strictly increasing as t increases where the higher the t, the most recent is the data, we have f is a continuous function and $f_{Weight}(t) < f_{Weight}(t+1)$.
Note that we have seen in the introduction that we will dynamically retrain all of our available data preceding whatever we are trying to forecast so as to have the most updated system possible. There are here two or perhaps three methods, with the last one being a combination of both, able to model this model. The first one is simply to feed half of the past data into the training process along with a forgetting factor. The second one is to

feed some of the past data (the most recent) and simply disregard the older one without a forgetting factor. You can visualize this idea in the figure 2 below. Finally the last possibility is a combination of these two ideas that is to feed some of the past data (the most recent) and simply disregard the older one with a forgetting factor.

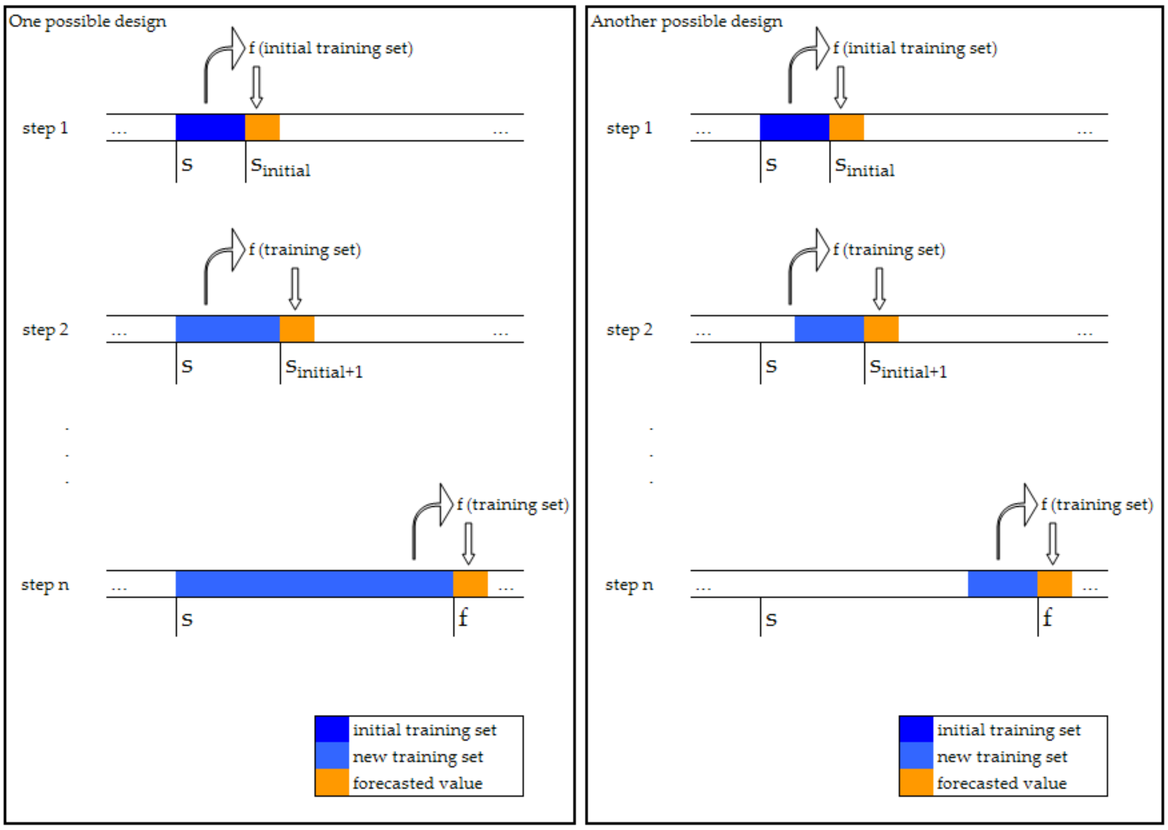


Figure 2: This Figure represents the two main training methods available where training is done dynamically so that the newest data is always in the training set.

• Also Chartfield (1993) [8] and Christoffersen (1997) [23] rightfully mentioned that in financial time series the noise is sometimes bigger than the signal itself which suggests that one should first correct the data before proceeding into training, and assigning probability forecast [20, 9] along with density forecast.

First, then, in order to remove noise, the exponential moving average suggested by Andreas S. Weigend and Shanming Shi (2000) [33] was chosen in order to modify our data:

$$\xi_t = \lambda \xi_{t-1} + (1 - \lambda) P_t$$

where:

P_t is the value of the S&P 500 at time t.

λ is a variable such that $0 \leq \lambda \leq 1$, here set to 0.95

$\xi_0 = 0$ as for the initial criteria for our recursive design.

• Finally, the data is normalized especially for our multivariate designs so that the unit value of each indicator does not outweigh the importance of other indicators that may have a smaller unit value but whose real importance is primordial. The data is normalized so that it falls between -1 and +1. Besides, the interval [-1 1] is convenient. We will see why in details, in the Neural Network subsection, but the main idea is that the Tan-Sigmoid function, the activation function we use, ranges between -1 and 1. One naive way of thinking of normalization is that the highest value of your data set takes the value of 1 and your smallest data, the value of -1 and everything else falls in conveniently in between. The relevant program can be found in the appendix. The name of one the functions mentionned here is:

`forgettingFactor`.

3 Univariate Methods

The aim of this section is to see whether one can predict the fluctuations of the S&P 500 using the S&P 500 itself. We would like to create a model that would first be trained to capture what makes the values of the S&P 500 at time t, fluctuate, use it aquired knowledge in order to make future predictions and assign a confidence criteria to its guess via a density function.

3.1 Random Walk

Many people believe that the movement of the stock market is random. It can be formalized in different ways. One simple way is perhaps to describe it as a martingale (a discrete-time stochastic process: a sequence of random variables X_1, X_2, X_3, ...) that satisfies the identity:

$$E(X_{t+1}|X_t, X_{t-1}, \ldots, X_2, X_1) = X_t$$

We will therefore create a first simple design that we will call random walk and that will satisfy:

$$\hat{X}_{t+1} = X_t$$

- where $\hat{X}_{t+1}$ is the forecast for our model for time $t + 1$
- and X_t being the actual real return at time t

The relevant program can be found in the appendix. The name of the functions are:

`randomWalkcut` and
`randomWalkRol`.

3.2 Autoregressive Integrated Moving Average: ARIMA

Many people in the literature have modeled time series using mathematical models such as ARIMA which has been pointed out by Taylor, de Menezes and McSharry 2005 [31], to be a good benchmark in short-term forecasting methods. The idea is actually not new, Laing and Smith, 1987 [18]; Darbellay and Slama, 2000 [10]; Abraham and Nath, 2001 [6]; Taylor, 2003 [31], all had used this method as a benchmark.
Our purpose here would be to generate a code that would take as input the fluctuations of the S&P 500 (up, down) no more than 2 days ahead so as to compare it with other methods such as our second order Markov Chain model that we will see next, train an optimal ARMA model for the data we have inputted up to time t-1, compare the output

value from the model to the actual real value at time t, retrain the model every step so as to have the most updated model for each run and repeat the process, save the results as we go along and analyze the results at the end to use the efficiency of ARMA as a benchmark for our other methods.
The ARIMA(p,d,q) model is given by:

$$\phi(1-L)^d X_t = \theta \epsilon_t$$

where L is the lag operator such that:

$$L^k X_t = X_{t-k}$$

and,

$$\phi = 1 - \sum_{i=1}^{p} \phi_i L^i$$

$$\theta = 1 + \sum_{i=1}^{q} \theta_i L^i$$

The AR coefficients can be solved using the Yule Walker Equations or the following method.
If the lag is 1 then we could write the ARIMA(1,0,0) the following way:

$$X_t = \phi_1 X_{t-1} + \epsilon_t$$

Using, matrix representation we get:

$$\phi_1 \underbrace{\begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_{N-1} \end{pmatrix}}_{A_t} = \underbrace{\begin{pmatrix} x_2 \\ x_3 \\ \vdots \\ x_N \end{pmatrix}}_{A_{t+1}}$$

The estimator for ϕ_1 can be found by using the least-squares estimator that is:

$$A_{t+1}^T A_{t+1} \hat{\phi}_1 = A_{t+1}^T A_t$$

$$\hat{\phi}_1 = \left(A_{t+1}^T A_{t+1}\right)^{-1} A_{t+1}^T A_t = \frac{\sum_{i=1}^{N-1} x_i x_{i+1}}{\sum_{i=1}^{N-1} x_i^2} = \frac{c_1}{c_0} = r_1$$

with c_1 and r_1 being, respectively, the first autocovariance and autocorrelation coefficients.
Note that even though this method seems rather straight forward for a lag equal to 1, this method becomes increasingly complex as we add lags. Therefore, since we want a $\hat{\phi}_i$ for

all of our rows (more than a 1000 in the initial training part and increasing to more than 2000), we want a method that would be less complicated for higher indexes. A better method would perhaps be the Yule Walker method that can be generalized to any lags k, that is for AR(p):

$$x_{i+1} = \phi_1 x_i + \phi_2 x_{i-1} + \cdots + \phi_p x_{i-p+1} + \theta_{i+1}.$$

Multiplying by x_{i-k-1}, we get

$$x_{i-k-1}x_{i+1} = \sum_{j=1}^{p} \left(\phi_j x_{i-k+1} x_{i-j+1}\right) + x_{i-k+1}\epsilon_{i+1}$$

taking the expectations on both sides, we have:

$$\langle x_{i-k-1}x_{i+1}\rangle = \sum_{j=1}^{p} \phi_j \langle x_{i-k+1}x_{i-j+1}\rangle$$

note that $\langle x_{i-k+1}\epsilon_{i+1}\rangle$ is simply 0 because the errors are uniformly distributed with $\mu = 0$. Now, dividing through by (N - 1), and using the fact that $c_{-l} = c_l$, we get:

$$c_k = \sum_{j=1}^{p} \phi_j c_{j-k}$$

dividing through by c_0,

$$r_k = \sum_{j=1}^{p} \phi_j r_{j-k}$$

Therefore we have:
$r_1 = \phi_1 r_0 + \phi_2 r_1 + \cdots + \phi_p r_{p-1}$
$r_2 = \phi_1 r_1 + \phi_2 r_2 + \cdots + \phi_p r_{p-2}$
$\vdots \ddots$
$r_p = \phi_1 r_{p-1} + \phi_2 r_{p-2} + \cdots + \phi_p r_0$
r_0 being set to 1, this can be rewritten as:

$$\underbrace{\begin{pmatrix} r_1 \\ r_2 \\ \vdots \\ r_{p-1} \\ r_p \end{pmatrix}}_{r} = \underbrace{\begin{pmatrix} 1 & r_1 & r_2 & \cdots & r_{p-1} \\ r_1 & 1 & r_1 & \cdots & r_{p-2} \\ r_2 & r_1 & 1 & \cdots & r_{p-3} \\ \vdots \ddots & & & & \\ r_{p-1} & r_{p-2} & r_{p-3} & \cdots & 1 \end{pmatrix}}_{R} \underbrace{\begin{pmatrix} \phi_1 \\ \phi_2 \\ \vdots \\ \phi_{p-1} \\ \phi_p \end{pmatrix}}_{\Phi}$$

Therefore, we have:

$$r = R\Phi$$

$$R^{-1}r = R^{-1}R\Phi \Rightarrow \Phi = R^{-1}r$$

Note that Matlab provides an already predefined algorithm that finds the necessary coefficients of AR and MA, but one requirement is for our autocovariance functions to be stationary and invertible.
The time series $\{X_t, t \in \mathbb{Z}\}$ (where $\mathbb{Z}$ is the integer set) is said to be stationary if:
$\forall t \in \mathbb{Z} \Rightarrow E\left(X_t^2\right) < \infty$
$\forall t \in \mathbb{Z} \Rightarrow E(X_t) = \mu$
$\forall a, b, t \in \mathbb{Z} \Rightarrow Cov(X_a, X_b) = Cov(X_{a+t}, X_{b+t})$
Similarly, we say an ARMA model is invertible if all the zeros of its MA polynomial lie outside the unit circle.
Therefore in changing our p and q from our ARIMA(p,d,q) model, we will have to be careful in how we increment or decrement our parameters so as to respect the stationarity and invertibility criterion.
The relevant program can be found in the appendix. The name of the functions are

```
ARIMAtestCutting and
ARIMAtestRolling.
```

3.3 Probabilistic Modeling: Second-Order Markov Chains

The aim of this section is to see whether one can predict the fluctuations of the S&P 500 using the S&P 500 itself. The idea is that the market either goes up, down or stays pretty much constant. We would like to create a model that would first be able to categories the value of the S&P 500 at time t, V_t^i, in each of these three following different groups, $V^{up}, V^{equal}, V^{down}$, "learn" what did put them in these groups and would then use its "acquired knowledge" in order to make future predictions and assign a confidence criteria to its guess via a density function.
The hypothesis is that the fluctuation of our index is only determined by what happened in the two previous steps:

$$V_t^i = f(V_{t-1}^j, V_{t-2}^k)$$

Where i, j and k are in the set of possible outputs: O = {up, equal, down}and with V representing an object with an internal memory, "path" (eg: up-down / up-up/ equal-down etc...) and its relation to an output O. Since we are using a second order Markov Chain, the number of paths will be 8 (2^3) and the number of outputs 3. The probability of our value at time t is determined by the probability of V^i at t-1 and t-2:

$$P(V_t^i) = P(V_{t-1}^j | V_{t-2}^k)$$

And O is defined such that:

$$O_t^{up} \Leftrightarrow \Delta\xi_t > c_t^+ . \mu_{|\Delta\xi_t|}$$

$$O_t^{down} \Leftrightarrow \Delta\xi_t < -c_t^-.\mu_{|\Delta\xi_t|}$$

$$O_t^{equal} \Leftrightarrow c_t^+.\mu_{|\Delta\xi_t|} \geq \Delta\xi_t \geq -c_t^-.\mu_{|\Delta\xi_t|}$$

With:
- c_t^+ and c_t^- being constants updated at each run so as too separate evenly our set O at each time step. c_t^+ and c_t^- are set to the dummy variables 0.485 and 0.530 respectively for the first initial 1000 rows of training set and updated so as to incorporate the tested data at each new run.
- $\Delta\xi_t = (\xi_t - \xi_{t-1})/\xi_{t-1}$.
- $\mu_{|\Delta\xi_t|}$ being the average positive change at time t for ξ.

It is believed that the stock market evolves very quickly and that in using time series as an analytical tool, one should put an emphasis on this rapid change by incorporating a forgetting factor so that more recent changes should have higher weights in our forecast compared to less recent ones. The forgetting factor for our analysis is logarithmic so that the weight of our object V at time t is log(t). This choice was made for two reasons. First, the way our recursive call was designed, one needed to have the first couple of rows neglected (log(1) = 0 and log(2) = 0.7) and second, since we have a second order Markov Chain, therefore 8 different paths with each 3 possible outputs for an initial training set of 1000, one could argue that the average dataset per "box" would be around 41 which is close to 30, a number many statistician consider minimum in order to have a sample space that can mean something and be analyzed. Therefore a first order Markov Chain would have been too simple and a third order Markov Chain would not have allowed enough data per "box" to have anything meaningful. Otherwise, a classic weighted average or an exponential decay would have put way to much emphasis on the new data and potentially bias the training design. A logarithmic model is perfect in this situation because:
- log(t) moves rapidly towards 0 as t approaches 1.
- log(t) moves more and more slowly towards infinity as t approaches infinity.
Therefore:

$$Weight_{V_t^i} = log(t)$$

The forecasted return $\hat{\xi_t}$ is given by:

$$\hat{\xi_t} = \varpi(\xi_{t-1}, \xi_{t-2})$$

- where $\varpi(a, b)$ returns the expected movement, Mv, from the probability matrix given the path from (a, b)
- if Mv = "up", then $\hat{\xi_t} = \overline{\xi_{1:t-1}} + std(\xi_{1:t-1}) + \epsilon \times std(\xi_{1:t-1})$.
- if Mv = "equal", then $\hat{\xi_t} = \overline{\xi_{1:t-1}} + \epsilon \times std(\xi_{1:t-1})$.

- if $Mv =$ "down", then $\hat{\xi}_t = \overline{\xi_{1:t-1}} - std(\xi_{1:t-1}) + \epsilon \times std(\xi_{1:t-1})$.
- with ϵ being an error normally distributed with mean 0 and standard deviation 1.

The relevant program can be found in the appendix. The name of the functions are:

```
trainMarkovWithWeight,
testMarkovWithWeightcut and
testMarkovWithWeightrol.
```

3.4 Neural Networks

The aim of this section is to see whether one can predict the fluctuations of the S&P 500 using the S&P 500 itself. The idea is that the market either goes up, down or stays pretty much constant. We would like to create a model that would first be able to categories the value of the S&P 500 at time t, V_t^i, in each of these three following different groups, $V^{up}, V^{equal}, V^{down}$, "learn" what did put them in these groups and would then use its "acquired knowledge" in order to make future predictions and assign a confidence criteria to its guess via a density function.
Since our design is univariate, a simple neural network should be able to pick a pattern up if it exists; therefore we have chosen a two layer network with three perceptrons in the first layer and one perceptron in the hidden layer. A Tan-Sigmoid activation function is chosen for the first layer and a linear transfer function for the hidden layer. The learning rate is set to 0.05, the momentum to 0.9 and the number of epochs to 300. Note that the Matlab help section has a good explanation on the use of momentum [3]:
"Momentum allows a network to respond not only to the local gradient, but also to recent trends in the error surface. Acting like a low-pass filter, momentum allows the network to ignore small features in the error surface. Without momentum a network may get stuck in a shallow local minimum. With momentum a network can slide through such a minimum. [...] The magnitude of the effect that the last weight change is allowed to have is mediated by a momentum constant, mc, which can be any number between 0 and 1. When the momentum constant is 0, a weight change is based solely on the gradient. When the momentum constant is 1, the new weight change is set to equal the last weight change and the gradient is simply ignored. The gradient is computed by summing the gradients calculated at each training example, and the weights and biases are only updated after all training examples have been presented." The gradient descent algorithm for our design is derived the following way:

$$f(x_t, v, w) = g_2\left(\sum_{h=1}^{1} v_h \; g_1\left(\sum_{i=1}^{3} w_{hi} x_{it}\right)\right)$$

where $g_1(x) = \frac{1}{1+e^{-x}}$ and $g_2(x) = x$ are the activation function, v_h and w_{hi}, the weights for the hidden and the input layer.

$$E = \frac{1}{2}Err^2 \equiv \frac{1}{2}(y_{real} - f(x_t, v, w))^2$$

where $f(x_t, v, w)$ is the output for our design. We can use gradient descent to reduce the squared error by calculating the partial derivative of E with respect to each weight .

$$\frac{\partial E}{\partial v_h} = Err \times \frac{\partial Err}{\partial v_h}$$

$$\left(\frac{\partial E}{\partial v_h}\right)_t = -Err \times g_1\left(\sum_{i=1}^{3} w_{hi}x_{it}\right) \times g_2'\left(\sum_{h=1}^{1} v_h\ g_1\left(\sum_{i=1}^{3} w_{hi}x_{it}\right)\right)$$

$$\left(\frac{\partial E}{\partial v_h}\right)_t = \frac{-Err}{1+\exp-\left(\sum_{i=1}^{3} w_{hi}x_{it}\right)} \times \frac{v_1}{1+\exp-\left(\sum_{i=1}^{3} w_{hi}x_{it}\right)} = \frac{-Err \times v_1}{1+\exp-\left(\sum_{i=1}^{3} w_{hi}x_{it}\right)}$$

Therefore at each run v_h is updated the following way:

$$v_h \leftarrow v_h + 0.05 \times a_h \left[0.1\left(\frac{\partial E}{\partial v_h}\right)_t + 0.9\left(\frac{\partial E}{\partial v_h}\right)_{t-1}\right]$$

where $a_h = g_1\left(\sum_{i=1}^{3} w_{hi}x_{it}\right)$

$$\frac{\partial E}{\partial w_{hi}} = Err \times \frac{\partial Err}{\partial w_{hi}}$$

$$\left(\frac{\partial E}{\partial w_{hi}}\right)_t = -Err \times \sum_{h=1}^{1} v_h g_1'\left(\sum_{i=1}^{3} w_{hi}x_{it}\right) \times g_2'\left(\sum_{h=1}^{1} v_h\ g_1\left(\sum_{i=1}^{3} w_{hi}x_{it}\right)\right)$$

from here since we know that:
$g_1'(x) = \frac{\partial sig(x)}{\partial(x)} = sig(x)(1 - sig(x))$ where $g_1(x) = sig(x) = \frac{1}{1+e^{-x}}$
$g_2'(x) = 1$ where $g_2(x) = x$
we get:

$$\begin{aligned}\left(\frac{\partial E}{\partial w_{hi}}\right)_t &= -Err \times v_1 \frac{1}{1+exp\left(-\sum_{i=1}^{3} w_{hi}x_{it}\right)} \\ &\quad \times \left(1 - \frac{1}{1+exp\left(-\sum_{i=1}^{3} w_{hi}x_{it}\right)}\right) \times 1 \\ &= \frac{-Err \times v_1 \times \left[1+exp\left(-\sum_{i=1}^{3} w_{hi}x_{it}\right) - 1\right]}{1+exp\left(-\sum_{i=1}^{3} w_{hi}x_{it}\right)} \\ &= \frac{-Err \times v_1 \times exp\left(-\sum_{i=1}^{3} w_{hi}x_{it}\right)}{1+exp\left(-\sum_{i=1}^{3} w_{hi}x_{it}\right)}\end{aligned}$$

Therefore:

$$w_{hi} \leftarrow w_{hi} + 0.05 \times x_{it} \left[0.1\left(\frac{\partial E}{\partial w_{hi}}\right)_t + 0.9\left(\frac{\partial E}{\partial w_{hi}}\right)_{t-1}\right]$$

The algorithm consists of running a loop of 300 runs or until a stopping criterion is reached, in our case a $|\Delta Err < 10^{-3}|$, while updating the weights at each run.
The relevant program can be found in the appendix. The name of the functions are:

`Train_with_simple_FF_2Inputs`,
`test_Train_with_simple_FF_2InputsCut` and
`test_Train_with_simple_FF_2InputsRol`.

3.5 Principal Components Analysis: PCA

The aim of this section is to see whether one can predict the fluctuations of the S&P 500 using the S&P 500 itself but with a different lag (here one). The idea is that the market either goes up, down or stays pretty much constant. We would like to create a model that would first be able to categories the value of the S&P 500 at time t, V_t^i, in each of these three following different groups, $V^{up}, V^{equal}, V^{down}$, "learn" what did put them in these groups and would then use its "acquired knowledge" in order to make future predictions and assign a confidence criteria to its guess via a density function.
As it is normally the case for the PCA method we will first subtract the mean from our data set such that:

$$P_t^{adj.} = P_t - \bar{P}$$

where $P_t^{adj.}$ is the adjusted value for the S&P 500 at time t, P_t is the real value of the S&P 500 at time t and $\bar{P}$ is the mean for P_t. To simplify the notation we will use P_t instead of $P_t^{adj.}$ from now on, unless specified otherwise, this, in order to make the future notations easier. In the next step we have to calculate the covariance of our matrix such that:

$$cov(P_t, P_{t-l}) = \frac{\sum_{i=1}^{n}(P_t - \bar{P})(P_{t-l} - \bar{P}_l)}{(n-1)}$$

where $cov(P_t, P_{t-l})$ is the covariance between the S&P 500 at time t and S&P 500 at time $t-l$ with l being the lag. Also $\bar{P}_l$ is the mean value for the data for the S&P 500 l days before t. Also note that the divisor is $n-1$ and not n, this because we are using only a subset of the S&P 500 historic values.
The next step is to find the covariance matrix that is:

$$\mathrm{C} = \begin{pmatrix} cov(P_t, P_t) & cov(P_t, P_{t-l}) \\ cov(P_{t-1}, P_t) & cov(P_{t-1}, P_{t-l}) \end{pmatrix}$$

We then got the unit covariance vector:

$$C_{unit} = \begin{pmatrix} \frac{cov(P_t,P_t)}{length} & \frac{cov(P_t,P_{t-l})}{length} \\ \frac{cov(P_{t-1},P_t)}{length} & \frac{cov(P_{t-1},P_{t-l})}{length} \end{pmatrix}$$

$$\text{with } length = \sqrt{cov(P_t, P_t)^2 + cov(P_t, P_{t-1})^2}$$

Note that since $cov(P_t, P_{t-1}) = cov(P_{t-1}, P_t)$ and that $cov(P_t, P_t) = cov(P_{t-1}, P_{t-1})$, we do not need to define two different lengths for our covariance matrix. In the next step we will try to find the eigenvalues and eigenvectors for our covariance matrix:

$$C_{unit}x = \lambda x$$

where λ is a scalar and x is a vector.
Once all the eigenvalues are retrieved, we will have to select the highest eigenvalue λ_{max} amongst our set $\Lambda = [\lambda_1, \lambda_2, \lambda_3, \lambda_4]$, the eigenvector with the highest eigenvalue is the principle component of our data set. From here we create a FeatureVector whose purpose is to rearrange the different eigenvectors in decreasing order of eigenvalues, this gives us the components in order of significance.
The final stage is to derive an estimate for our forecast. This is done by multiplying the slope for the FeatureVector by the value of the S&P 500 at time $t-1$. The data is trained at each run so as to have the most updated model and so as to record the error at each run.
The relevant programs can be found in the appendix. The names of the functions are :

```
PAC,
testPCAcutting and
testPCArolling.
```

3.6 Genetic Algorithms

The aim of this section is to see whether one can predict the fluctuations of the S&P 500 using the S&P 500 itself. The idea is that the movement of the market can be split in 8 different intervals ranging from superior than a big positive jump to inferior than a big negative jump. We would like to create a model that would first be able to categories the value of the S&P 500 at time t, V_t^i, in each of these eight following different groups, V^{up^+}, $V^{up^=}$, V^{up^-}, V^{equal^+}, V^{equal^-}, V^{down^+}, $V^{down^=}$, V^{down^-}, "learn" what did put them in these groups and would then use its "acquired knowledge" in order to make future predictions and assign a confidence criteria to its guess via a density function. We will use a genetic algorithm approach for this section.
Many of our most advanced technologies are the fruit of our perception of the nature surrounding us. For example, the invention of planes was inspired looking at birds and their different techniques to fly and glide. This model we have developed here is inspired by Darwin's Theory of Evolution and Natural Selection [11]. Just as a refresher, Darwin's Theory of Evolution is based on the idea that the different species on earth are all the

offspring of one simple organism and these different species are the fruit of the combination of random mutation and natural selection [11]. Meaning that occasionally a random genetic mutation occurs in some specie and if that genetic mutation helps for survival (meaning: the environment surrounding it is appropriate for its survival) then it survives and has the possibility to "pass on" its genes to its offspring and create a new specie. This specie can later grow or disappear as the environment around it changes.
Therefore, the first step in our model is to define a fitting function that will dictate the way our population will evolve through time as well as inputs to this fitting function:

$$g_j\left(\sum_{i=1}^{n} w_i s_i\right)_{L^k I i_t} = g_j\left(\hat{\xi}_t\right)_{T_t}$$

where:

- g_j is a function that maps historical values to an interval divided in j different, equal in length sub-intervals between $[max(IHD), min(IHD)]$ where IHD is the input of historical data to the function g_j. The g_j function splits the interval and translates the values into binary for the relevant interval. For example a return of $+1.03\%$ and -1.05% on historical values of the S&P 500 on our time scale will translate to 1101 and 0010 respectively if $j = 16$. n represents the number of leading indicators we will use.

- $s_i = \{-1, 1\}$ represents whether there is a positive or negative correlation between the leading indicator and the value of the S&P500.

- $w_i = \{3, 2, 1, 0\}$ represents how strong the correlation is between the leading indicator and the value of the S&P500.

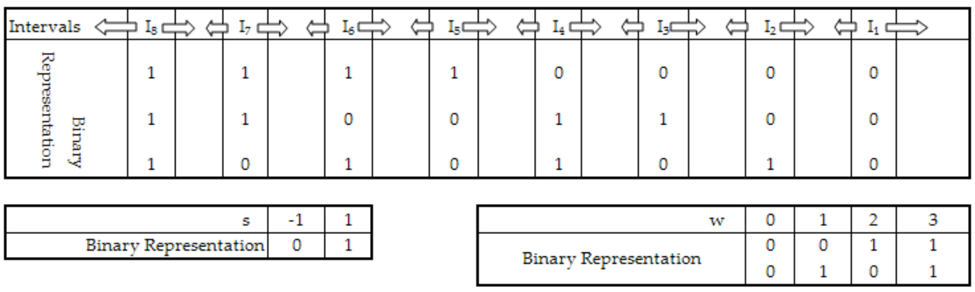
- L is the lag operator such that $L^k X_t = X_{t-k}$

- $\sum_{i=1}^{n}$ here means append rather than sum.

A univariate model where $j = 8$ could be modeled this way:

$$g_8\left(ws\right)_{L^k I_t} = g_8\left(\hat{\xi}_t\right)$$

We will therefore have (Figure 3):



Intervals	I8	I7	I6	I5	I4	I3	I2	I1
Binary Representation	1	1	1	1	0	0	0	0
	1	1	0	0	1	1	0	0
	1	0	1	0	1	0	1	0

s	-1	1
Binary Representation	0	1

w	0	1	2	3
Binary Representation	0	0	1	1
	0	1	0	1

Figure 3: This Figure represents how the g_8 function codes the different values of s and w into binary for each interval.

$$\begin{pmatrix} g_8\left(s_1 w_1\right) & g_8\left(\hat{\xi}_t\right)_1 \\ g_8\left(s_2 w_2\right) & g_8\left(\hat{\xi}_t\right)_2 \\ \vdots & \vdots \\ g_8\left(s_n w_n\right) & g_8\left(\hat{\xi}_t\right)_n \\ \vdots & \vdots \\ g_8\left(s_8 w_8\right) & g_8\left(\hat{\xi}_t\right)_8 \end{pmatrix}$$

with $g_8\left(\hat{\xi}_t\right)_i$ being any binary number of three digits representing the forecasted value of our index. Note that our model is total but not necessarily onto since we might not be able to forecast the value of our index in certain intervals.

Also, since the returns are normally distributed, the intervals, in order to be equal in number of components inside them, have to be different in lengths. The interval is defined as being $c_i \mu_{\xi_t} < Interval_i < c_{i+1}\mu_{\xi_t}$ for $2 < i < 7$ else $c_8 \mu_{\xi_t} < Interval_8$ and $c_1 \mu_{\xi_t} < Interval_1$.

In our genetic algorithm we will consider j equations representing the fitting functions (different species) that are independent from one another. The input to our system determines which equation we will look at. For a univariate approach, we only have to look at one leading indicator which makes the task easier. We will model our reduced DNA with the binary code for w first, then s and finally the input to our model or $L^k I_t$ which determines the "species". Since $w_i = \{3, 2, 1, 0\}$ and $s_i = \{-1, 1\}$, we will have 8 different possible solutions for our 8 different species. The genetic algorithm we performed works the following way. We initially have an equal proportion of the eight potential solutions. The idea is that depending on what input we get and therefore its position in our split, this value will have a higher likelihood than others to rich the interval x at time $t+1$. Each training period has several steps, but before the training section starts we initialized the eight possible populations that are defined as being the eight potential different inputs. Depending on what input we get, we will modify our initial population by training it thanks to our corresponding output (Figure 4). Therefore, within these eight functions, we create a population of 25 individuals for each potential possible output, so we will have 200 individuals for each of the eight functions.

Therefore, the input to our system will determine which population will be trained for

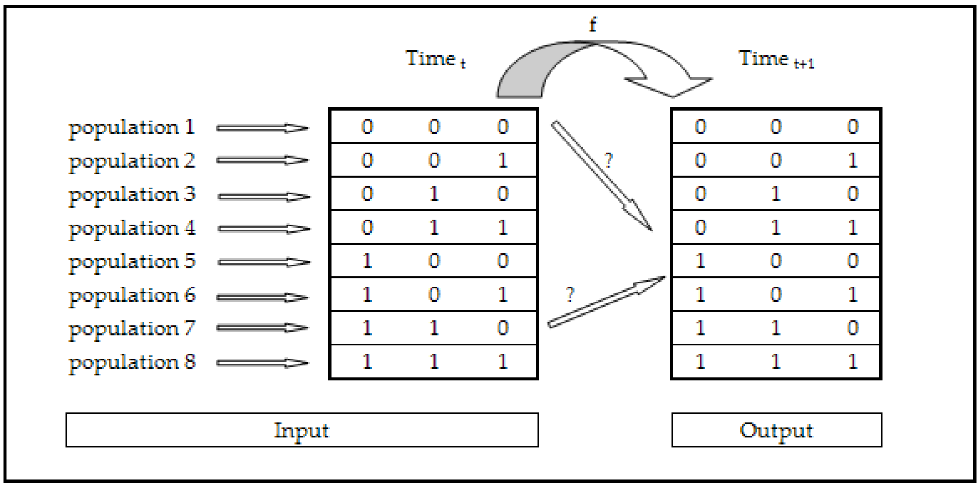


Figure 4: This Figure is a pictoral representation of how each input has its own fitting equation that the function, f, is supposed to map to eight possible outputs depending on how our Genetic Algorithm function, here f, is trained.

our forecast. The ratio of trained potential outputs changes as we compare the forecasted output to the real output. If the gene code for the s parameter is good, we will allocate a "survival + reproduction chance" rate of $rate_s = 0.9$, otherwise 0.8. Likewise if the w parameter is good we will allocate a "survival + reproduction chance" $rate_w = 0.9$, otherwise 0.85. The total rate is defined as being $rate_{total_t} = rate_{s_t} \times rate_{w_t} + \epsilon_t$ where ϵ_t is an error uniformly distributed between 0 and 1. If $rate_{total_t} > 1$ we allocate the corresponding individual in the set of parents. 200 Couples are then randomly created and a child is born via this reproduction. The child is a result of a combination of crossover between the parents and genetic mutation set to 8/1000, a number chosen in the range of [0.005 0.01] [13]. From the resulting population, 200 are randomly chosen, the others therefore "die" and the final population is saved until the next time the same input is called.

The relevant programs can be found in the appendix. The name of the functions are:

```
binaryToDecimal, get_binaryRepfor8,
initializePopulation, changePopulation, getProportions,
crossover, mutation,
train_GA,
test_train_GAcut and
test_train_GArol.
```

4 Multivariate Approach

4.1 A Time Complexity Issue

As promised in the introduction, and as we have seen in the last few sections, there exist many different univariate methods that can forecast an economic indicator. However, given the complexity of the function we are trying to model and given the many additional complexity issues, potentially arousing from using the other relevant methods we thought focusing on Neural Networks for our multivariate model was the best thing to do. Supporting this idea, Min Qi mentions: "It is commonly agreed among economists that business cycles are asymmetric and cannot be adequately accommodated by linear constant parameter single-index models. NNs are a class of flexible nonlinear models. Given enough data, they can approximate almost any functions arbitrarily close. Because there is little a priori knowledge about the true underlying function that relates financial, economic and composite indicators to the probability of future recessions, the NN models are an ideal choice for modeling these relationships".
We were studying the possibility of initially using a multiple linear regression as a reference to our study when the time complexity issue arose. Note that this is a primordial problem since directly linked on what data to choose for our design. To emphasis, we have collected 40 potential leading indicators but do not know the importance of each of these leading indicators and what lag operator to use for each of them. From the literature search we have done, we think that an initial shift of 5 days would be a good choice, but the likelihood that this shift is different given the different indicators is obviously high. This ambiguity lead us in thinking of simply writing a program with 40 nested loops making the shift range between 1 and 15 days and make a multiple linear regression with each of these possible sets of data and get the best one. However, this method would calculate 15^{40} linear regressions which is 1.12×10^{47}. If each linear regression takes 1 second to calculate the whole thing would take 3.5×10^{39} years.

4.2 Proposed Method for Reducing the Time Complexity Issue

This being an obvious problem, we then thought of diminishing the number of leading indicators without endangering the training process. As we have seen earlier, in an interview with Dr. Zandi, Emeritus Professor of Penn Engineering and the Wharton School: "in order to get viable results with neural networks we will need approximately at least 15 times as many rows than we have columns". Since we have around 2100 rows of data, if we intend to have an initial training set of 1000 rows we may not want to have more than 66 (1000/15) leading indicators which is actually fine given that we have selected 40 initial indicators. Considering our time complexity issue, we thought of diminishing the two ranges: 15 for the number of different shifts and 40 for the number of leading indicators to 4 and 18 respectively which reduces the computation time to around 2000 years. This being still way to high, we came up to the conclusion that fixing the redundant data rather than eliminating it was the best thing to do. The idea is that the higher the correlation between the different leading indicators at certain shift s is, the higher is the likelihood that these indicators impact the value of the S&P500 at the same "shift s" from one and other (Figure 5). This method although perhaps arguable stays however the most efficient way of solving our problem. Note that, even if we do not capture the exact optimal lag value for each indicator, provided that we stay close enough, methods like Neural Networks will always approximate a decent function if there is any pattern to capture. By fixing here, I mean looking for correlation between the different leading indicators and then run the program in a way that would keep the shift between these indicators constant so that we have fewer cases to check. We can say that in the worst case scenario the shift between the S&P500 and the leading indicator with the biggest shift is 10 days. From there finding "redundant" leading indicator becomes easier. We

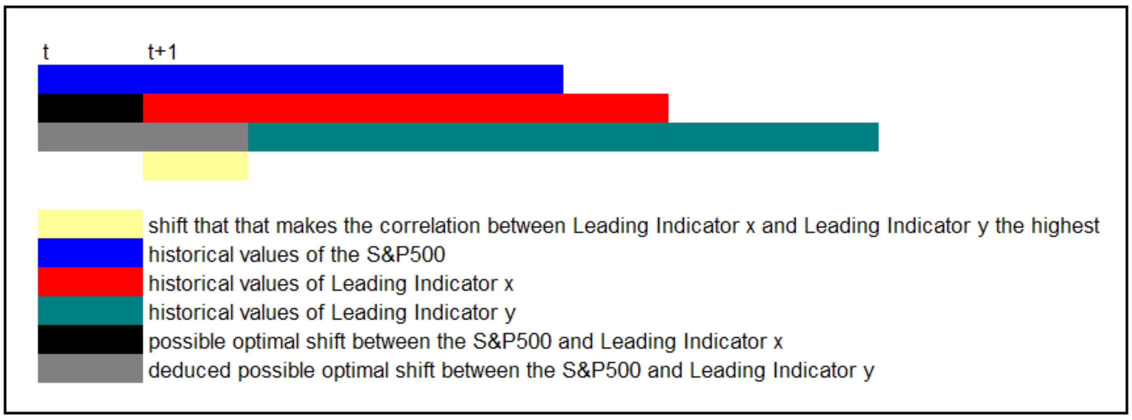

t
t+1
shift that that makes the correlation between Leading Indicator x and Leading Indicator y the highest
historical values of the S&P500
historical values of Leading Indicator x
historical values of Leading Indicator y
possible optimal shift between the S&P500 and Leading Indicator x
deduced possible optimal shift between the S&P500 and Leading Indicator y

Figure 5: This Figure represents how the shift between two leading indicators is fixed thanks to the maximum correlation method and how the shift for the leading indicator y is deduced from finding the optimal shift for the leading indicator x.

need to check the correlation between the different leading indicators at different shift [-4 4]. We aim at spending less than let's say 1 day for our final computation and therefore need to reduce our data accordingly: so $10^n = 60 \times 24 = 1440$ seconds, therefore $nln(10) = ln(1440)$, we get $n = 6$. Therefore we will have to select a range of leading indicators so that their shift is fixed in 6 different possible time's shifts. Again, having $n = 6$ does not mean that I will have to choose amongst 6 leading indicators but limit my sets of possible shifts to 6.

The relevant programs can be found in the appendix. The names of the functions are:

`hashGet,`

`CorrUsingBestShiftSingleton and`

`CorrUsingBestShift.`

Unfortunately for us, the results of our algorithm did not quite work the way we wanted. Our program is correct and simply enumerates the different correlations, shifts and corresponding indicators but shows a bug in our method itself. The indicators do not order themselves in an interesting way. To be more precise given our assumption that indicators with highest mutual correlation would impact the value of the S&P500 at the same lag from one and other, we get the following problem. Our model supposes transitivity because of the boundaries of our problem that is: any given indicator has to be around a shift of 5 with the S&P500 with a potential error of + or -2. If the best lag between indicator L1 and L2 is 3, and the best lag between L3 and L1 is 2, if at the best case L2 is found at a shift of 3 (5-2), one of our boundaries, then that would mean that L3 would be out bound, which would first not solve our complexity issue and would be going against the finance literature.
The relevant programs can be found in the appendix. The name of the function are hashGet, CorrUsingBestShiftSingleton and CorrUsingBestShift.
A second way to solve this problem is to simply check the correlation between the return of each leading indicator with the returns of the S&P500 at different lags and memorize the lag that offers the highest correlation for each indicator.

$$R(i_{a:b}, j_{a:b}, k) = \left| \frac{C(\Delta i_{a:b}, \Delta j_{a':b'}, k)}{\sqrt{C(\Delta i_{a:b}, \Delta i_{a:b}, k) C(\Delta j_{a':b'}, \Delta j_{a':b'}, k)}} \right|$$

- where $C(\Delta i_{a:b}, \Delta j_{a':b'}, k) = E((\Delta i_{a:b} - \mu)(\Delta j_{a'+k:b'+k} - \nu))$
- and $\mu = E(\Delta i_{a:b})$, $\nu = E(\Delta j_{a'+k:b'+k})$

This method even though weak still remains the best known way to eliminate this time complexity issue. It is weak because at lag l, one may happen to see that the correlation between the returns of the S&P500 and the return of leading indicator I is the highest by chance. That is in a multivariate design this particular lag would have not had the same positive impact. However it is still considered as the best approach because the probability that the lag l is the best compared to any other lag is the highest since it is the point at which the correlation is the highest. Note that here when we talk about correlation we really talk about absolute value of the correlation.
The relevant program can be found in the appendix. The name of the function is:

`shiftForMaxCorr.`

4.3 A Naive Neural Network Design

Since we have established that given our resources Neural Networks are the best available tool for a multivariate approach, we thought that we would first randomly select our

design and input all of our available data to see whether the design is as important as the data itself. To summarize we will make a very naive first multivariate design.
So in this part we were interested in seeing whether a lag of 5 days within our leading indicators and the S&P500 would yield satisfactory results and therefore abandoning the idea that there might be different shifts for each leading indicators around this value of five.
We decided to create a neural network of the form 8, 4, 2, 1 with a sigmoid activation function for the first three layers and a linear activation for function for the output layer. The relevant programs can be found in the appendix. The names of the functions are

`test_Train_with_simple_FF_40InputsCut`,
`Train_with_simple_FF_40Inputs` and
`test_Train_with_simple_FF_40InputsRol`.

4.4 Gately's Method

As a second design we thought more or less replicating one of the references in Neural Network designs applied to the forecasting of the S&P500 to see if we can get good result before making our own design from scratch. Note that what really counts here is not really the design of the network itself but the leading indicators used as well as the lag with the S&P500, to summarize the input data. This simply because given a dataset, several designs can yield similar training performance so that since we dynamically retrain the network at every step if we can build a network that will be able to perform as well but faster then we will use the fastest network. Gately [15] basically says that the leading indicators of a given day are "High", "Low", "Volume", "SP", "DJT", "DJU", "DJI", "moving average 30", "Oil Index" and "Gold and Silver". He also mentions that the lag of these indicators with the S&P500 is 10 days.
The relevant programs can be found in the appendix. The names of the functions are:

`train_Gately`,
`test_train_GatelyCut` and
`test_train_GatelyRol`.

4.5 A Final and Better Design

As a final design we wanted to combine everything we have learned, that is

- what machine learning technique is best to use.
- what set of leading indicators to use.

- what lag should we use for each individual leading indicator.
- whether or not a ”cut and forecast” or ”rolling forecast” is best to use.
- if a ”rolling forecast” is the best, then what type of forgetting factor to use.

First then, we have decided to pick Neural Networks as the preferred method of forecasting because in theory it is the only method that we have used capable of picking up a non linear pattern, an obvious advantage when it comes to forecasting markets. However, one of the main differences from our univariate methods is that we have chosen a Resilient Backpropagation algorithm [3, 4] instead of the regular gradient descent because of the fact that it works faster and better for a multivariate purpose. The help section of Matlab gives a good description of what this algorithm does. ”Multilayer networks typically use sigmoid transfer functions in the hidden layers. These functions are often called ”squashing” functions, because they compress an infinite input range into a finite output range. Sigmoid functions are characterized by the fact that their slopes must approach zero as the input gets large. This causes a problem when you use steepest descent to train a multilayer network with sigmoid functions, because the gradient can have a very small magnitude and, therefore, cause small changes in the weights and biases, even though the weights and biases are far from their optimal values.
The purpose of the resilient backpropagation (Rprop) training algorithm is to eliminate these harmful effects of the magnitudes of the partial derivatives. Only the sign of the derivative is used to determine the direction of the weight update; the magnitude of the derivative has no effect on the weight update. The size of the weight change is determined by a separate update value. The update value for each weight and bias is increased by a factor deltInc whenever the derivative of the performance function with respect to that weight has the same sign for two successive iterations. The update value is decreased by a factor deltDec whenever the derivative with respect to that weight changes sign from the previous iteration. If the derivative is zero, then the update value remains the same. Whenever the weights are oscillating, the weight change is reduced. If the weight continues to change in the same direction for several iterations, then the magnitude of the weight change increases.”
Otherwise, the method remains quite similar with a network of the form [10 15 1] with tan-sigmoid activation function for all the neuron. Otherwise we have 13 inputs for one output. The learning rate remains at 0.35 but the momentum is disabled for the initial part. The number of epochs is 1000 enough to reach an error close to 10E-4, a good approaximate for the training part.
Our leading indicators are the following: Dow Jones (lag 5 days), News (lag zero day),COT S&P500 Stock Index Futures for Commercial Traders Net (lag 15 days), COT S&P500

Stock Index Futures for Non Commercial Traders Net(lag 5 days), Volume S&P500 (lag 5 days), Dow Jones Transportation (lag 5 days), Dow Jones Utilities (lag 5 days), AMEX Oil Index (lag 5 days), Gold and Silver (lag 5 days), COT Mini S&P500 for Non Commercial Traders Net (lag 15 days), COT Mini S&P500 Commercial Traders Net (lag 15 days) and finally the S&P500 itself at lag one and two in order to capture our best benchmark so far. Note that these indicators and their lags are a combination of Gately's [15] work, our experiences and three 5 new indicators with for of them being data related to the Commitment of Traders (COT) and the last one being News. News is an external factor that might come and confuse the network's training. For example, the collapse of the world trade centers (Figure 6) as well as wars or news about strong employment report (Figure 7) are unpredictable and can have an impact on the stock indexes as one can see from the figures included.

These news, because they so drastically impact the value of the S&P500 have to be

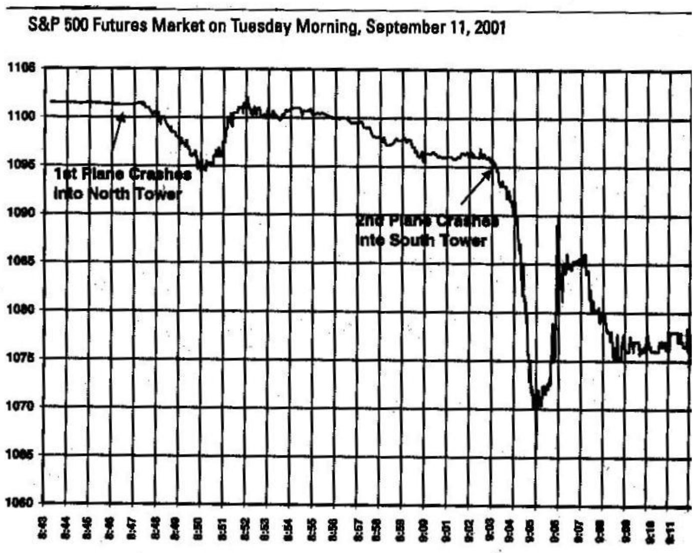


Figure 6: This Figure represents the impact of the world trade center collapse on the value of the S&P500 [27].

somehow taken into account into the design. As it is impossible to list them as "leading indicators" because, again unpredictable, what we can do is to take them into account as "coincident indicator" and therefore correct our input so that we have the best design possible in a world "free of catastrophes". Note that this will theoretically enable the network to predict the stock market in situations where there is no major breaking news (the majority of times).

Otherwise, as we have done previously, we will look at a rolling or a cut forecast while including a forgetting factor to abide by the general rule that states that the stock markets evolves constantly.

The relevant programs can be found in the appendix. The names of the functions are:

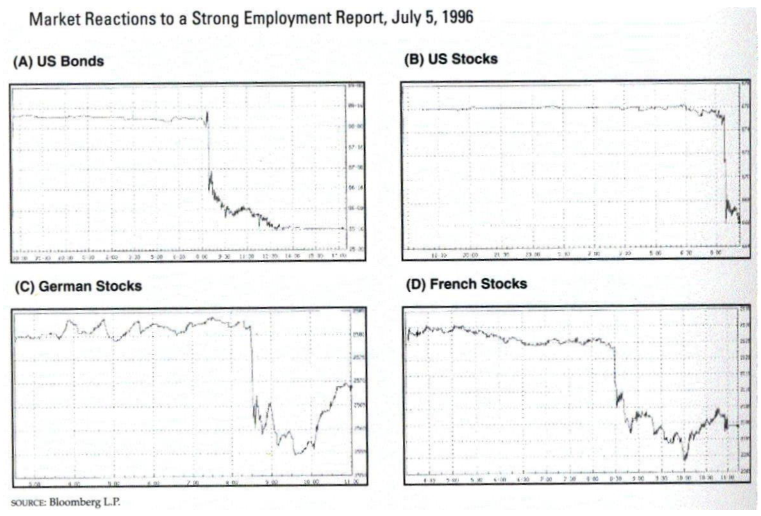


Figure 7: This Figure represents the impact of the world trade center collapse on few financial indexes [5].

`train_Final_BetterDesign`,
`test_Final_BetterDesignCut` and
`test_Final_BetterDesignRol`.

5 Methods of Analysis

5.1 Traditional Statistical Methods

We will perform several tests in order to analyze the performance of our designs. The two available designs per methods are:
- The "cut and forecast" which is basically taking the data, splitting it in two and using the trained model to make a forecast for the remainder of the data.
- The "rolling forecast" which is like in the "cut and forecast" technique, taking the data, splitting it in two and using the trained model to make a forecast for the remainder of the data with the difference that we dynamically retrain the model at every step and introduce a forgetting factor.
The simplest statistical method we will use is the success (directional) rate which we will define the following way:

$$SR = \frac{\sum_{t=i}^{j} f(x_t, \hat{x_t})}{j - i}$$

- where: f is a function that takes two parameters multiply them together and returns 1 if the product is positive and 0 otherwise:
  if $x_t \times \hat{x_t} > 0$ then return 1, else 0. So that if the predicted return and real return has the same sign, the success is incremented by one.

- and: i and j are indexes of the S&P500 in the testing section of our dataset.
- logically the higher SR is the better is our design

We will also look at the error distribution with error at time t being defined as:

$$\varepsilon_t = \hat{x_t} - x_t$$

- where $\hat{x_t}$ is the estimate forecasted using the trained design with new data and x_t being the corresponding real return value.

We will also use the root mean square error defined the following way [12]:

$$RMSE_{i:j} = \sqrt{\overline{\varepsilon^2_{i:j}}}$$

- where $i : j$ represents a range of value between i and j
- and $\overline{\varepsilon^2_{i:j}} = \left(\sum_{t=i}^{j} \varepsilon_t^2\right) / (j - i)$
- the smaller is the RMSE, the better our design is.

The Normalised Root Mean Square Error will also be used which is defined as:

$$NRMSE_{i:j} = \frac{RMSE_{i:j}}{s_{i:j}(x_t)}$$

- where $s_{i:j}(x_t) = \sqrt{\frac{1}{j-i}\sum_{t=i}^{j}(\varepsilon_t - \bar{\varepsilon})^2}$

The Mean Absolute Error will be also used [12].

$$MAE_{i:j} = \frac{\sum_{t=i}^{j}|\varepsilon_t|}{j-i}$$

The smaller is the MAE, the better is our design. The Mean Absolute Percentage Error will be also used [12].

$$MAPE_{i:j} = 100 \times \frac{\sum_{t=i}^{j}\left|\frac{\varepsilon_t}{100}\right|}{j-i}$$

The smaller is the MAPE, the better is our design.
We will also calculate when relevant the 95% confidence intervale given by [12].

$$95CI = \left[\bar{X} - 1.96 \times \frac{\sigma}{\sqrt{n}}, \bar{X} + 1.96 \times \frac{\sigma}{\sqrt{n}}\right]$$

The relevant programs can be found in the appendix. The names of the functions are:

```
SR,
RMSE,
NRMSE,
MAE,
MAPE and
ConfidenceIntervalCI30.
```

5.2 Non-Traditional Statistical Methods

In selecting what our best design is we will look at the traditional statistical methods, and if it happens that few of our designs are of similar quality we may argue against or for one design using some of the argument of the Non-Traditional Statistical Methods. Note that amongst the CIS, CI, HSRI and CI30 that we are about to look at CI30 is the most rigorous of all and we will put an equal weight to the results of this method as the other traditional methods such as RMSE for example.
One interesting analysis would be to check how the forecasted returns compare to the real return as the design is increasingly "sure" of its bet. This is a test of our own inspiration. Note the quotation marks on the word sure. We wanted to emphasis on the term here because certainty amongst statisticians is a concept that involves confidence interval after

careful analysis of a sample data over a known population. Nevertheless, the correctness of our design in its ability to pick up the right pattern put aside, this statistical formula still translates the "certainty" of the later in its bet for the stocks movement. One possible mathematical model that would perhaps translate our idea is the following and that we will call the Certainty Index or CI:

$$CI = \left(|\hat{\xi_t} - \xi_{t-1}| \geq k.\overline{|\Delta\xi_{1:(t-1)}|}\right) \cup \left(\xi_t.\hat{\xi_t} \geq 0\right) \Rightarrow incrementSuccess$$

- where k is a constant representing a certainty index.
- ξ_t being the return so that $\xi_t = x_t - x_{t-1}$. We will sometimes define $\xi_t = (x_t - x_{t-1})/(x_{t-1})$ depending on the output of the algorithm we use.
- and $\overline{|\Delta\xi_{1:(t-1)}|} = (\sum_{t=1}^{t} |\Delta\xi_t|)/(j-i)$. Note that this along with k only provides a way to normalize the data.
- if the SR (success rate) increase as the certainty index increases, the better is our design.
- Note that the number of counts diminishes as k becomes bigger. Second, as k becomes bigger, if the success rate increase then this brings another potential argument as for the efficiency of the design. As "count" becomes smaller than 30 the success rate may become unstable because of the simple fact that they are not enough results to come up with a definitive answer. Note that this does not necessarily mean that the design is not picking something up at these very high certainty indexes and vice versa.

We will define the function CIS, a function whose aim will be to capture how consistent is the CI as k increases.

$$CIS = \chi(CI)$$

- where $\chi(CI)$ increments its success rate every time the next element in the array CI is bigger than its previous values.
- remember that the CI (certainty index) aims at finding how the system's success evolves as it is increasing sure of its bet.
- the higher CIS is the better our design is.

In the same family of statistical techniques we will also look at the certainty index that is closest yet above 30, to capture the last certainty index that has a population above 30 and therefore means something.

$$CI30 = c_1, c_2 \in CI.(c_1 \geq 30) \cup (!\exists c_2.((c_2.count \geq 30) \cup (c_2.count < c_1.count)))$$

One final test we will perform is once again, one of our own designs. This one is aimed at showing the success rate throughout its history. The idea is linked to one of the newest area of finance that is behavioral finance. Basically one could say that investors behave in a particular manner throughout time as they are aware or not of an arbitrage opportunity. To help us grasp this concept we have looked in the literature and have found the following very useful information and that can be viewed in the next subsection. In any case we will call this test, the Historical Success Rate or HSR. HSR will be defined as a one dimensional array of 20 elements such that:

$$HSR_{i:j} = [SR_{i:\frac{j}{20}}, SR_{\frac{j}{20}:\frac{2\times j}{20}}, \cdots, SR_{\frac{18\times j}{20}:\frac{19\times j}{20}}, SR_{\frac{19\times j}{20}:j}]$$

- where $SR_{i:j}$ is the success rate, previously defined, in the interval $[ij]$

We will define the Historical Success Rate Index (HSRI) as yet another way of assessing our design's performance. It is rather difficult to explain quickly why this index works but for now let's define its formula and what the output of this formula means before getting into the details.

$$HSRI_{i:j} = \Psi(SR_{i:\frac{j}{20}}, SR_{\frac{j}{20}:\frac{2\times j}{20}}) + \cdots + \Psi(SR_{\frac{18\times j}{20}:\frac{19\times j}{20}}, SR_{\frac{19\times j}{20}:j})$$

- where $\Psi(a_t, a_{t+1}) = |a_t - a_{t+1}|\Phi(a_t, a_{t+1})$, with $\Phi(a_t, a_{t+1})$ returning the following: if ($a_t < a_{t+1}$ and $a_t > 50\%$)OR($a_t > a_{t+1}$ and $a_t < 50\%$) then return $a_t - a_{t+1}$, else $-a_t + a_{t+1}$, and vice versa when $a_t > a_{t+1}$. The index returns the average of these returns.

- the higher is the HSRI, the better is our design.

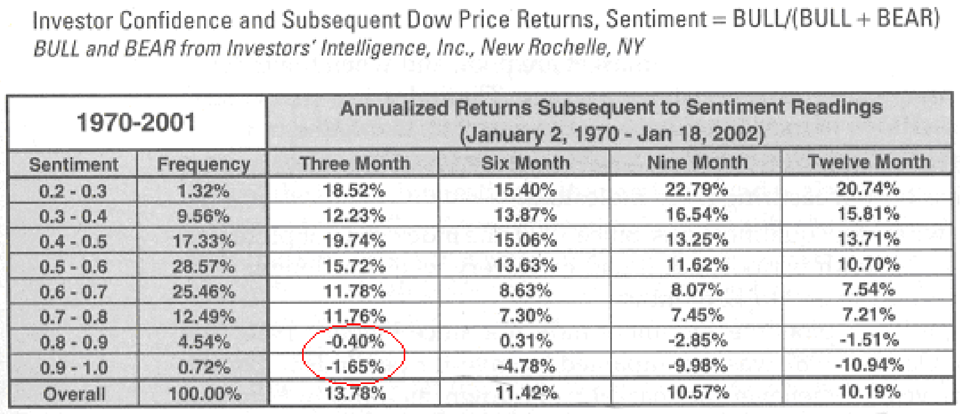
The relevant programs can be found in the appendix. The names of the functions are:

```
CI,
CertaintyIndexSum,
CI30 and
HSRI.
```

5.3 A Note on Behavioral Finance

The two most relevant indicators related to behavioral finance are Investor Sentiment and Expectation. The reason why we have incorporated these two different leading indicators

into the same paragraph is because people tend to interchange them in the literature. In reality Investor sentiment is what "normal people" (non professional investors) perceive the economy doing in the close future whereas expectation is an actual number related to purchase on futures. Traders who take commercial positions to hedge a specific risk are referred to as large hedgers, those who take noncommercial positions for reasons other than hedging are referred to as large speculators and traders whose positions do not exceed the CFTC's reporting threshold are referred to as small traders.
Stock prices are based not just on economic values but also on psychological factors that influence the mood of the market. "The market is most dangerous when it looks best; it is most inviting when it looks worst[34]". Psychologists call this penchant to follow the crowd the herding instinct. Men it has been well said, think in herds they go mad in herds, while they only recover their senses slowly and one by one. Economists describe the decision-making process an information cascade. For longer-term investors, standing apart from the crowd might be quite profitable, such an investor is said to be a contrarian. "When everyone thinks alike, everyone is likely to be wrong"[21]. Some contrarian approaches focus on fundamentals, such as earnings, prices, or dividends, and others are based on more psychologically driven indicators such as investor sentiment. Dr. Siegel, from Investors' intelligent data, computed an index of investor sentiment by finding the ratio of bullish newsletters to bullish plus bearish newsletters (omitting the neutral category). Whenever the index of investor sentiment is high subsequent returns on the market are poor, and when the index is low, subsequent returns are above average. This index is a particularly strong predictor of market return over the next 9 to 12 months. The Figure 8 shows evidence of the inverse relationship between stock returns and investors' sentiment. The reason why we mention this is that usually when there is an arbitrage



Investor Confidence and Subsequent Dow Price Returns, Sentiment = BULL/(BULL + BEAR)
BULL and BEAR from Investors' Intelligence, Inc., New Rochelle, NY

1970-2001		Annualized Returns Subsequent to Sentiment Readings (January 2, 1970 - Jan 18, 2002)			
Sentiment	Frequency	Three Month	Six Month	Nine Month	Twelve Month
0.2 - 0.3	1.32%	18.52%	15.40%	22.79%	20.74%
0.3 - 0.4	9.56%	12.23%	13.87%	16.54%	15.81%
0.4 - 0.5	17.33%	19.74%	15.06%	13.25%	13.71%
0.5 - 0.6	28.57%	15.72%	13.63%	11.62%	10.70%
0.6 - 0.7	25.46%	11.78%	8.63%	8.07%	7.54%
0.7 - 0.8	12.49%	11.76%	7.30%	7.45%	7.21%
0.8 - 0.9	4.54%	-0.40%	0.31%	-2.85%	-1.51%
0.9 - 1.0	0.72%	-1.65%	-4.78%	-9.98%	-10.94%
Overall	100.00%	13.78%	11.42%	10.57%	10.19%

Figure 8: This Figure represents the annualized returns of the Dow subsequent to sentiment reading [27, 16]

opportunity, large speculators pick it up quickly and take the right action so as to make a profit. Since the late 80's we have witnessed a real revolution in the branch of Artificial

Intelligence [30] and an increasing number of engineers and scientists have used some of the new tools available to make a forecast. Some people have even made an art of it by making books specialized in this area [36, 17, 26]. The problem with any arbitrage opportunity is that it is usually very ephemeral. People seeing the arbitrage take the right action and the arbitrage disappears. For books about Neural Networks applied to forecasting to be that successful in the industry, one may extrapolate that it in fact works. But bearing in mind the idea of an arbitrage is brief, one could argue that the success rate of the forecast should be cyclical because as the first people see the arbitrage they take action, which absorbs the potential profit by impacting the model and as subsequent investors take on the same position and instead of performing the way we expect, the system performs exactly the opposite: "When everyone thinks alike, everyone is likely to be wrong" [21]. As people realize that the arbitrage opportunity is absorbed, they withdraw their initial position and the systems performs the way it did initially: The Historical Success Rate Index tries to capture this idea of cyclicality. Basically when the success rate at interval t, I_t is below 50%, then if the design is good, one should expect the success rate to rise at interval $t+1$. Likewise if the success rate is above 50% at interval t, I_t, then if the design is good, one should expect the success rate to decrease at interval $t+1$. We assign one point every time this pattern is respected, 0 otherwise. Note that we have added some sophistication to this simplistic model by multiplying the point by some weight that linearly increases or decreases as the rate approaches 50% from above or below so that at points where the investors may be confused (close to 50% success rate) then it is not as bad to make a mistake there.

6 Results

In this section we will examine the results of our different methods by looking at several statistical techniques in order to both see if any of our codes might have a bug and also check the success of our designs in picking pattern up. Note that even though the Error, the RMSE, the NRMSE, the MAE and the MAPE are standard statistical techniques, we have designed the rest so as to clarify certain ideas that we wanted to translate.
In the next subsections U- and M- stands for respectively Univariate and Multivariate. RW stands for Random Walk, ARIMA stands for Auto-Regressive Integrated Moving Average, PCA stands for Principal Component Analysis, PM stands for Probabilistic Modeling (in our case second order markov chains), GA for Genetic Algorithm, NN for Neural Networks, NNN for Naive Neural Networks, GNN for Gately Neural Networks and BNN for Better Neural Networks. Before going further though, it is interesting to look at our data distribution so as to have a reference (Figure 9).

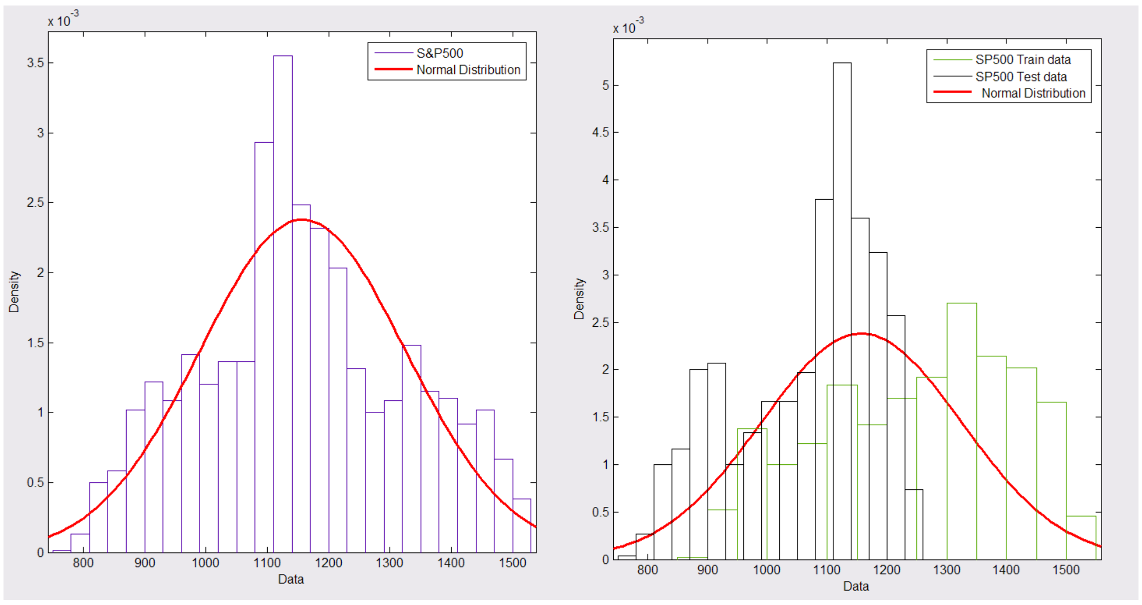
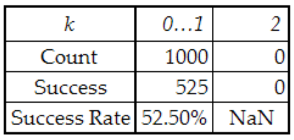


Figure 9: On the left Figure represents the distribution of the S&P500 including training and testing set. On the right Figure represents the distribution of the S&P500 seperating training and testing set.

6.1 U-RW

Below the results for the univariate Random Walk Design. For the "cut and forecast" method we found the following results:

- RMSE = 1.58%.
- NRMSE = 142.16%.
- MAE = 1.31%.
- MAPE = 1.3E-04.
- SR = 52.50%.
- CI (Figure 10, the table does not show anything really interesting because for this design it does not make sense to calculate the CI but it was however still done to comply with the template used in the other techniques)



k	*0...1*	2
Count	1000	0
Success	525	0
Success Rate	52.50%	NaN

Figure 10: Certainty Index for the RW "cutting forecasting" method

- CIS = 0.00.%
- HSRI = 0.00.
- CI30 = 52.50%.

We can also have a visual representation of our results in Figure 11.
Note that for this cut and forecast, we picked the forecasted value for our target indicator at time step 1000 and used this value as our trained designed. The return at time 1000 happened to be positive which accounts for this enormous cumulative positive error. The error is so big with respect to the actual value that one cannot see the fluctuations of the S&P500 anymore. This method might sound silly, and it is, but it is still good practice because we have done the same with all the other models.
Below the result of the real actual random walk which our RWr (random walk "rolling forecast") method we found the following results:

- RMSE = 1.61%.
- NRMSE = 99.95%.

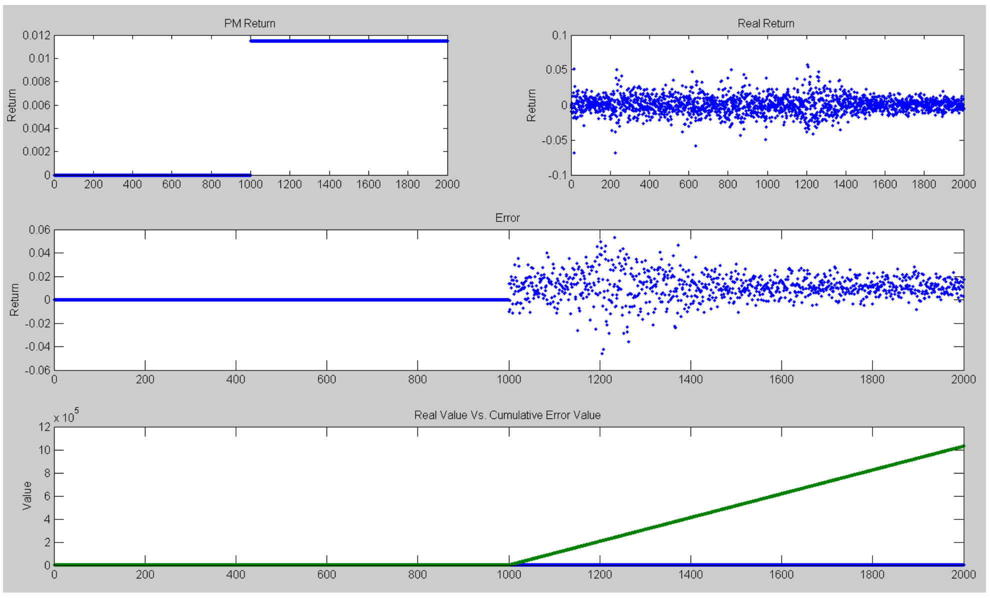
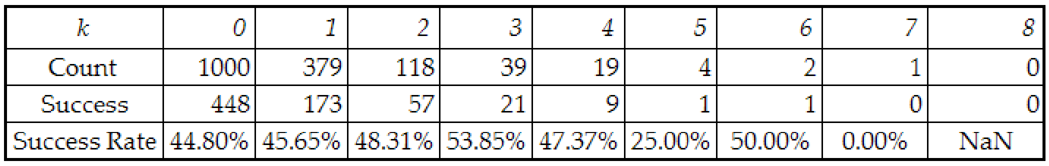


Figure 11: visual representation for the success of the RW "cut and forecast" method

- MAE = 1.21%.
- MAPE = 1.2E-04.
- SR = 44.60%.
- CI (Figure 12)



k	*0*	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*
Count	1000	379	118	39	19	4	2	1	0
Success	448	173	57	21	9	1	1	0	0
Success Rate	44.80%	45.65%	48.31%	53.85%	47.37%	25.00%	50.00%	0.00%	NaN

Figure 12: Certainty Index for the RW "rolling forecasting" method

- CIS = 42.9%
- HSRI = 8.99.
- CI30 = 53.9%.

We can also have a visual representation of our results and error distribution, respectively in Figure 13 and Figure 14.

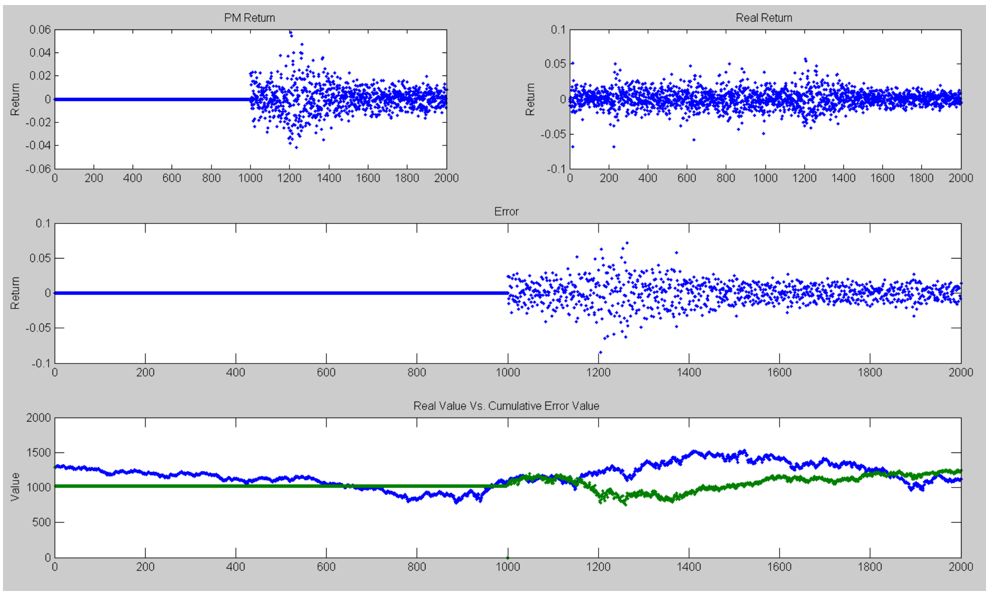


Figure 13: visual representation for the success of the RW "rolling forecasting" method

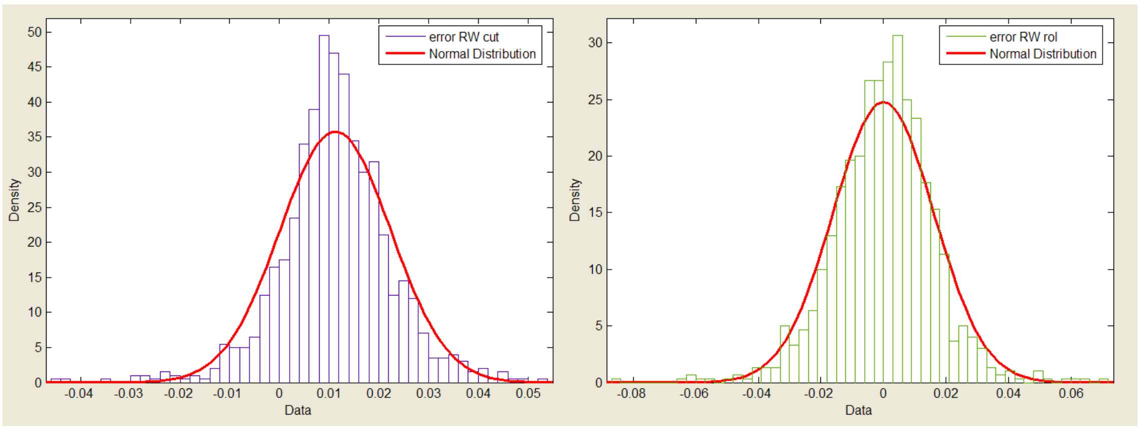


Figure 14: This Figure represents the error distribution using our two RW models.

6.2 U-ARIMA

Below the results for the univariate Autoregressive Integrated Moving Average Design:
As one can see visually from Figure 15, increasing the p and q parameter of an ARIMA model does not change its forecasting ability. We will choose as our benchmark the simplest model.
Intuitively, one could think that the best choice of p, d and q for our model would be the ARIMA(1,1,1) which simply corresponds to:

$$X_t = \phi X_{t-1} + \epsilon_t$$

where ϕ represents the average increase of the S&P 500 during the training period and ϵ_t normally distributed with mean 0 and variance σ^2: $\epsilon_t \sim N(0, \sigma^2)$. But since our

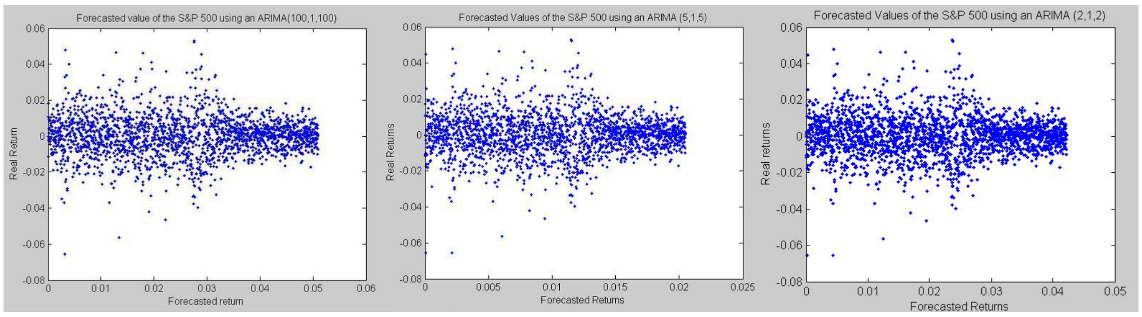


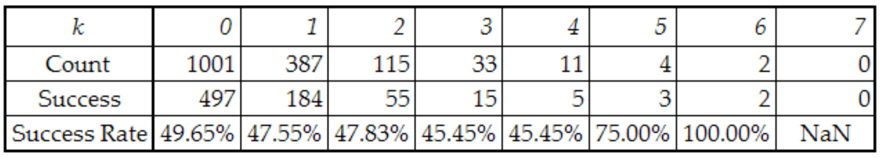
Figure 15: This Figure represents the real value of the S&P500 and the forecasted value using an ARIMA(100,1,100), an ARIMA(5,1,5) and an ARIMA(2,1,2).

confidence for the forecasted function is null, a more humble way to represent our data would simply to use the ARIMA(0,1,0) which corresponds to a random walk and which is the model that agrees the most with the literature:

$$X_t = X_{t-1} + \epsilon_t$$

For the "cut and forecast" method we found the following results:

- RMSE = 0.89%.
- NRMSE = 99.98%.
- MAE = 0.47%.
- MAPE = 4.71E-05.
- SR = 49.60%.
- CI (Figure 16)



k	*0*	*1*	*2*	*3*	*4*	*5*	*6*	*7*
Count	1001	387	115	33	11	4	2	0
Success	497	184	55	15	5	3	2	0
Success Rate	49.65%	47.55%	47.83%	45.45%	45.45%	75.00%	100.00%	NaN

Figure 16: Certainty Index for the ARIMA "cutting forecasting" method

- CIS = 60.00.%
- HSRI = 11.75.
- CI30 = 45.45%.

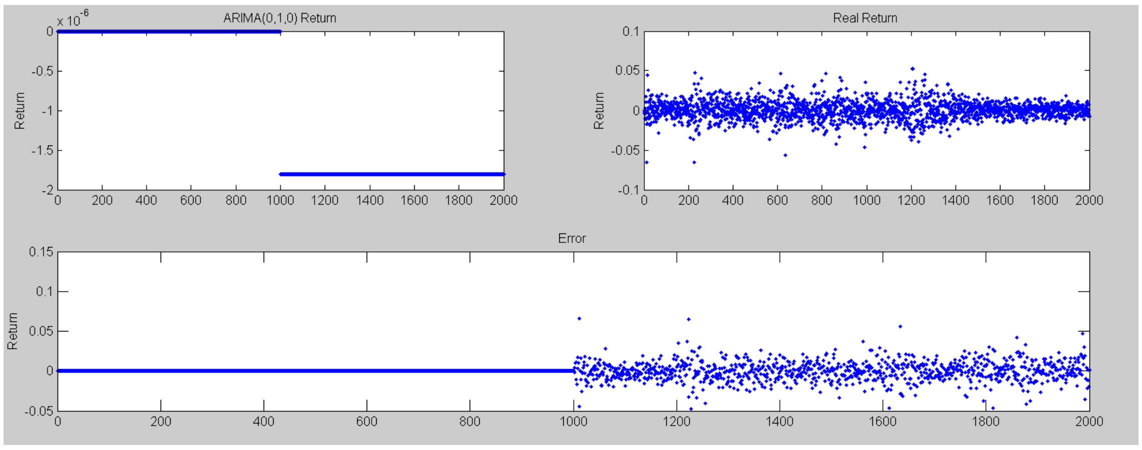
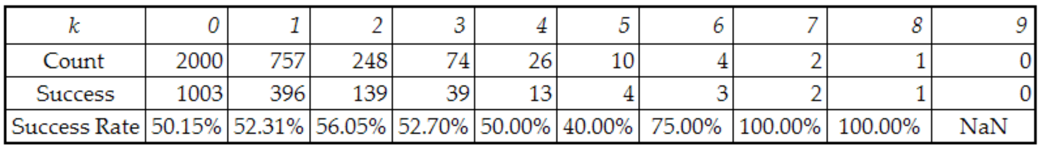


Figure 17: visual representation for the success of the ARIMA ”cut and forecast” method

We can also have a visual representation of our results and error distribution, respectively in Figure 17 and Figure 20.

For the ”rolling forecast” method we found the following results:

- RMSE = 1.17%.
- NRMSE = 100.02%.
- MAE = 0.86%.
- MAPE = 8.57E-05.
- SR = 51.75%.
- CI (Figure 18)



k	*0*	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*
Count	2000	757	248	74	26	10	4	2	1	0
Success	1003	396	139	39	13	4	3	2	1	0
Success Rate	50.15%	52.31%	56.05%	52.70%	50.00%	40.00%	75.00%	100.00%	100.00%	NaN

Figure 18: Certainty Index for the ARIMA ”rolling forecasting” method

- CIS = 55.56%
- HSRI = 9.11.
- CI30 = 52.70%.

We can also have a visual representation of our results and error distribution, respectively in Figure 19 and Figure 20.

As Dr. Patrick McSharry [1] was explaining us during an interview the main point with

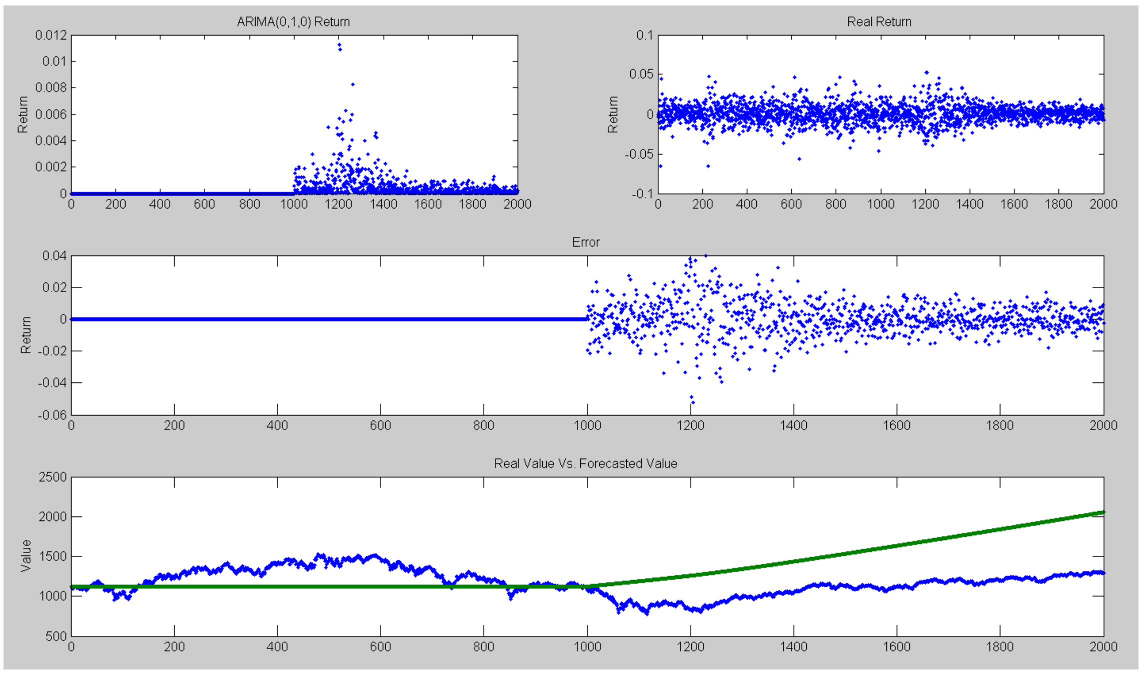


Figure 19: visual representation for the success of the ARIMA "rolling forecasting" method

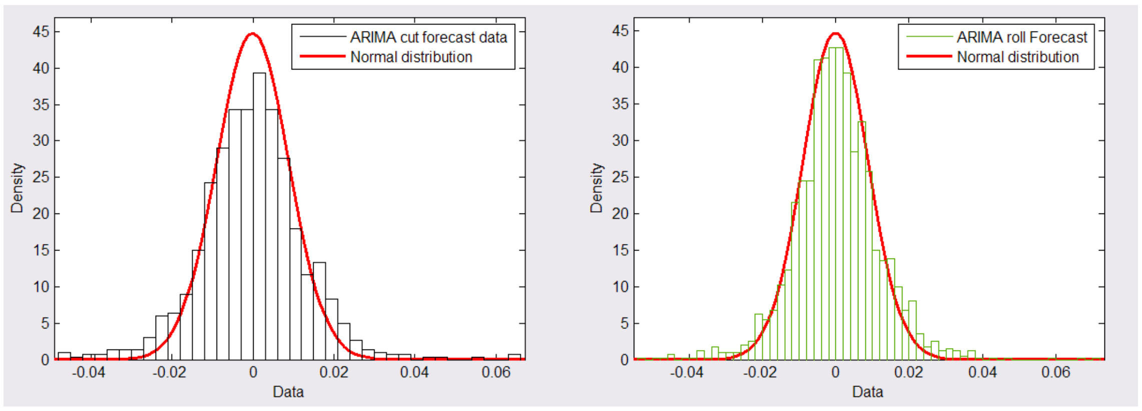


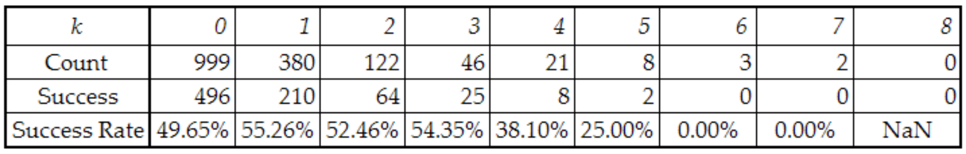
Figure 20: This Figure represents the error distribution using our two ARMA models. Note how the rolling forecast outputs a nice normally distributed error population whereas the cut and forecast method does not.

forecasting is that it is in general very difficult because of the number of bugs that can enter in the process. Without some good benchmarks, it is always difficult to spot these mistakes. In this sense, thinking of ways of breaking the model is as useful as accepting the results. For this reason, it is really important to contrast multivariate and univariate. Given that the former contains the latter, it should provide equal if not better performance. Our benchmark will therefore be the univariate ARIMA (0,1,0) model or simply the random walk. Therefore, from here any model we will study will have to do at least as well as the random walk.

6.3 U-PCA

Below the results for the univariate Principal Component Analysis Design:
First, for the "cut and forecast" method we found the following results:

- RMSE = 1.07%.
- NRMSE = 100.22%.
- MAE = 0.77%.
- MAPE = 7.66E-05.
- SR = 46.25%.
- CI (Figure 21)



k	*0*	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*
Count	999	380	122	46	21	8	3	2	0
Success	496	210	64	25	8	2	0	0	0
Success Rate	49.65%	55.26%	52.46%	54.35%	38.10%	25.00%	0.00%	0.00%	NaN

Figure 21: Certainty Index for the PCA "cutting forecasting" method

- CIS = 37.50.%
- HSRI = 8.08.
- CI30 = 54.35%.

We can also have a visual representation of our results and error distribution, respectively in Figure 22 and Figure 25.

For the "rolling forecast" method we found the following results:

- RMSE = 1.31%.
- NRMSE = 99.95%.
- MAE = 0.99%.

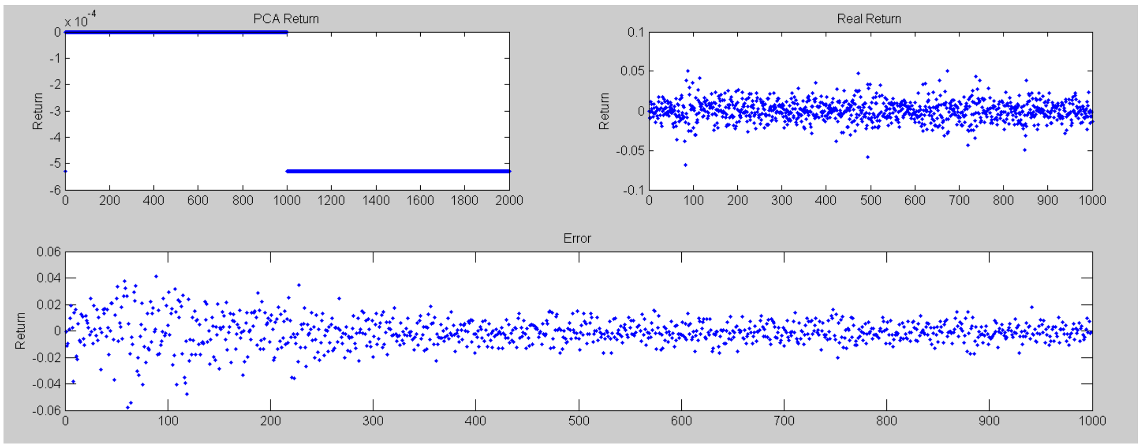


Figure 22: visual representation for the success of the PCA "cut and forecast" method

- MAPE = 9.89E-05.
- SR = 46.95%.
- CI (Figure 23)

k	*0*	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*
Count	999	375	116	46	21	6	3	1	0
Success	470	192	57	23	7	1	0	0	0
Success Rate	47.05%	51.20%	49.14%	50.00%	33.33%	16.67%	0.00%	0.00%	NaN

Figure 23: Certainty Index for the PCA "rolling forecasting" method

- CIS = 37.50%
- HSRI = 7.13.
- CI30 = 50.00%.

We can also have a visual representation of our results and error distribution, respectively in Figure 24 and Figure 25.

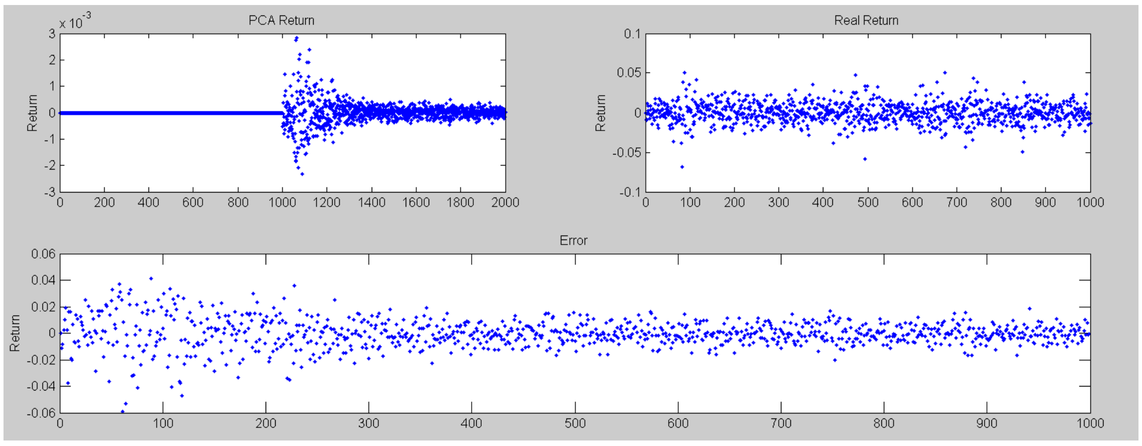


Figure 24: visual representation for the success of the PCA "rolling forecasting" method

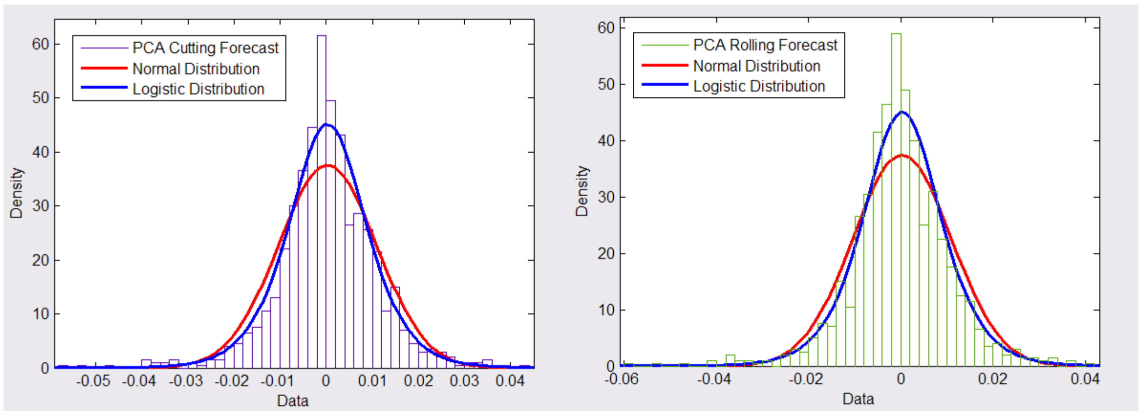
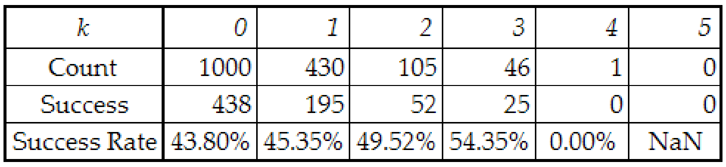


Figure 25: This Figure represents the error distribution using our two PCA models. Note how both of the distribution seem closer to the logistic population rather than the normal distribution.

6.4 U-PM

Below the results for the univariate Probabilistic Model Design (Second Order Markov Chain):
For the "cut and forecast" method we found the following results:

- RMSE = 1.63%.
- NRMSE = 100.13%.
- MAE = 0.92%.
- MAPE = 9.25E-05.
- SR = 43.80%.
- CI (Figure 26)
- CIS = 80.00.%



k	*0*	*1*	*2*	*3*	*4*	*5*
Count	1000	430	105	46	1	0
Success	438	195	52	25	0	0
Success Rate	43.80%	45.35%	49.52%	54.35%	0.00%	NaN

Figure 26: Certainty Index for the PM "cutting forecasting" method

- HSRI = 14.10.
- CI30 = 54.35%.

We can also have a visual representation of our results and error distribution, respectively in Figure 27 and Figure 30. We get the following probability matrix after 1000 rows of

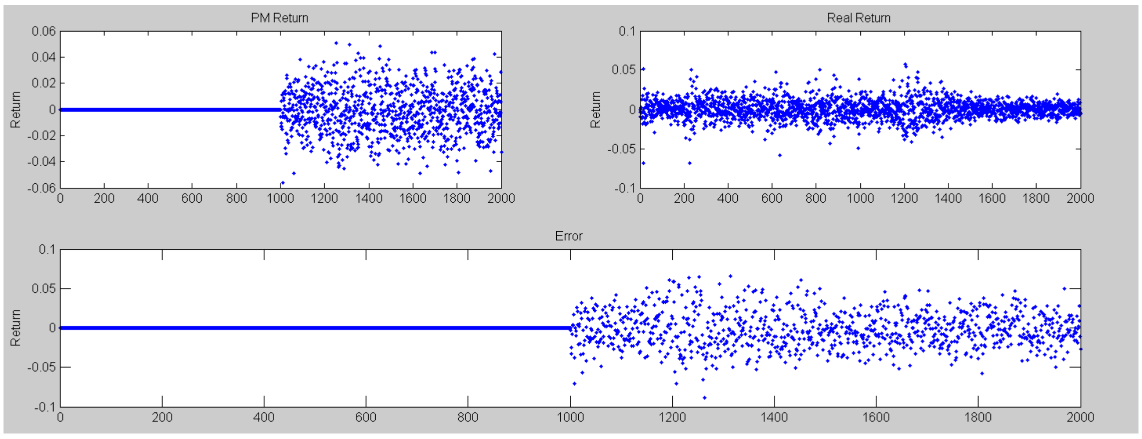


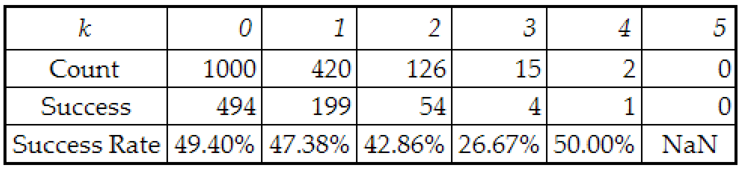
Figure 27: visual representation for the success of the PM "cut and forecast" method

training:

$$\begin{pmatrix} up_{t-2} & down_{t-1} & equal_{t-1} & up_{t-1} \\ - & down_t & down_t & down_t \\ - & equal_t & equal_t & equal_t \\ - & up_t & up_t & up_t \\ equal_{t-2} & down_{t-1} & equal_{t-1} & up_{t-1} \\ - & down_t & down_t & down_t \\ - & equal_t & equal_t & equal_t \\ - & up_t & up_t & up_t \\ down_{t-2} & down_{t-1} & equal_{t-1} & up_{t-1} \\ - & down_t & down_t & down_t \\ - & equal_t & equal_t & equal_t \\ - & up_t & up_t & up_t \end{pmatrix} = \begin{pmatrix} 33.33\% & 39.89\% & 31.47\% & 28.63\% \\ - & 34.03\% & 41.28\% & 45.00\% \\ - & 36.02\% & 21.37\% & 28.95\% \\ - & 29.95\% & 37.34\% & 26.05\% \\ 33.33\% & 27.69\% & 32.15\% & 40.16\% \\ - & 28.94\% & 23.30\% & 37.63\% \\ - & 42.81\% & 34.19\% & 28.90\% \\ - & 28.24\% & 42.50\% & 33.47\% \\ 33.33\% & 31.15\% & 38.33\% & 30.51\% \\ - & 35.33\% & 23.67\% & 44.82\% \\ - & 29.17\% & 33.55\% & 28.95\% \\ - & 35.50\% & 42.78\% & 26.23\% \end{pmatrix}$$

Note that this probability matrix also gives us the result of the first order Markov Chains as well in rows 1, 5 and 9 with columns 2,3 and 4.
For the "rolling forecast" method we found the following results:

- RMSE = 1.60%.
- NRMSE = 100.02%.
- MAE = 0.89%.
- MAPE = 8.93E-05.
- SR = 49.40%.
- CI (Figure 28)



k	*0*	*1*	*2*	*3*	*4*	*5*
Count	1000	420	126	15	2	0
Success	494	199	54	4	1	0
Success Rate	49.40%	47.38%	42.86%	26.67%	50.00%	NaN

Figure 28: Certainty Index for the PM "rolling forecast" method

- CIS = 40.00.%
- HSRI = 9.21.
- CI30 = 42.86%.

We can also have a visual representation of our results and error distribution, respectively in Figure 29 and Figure 30. We get the following probability matrix after 1000 rows of training:

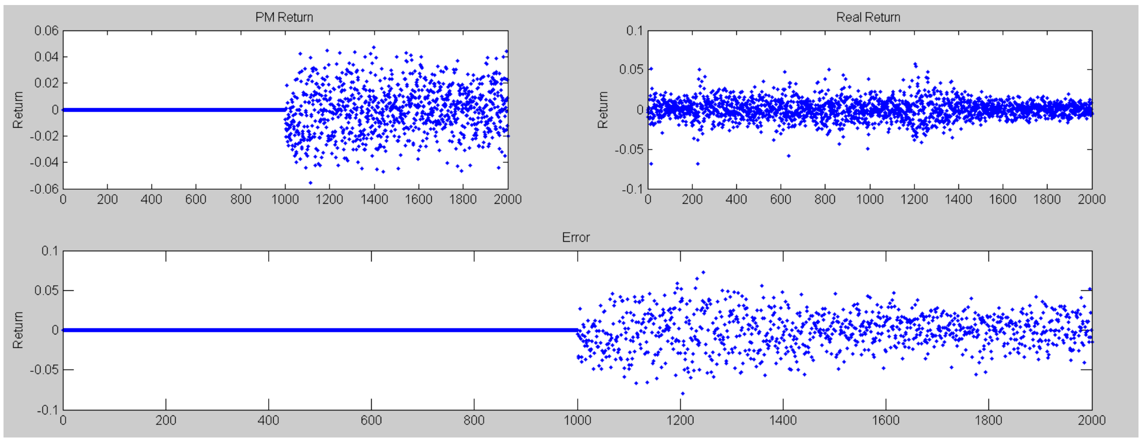


Figure 29: visual representation for the success of the PM "rolling forecast" method

$$
\begin{pmatrix}
up_{t-2} & down_{t-1} & equal_{t-1} & up_{t-1} \\
- & down_t & down_t & down_t \\
- & equal_t & equal_t & equal_t \\
- & up_t & up_t & up_t \\
equal_{t-2} & down_{t-1} & equal_{t-1} & up_{t-1} \\
- & down_t & down_t & down_t \\
- & equal_t & equal_t & equal_t \\
- & up_t & up_t & up_t \\
down_{t-2} & down_{t-1} & equal_{t-1} & up_{t-1} \\
- & down_t & down_t & down_t \\
- & equal_t & equal_t & equal_t \\
- & up_t & up_t & up_t
\end{pmatrix}
=
\begin{pmatrix}
33.33\% & 35.86\% & 33.51\% & 30.63\% \\
- & 41.98\% & 33.20\% & 41.89\% \\
- & 30.31\% & 28.86\% & 29.32\% \\
- & 27.71\% & 37.94\% & 28.79\% \\
33.33\% & 28.46\% & 33.74\% & 37.80\% \\
- & 30.31\% & 21.98\% & 29.27\% \\
- & 41.63\% & 34.38\% & 40.72\% \\
- & 28.06\% & 43.64\% & 30.01\% \\
33.33\% & 35.20\% & 33.83\% & 30.97\% \\
- & 33.40\% & 31.28\% & 39.47\% \\
- & 28.73\% & 34.88\% & 25.59\% \\
- & 37.86\% & 33.84\% & 34.94\%
\end{pmatrix}
$$

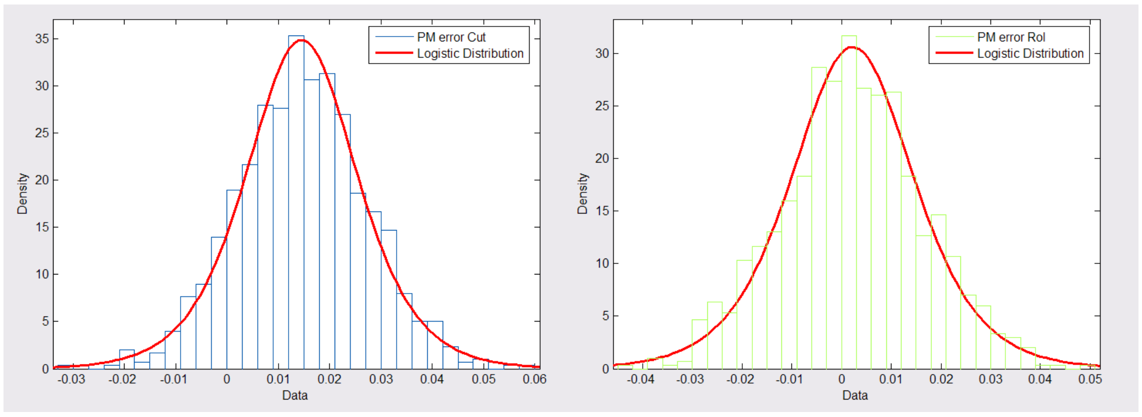
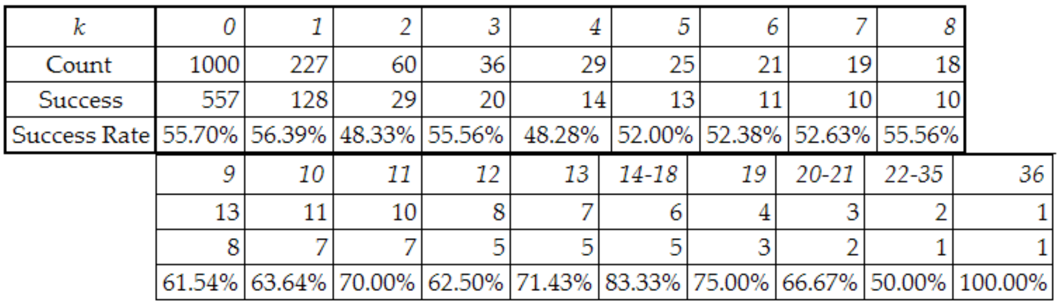


Figure 30: This Figure represents the error distribution using our two PCA models. Fit 1 and 3 are respectly the Logistic and Normal distribution.

6.5 U-NN

Below the results for our univariate Neural Network Method. Note here that the network itself has two input values S&P500 with a lag of one and S&P500 with a lag of two with the target being the S&P500 with no lag.
For the "cut and forecast" method we found the following results:

- RMSE = 1.58%.
- NRMSE = 99.98%.
- MAE = 0.55%.
- MAPE = 5.51E-05.
- SR = 55.70%.
- CI (Figure 31)



k	*0*	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*
Count	1000	227	60	36	29	25	21	19	18
Success	557	128	29	20	14	13	11	10	10
Success Rate	55.70%	56.39%	48.33%	55.56%	48.28%	52.00%	52.38%	52.63%	55.56%

9	*10*	*11*	*12*	*13*	*14-18*	*19*	*20-21*	*22-35*	*36*
13	11	10	8	7	6	4	3	2	1
8	7	7	5	5	5	3	2	1	1
61.54%	63.64%	70.00%	62.50%	71.43%	83.33%	75.00%	66.67%	50.00%	100.00%

Figure 31: Certainty Index for the NN "cut and forecast" method

- CIS = 68.42.%
- HSRI = 8.24.
- CI30 = 55.56%.

We can also have a visual representation of our results and error distribution, respectively in Figure 32 and Figure 35.

For the "rolling forecast" method we found the following results:

- RMSE = 2.08%.
- NRMSE = 99.98%.

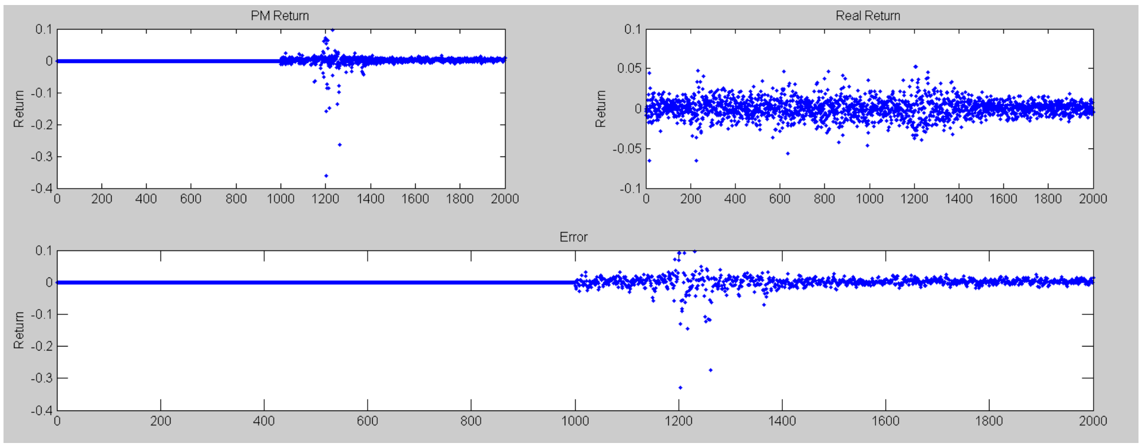
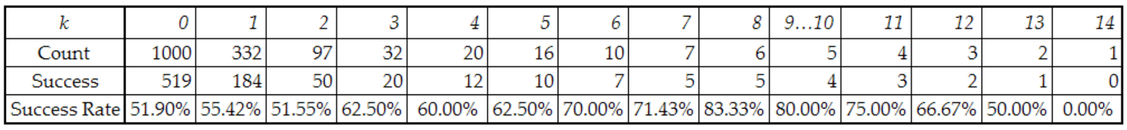


Figure 32: visual representation for the success of the NN "cut and forecast" method

- MAE = 0.88%.
- MAPE = 8.84E-05.
- SR = 51.90%.
- CI (Figure 33)



k	*0*	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9…10*	*11*	*12*	*13*	*14*
Count	1000	332	97	32	20	16	10	7	6	5	4	3	2	1
Success	519	184	50	20	12	10	7	5	5	4	3	2	1	0
Success Rate	51.90%	55.42%	51.55%	62.50%	60.00%	62.50%	70.00%	71.43%	83.33%	80.00%	75.00%	66.67%	50.00%	0.00%

Figure 33: Certainty Index for the NN "cut and forecast" method

- CIS = 53.85.%
- HSRI = 5.96.
- CI30 = 62.50%.

We can also have a visual representation of our results and error distribution, respectively in Figure 34 and Figure 35.

Figure 35 is the error distribution.
We can visualize the return of the S&P500 with respective two previous values in Figure 36.

The Figure 32 represents two screenshots of the return of the S&P500 with respect of its two previous values. Also note that from our work it is possible to get a visual representation of the value of the S&P500 at time t with respect of the value of the S&P500 at

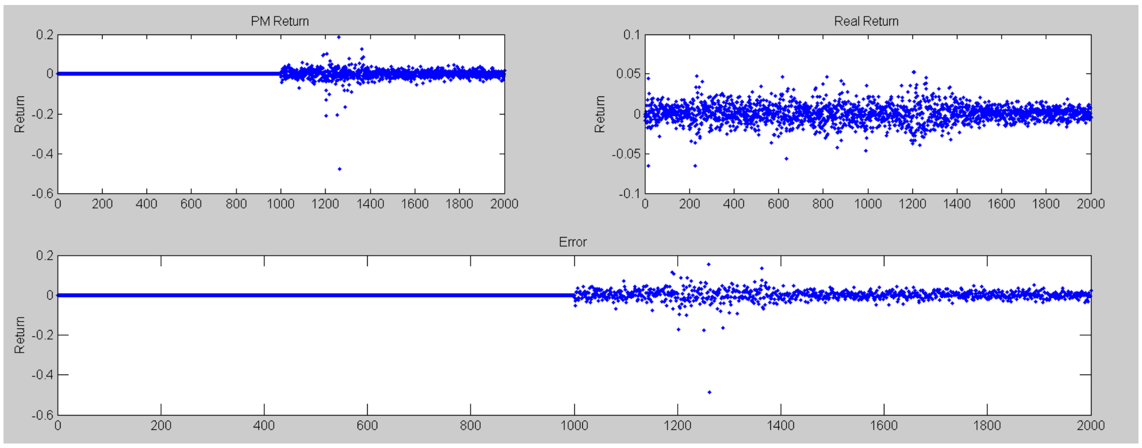


Figure 34: visual representation for the success of the NN ”rolling forecast” method

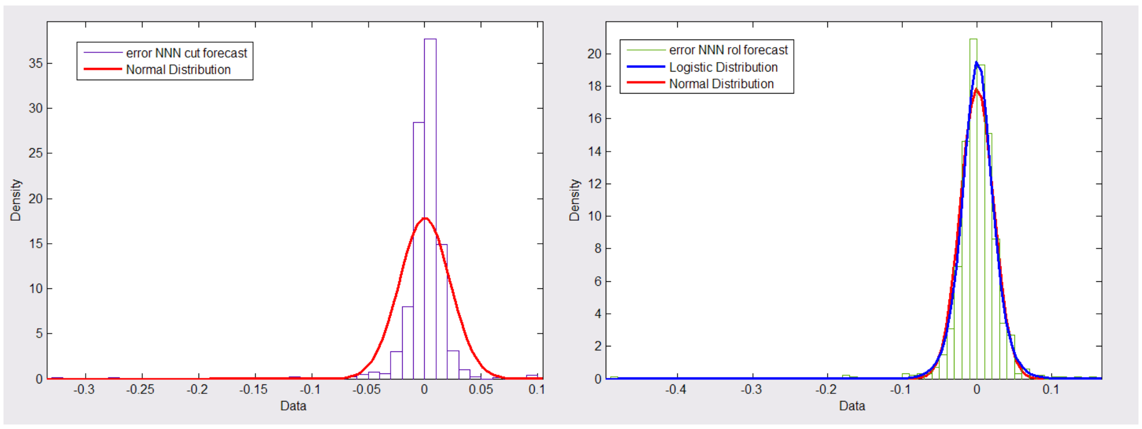


Figure 35: This Figure represents the error distribution using our two NN models.

time t-1 by rotating our previous shot screens as we can see in Figure 37.

As one can see from the last figure, the oval shaped form of the geometrical figure one gets from plotting the value of the S&P500 with respect of it previous value accounts for the movement of the S&P500 being close yet not completely random.

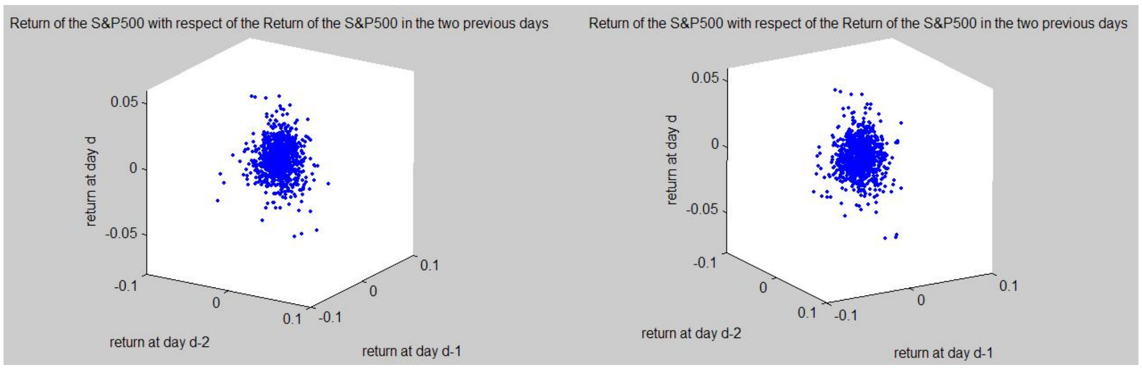


Figure 36: This Figure represents two screenshots of the return of the S&P500 with respect of its two previous values using a neural network method.

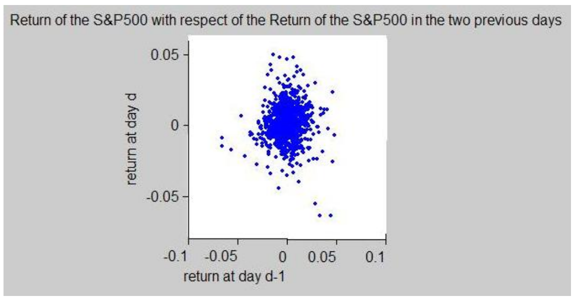
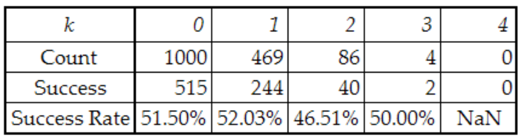


Figure 37: This Figure represents a screenshot of the return of the S&P500 with respect of its previou value using a neural network method.

6.6 U-GA

Below the results for the univariate Genetic Algorithm Design: For the "cut and forecast" method we found the following results:

- RMSE = 1.59%.
- NRMSE = 112.84%.
- MAE = 0.92%.
- MAPE = 9.18E-05.
- SR = 51.50%.
- CI (Figure 38)



k	*0*	*1*	*2*	*3*	*4*
Count	1000	469	86	4	0
Success	515	244	40	2	0
Success Rate	51.50%	52.03%	46.51%	50.00%	NaN

Figure 38: Certainty Index for the GA "cutting forecasting" method

- CIS = 75.00.%
- HSRI = 2.39.
- CI30 = 46.51%.

We can also have a visual representation of our results and error distribution, respectively in Figure 39 and Figure 42.

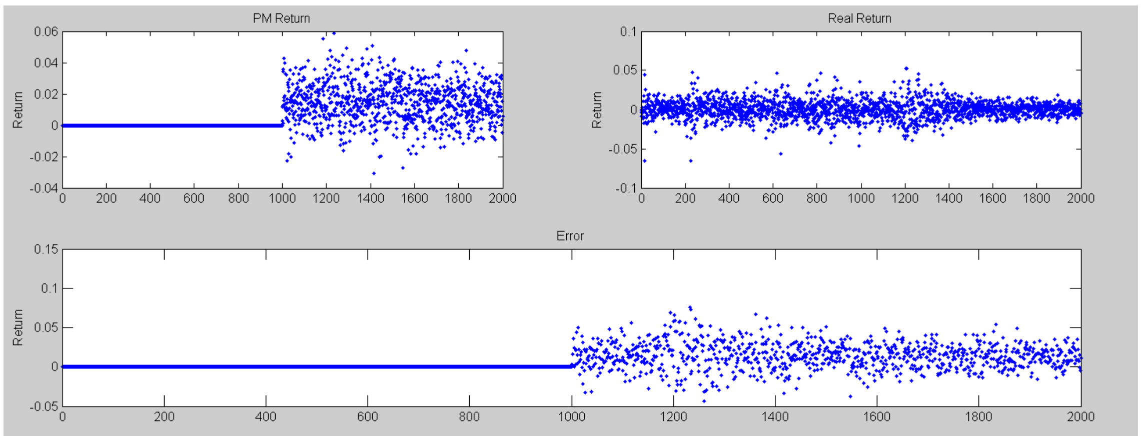


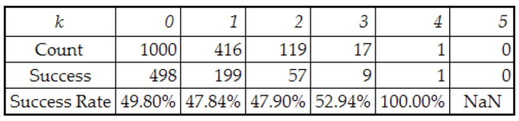
Figure 39: visual representation for the success of the PM "cut and forecast" method

The matrix result after 1000 rows of data is the following. Note that on the contrary to the algorithm previously described, the following is an average and not one run. We avoid doing many runs and taking the average results for the forecasted value for the testing part so as to avoid a cumbersome time complexity issue as well as trying to make the algorithm the most realistic way by allowing potential genetic drifts [11].
The following matrix represents the final population after 1000 rows of historical values, trained 100 times and averaged. The way we read this matrix is first by choosing on the first row what our input leading indicator is. This determines which column we will have to look at for our forecast, the maximum proportion in each column represents the likelihood of the corresponding leading indicator to reach that interval. For example if our g_8 function has translated the leading indicator $L^k I_t$ at time t to be of genetic code 101, we will look at the fifth column, to see that the highest rate is 26.16% at the second row, so that the forecasted value is 001.

$$\begin{array}{c|cccccccc} & pop_{000} & pop_{001} & pop_{010} & pop_{011} & pop_{100} & pop_{101} & pop_{110} & pop_{111} \\ \hline pop_{000} & 17.64\% & 08.90\% & 11.94\% & 10.52\% & 11.50\% & 07.11\% & 06.17\% & 17.53\% \\ pop_{001} & 20.61\% & 13.66\% & 11.31\% & 12.84\% & 07.87\% & 26.87\% & 20.31\% & 05.96\% \\ pop_{010} & 14.80\% & 15.29\% & 13.57\% & 12.99\% & 17.89\% & 06.32\% & 10.25\% & 17.31\% \\ pop_{011} & 09.42\% & 10.66\% & 08.35\% & 08.96\% & 08.02\% & 13.77\% & 16.37\% & 06.15\% \\ pop_{100} & 09.44\% & 11.46\% & 12.46\% & 10.48\% & 12.74\% & 05.60\% & 05.96\% & 18.42\% \\ pop_{101} & 11.97\% & 11.49\% & 11.47\% & 15.96\% & 09.57\% & 20.30\% & 17.09\% & 07.00\% \\ pop_{110} & 09.15\% & 18.58\% & 19.90\% & 15.63\% & 23.35\% & 05.62\% & 09.00\% & 21.98\% \\ pop_{111} & 06.98\% & 09.97\% & 11.00\% & 12.61\% & 09.06\% & 14.41\% & 14.85\% & 05.64\% \end{array}$$

For the "rolling forecast" method we found the following results:

- RMSE = 1.29%.
- NRMSE = 100.28%.
- MAE = 0.72%.
- MAPE = 7.19E-05.
- SR = 49.80%.
- CI (Figure 40)



k	0	1	2	3	4	5
Count	1000	416	119	17	1	0
Success	498	199	57	9	1	0
Success Rate	49.80%	47.84%	47.90%	52.94%	100.00%	NaN

Figure 40: Certainty Index for the PM "rolling forecast" method

- CIS = 80.00.%
- HSRI = 10.82.
- CI30 = 47.90%.

We can also have a visual representation of our results and error distribution, respectively in Figure 41 and Figure 42.

The matrix result after 1000 rows of data is the following.

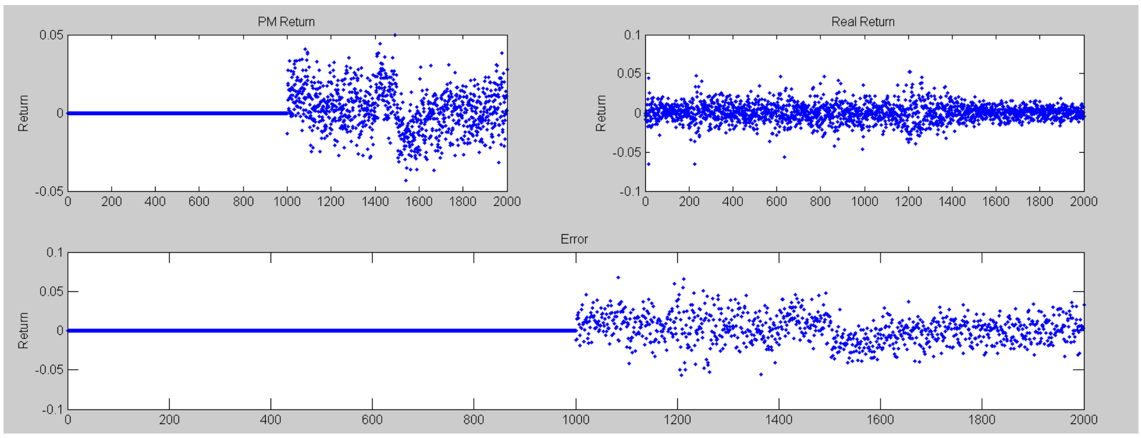


Figure 41: visual representation for the success of the GA "rolling forecast" method

$$\begin{array}{c}
\\
pop_{000}\\ pop_{001}\\ pop_{010}\\ pop_{011}\\ pop_{100}\\ pop_{101}\\ pop_{110}\\ pop_{111}
\end{array}
\begin{array}{c}
\begin{array}{cccccccc} pop_{000} & pop_{001} & pop_{010} & pop_{011} & pop_{100} & pop_{101} & pop_{110} & pop_{111} \end{array}\\
\begin{pmatrix}
13.56\% & 08.81\% & 13.72\% & 03.52\% & 06.75\% & 11.10\% & 09.30\% & 13.25\% \\
13.20\% & 12.64\% & 07.13\% & 14.14\% & 13.02\% & 21.07\% & 12.40\% & 09.18\% \\
17.32\% & 12.27\% & 18.00\% & 07.03\% & 08.51\% & 16.76\% & 18.33\% & 21.76\% \\
10.18\% & 11.03\% & 05.71\% & 15.48\% & 09.14\% & 14.48\% & 15.93\% & 10.31\% \\
10.80\% & 14.16\% & 23.98\% & 07.43\% & 12.67\% & 08.47\% & 12.21\% & 11.51\% \\
13.39\% & 19.60\% & 09.75\% & 26.08\% & 24.40\% & 13.25\% & 13.16\% & 08.09\% \\
13.07\% & 11.20\% & 16.15\% & 07.88\% & 11.90\% & 07.20\% & 10.44\% & 17.65\% \\
08.49\% & 10.28\% & 05.55\% & 18.44\% & 13.61\% & 07.70\% & 08.23\% & 08.24\%
\end{pmatrix}
\end{array}$$

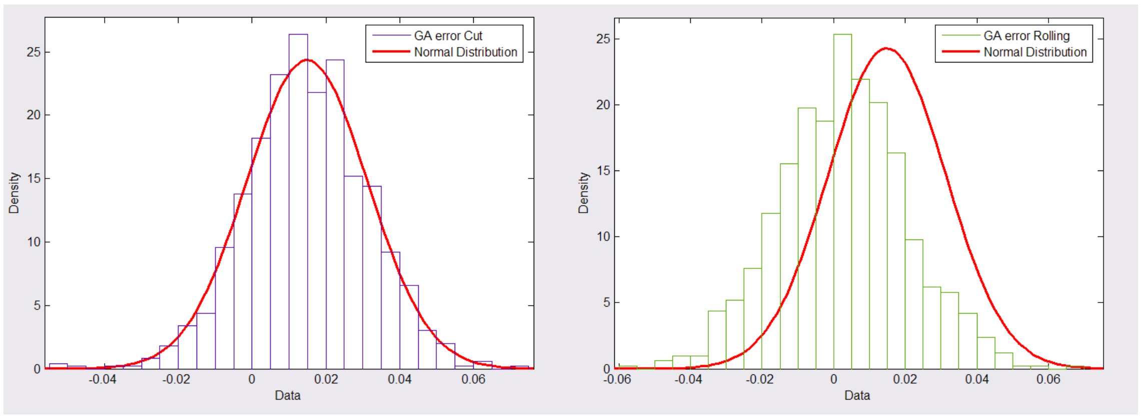


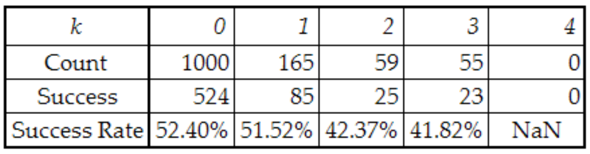
Figure 42: This Figure represents the error distribution using our two GA models.

Note that, even though this univariate GA method proved to be irrelevant for our approach, the method seems however best in practice for any type of forecasting that would involve behavioral finance. That is a model that evolve through time as investor are more and more aware of people's behavior and take action accordingly.

6.7 M-NNN

Using a multivariate naive neural network method we get the following results. Keep in mind that for this method the return was defined to be $R(t) = P(t) - P(t-1)$ rather than $R(t) = \frac{P(t)-P(t-1)}{P(t-1)}$. This simply to make things work.
For the "cut and forecast" method we found the following results:

- RMSE = 734.38%.
- NRMSE = 99.99%.
- MAE = 393.84%.
- MAPE = 3.94E-02.
- SR = 52.40%.
- CI (see next figure)



k	*0*	*1*	*2*	*3*	*4*
Count	1000	165	59	55	0
Success	524	85	25	23	0
Success Rate	52.40%	51.52%	42.37%	41.82%	NaN

Figure 43: Certainty Index for the NNN "cut and forecast" method

- CIS = 25.00.%
- HSRI = 2.65.
- CI30 = 41.82%.

We can also have a visual representation of our results and error distribution, respectively in Figure 44 and Figure 47.

For the "rolling forecast" method we found the following results:

- RMSE = 732.37%.
- NRMSE = 99.98%.
- MAE = 392.40%.

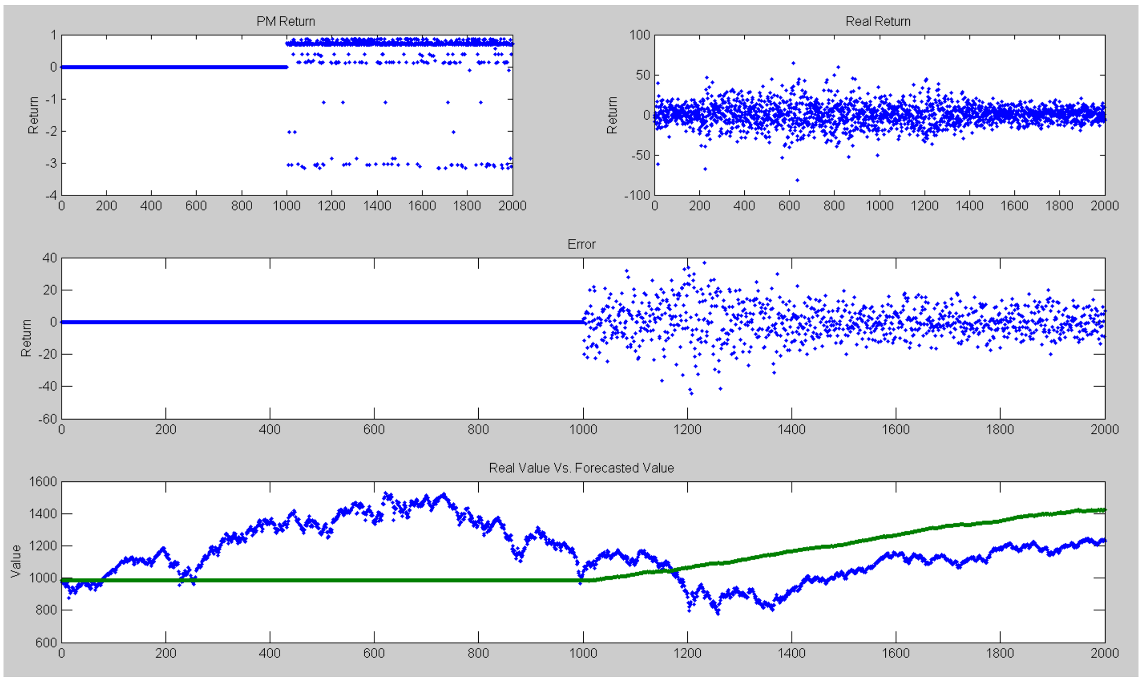


Figure 44: visual representation for the success of the NNN "cut and forecast" method

- MAPE = 3.92E-02.
- SR = 50.90%.
- CI (Figure 45)

k	*0*	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*
Count	1000	342	135	46	18	8	7	5	4	3	1
Success	509	181	74	30	13	3	2	1	1	1	0
Success Rate	50.90%	52.92%	54.81%	65.22%	72.22%	37.50%	28.57%	20.00%	25.00%	33.33%	0.00%

Figure 45: Certainty Index for the NNN "rolling forecast" method

- CIS = 70.00.%
- HSRI = 8.69.
- CI30 = 65.22%.

We can also have a visual representation of our results and error distribution, respectively in Figure 46 and Figure 47.

Figure 47 represents the error distribution.

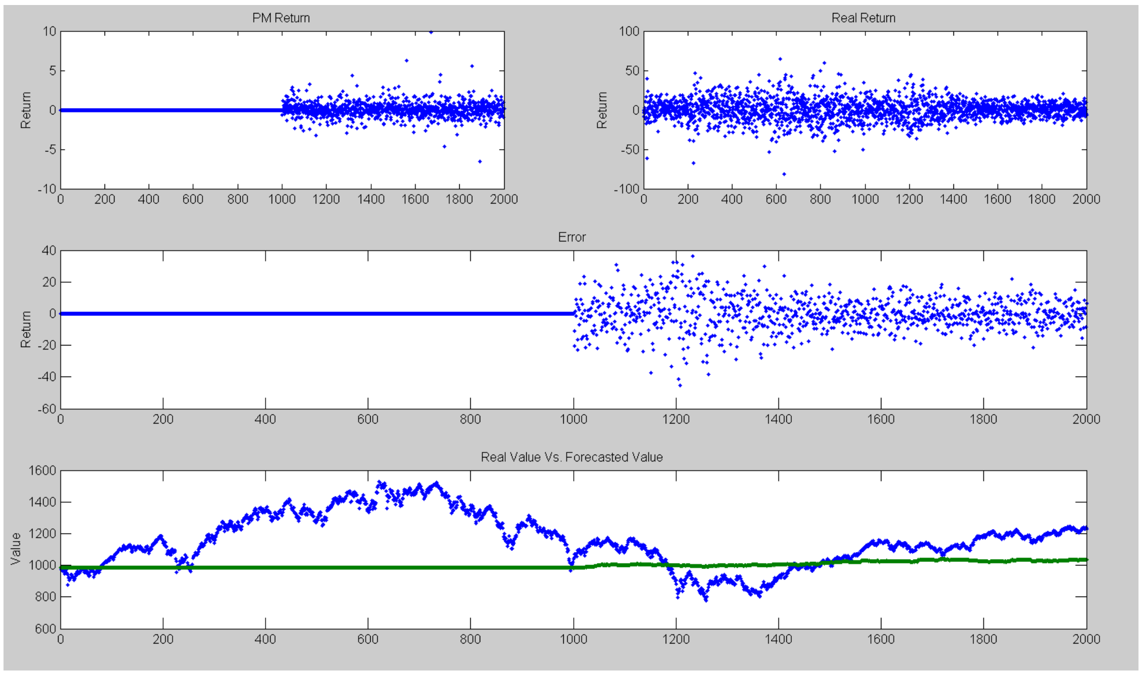


Figure 46: visual representation for the success of the NN "rolling forecast" method

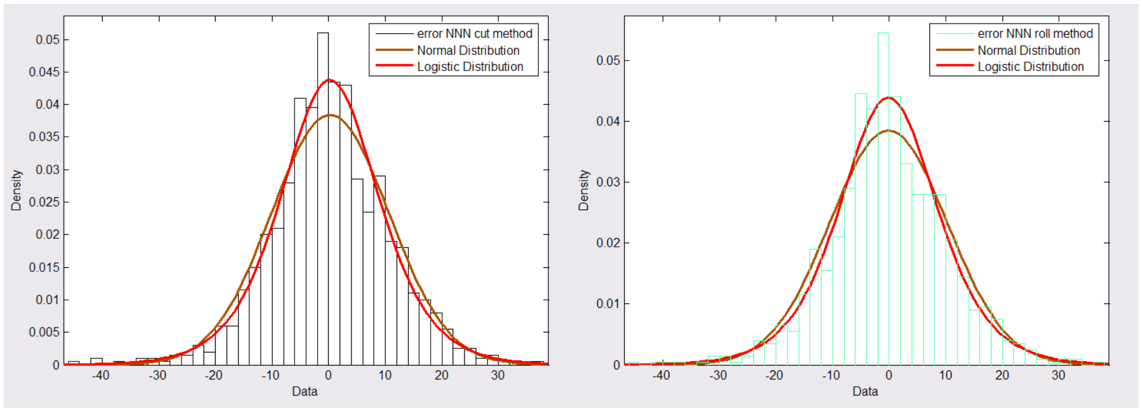
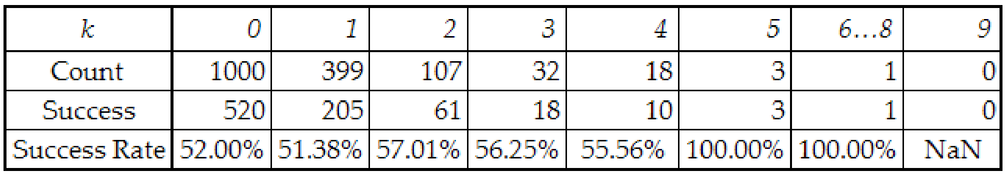


Figure 47: This Figure represents the error distribution using our two NNN models.

6.8 M-GNN

Using a multivariate Gately Neural Network method we get the following results.
For the ”cut and forecast” method we found the following results:

- RMSE = 0.81%.
- NRMSE = 100.79%.
- MAE = 0.42%.
- MAPE = 4.17E-05.
- SR = 52.00%.
- CI (Figure 48)



k	*0*	*1*	*2*	*3*	*4*	*5*	*6...8*	*9*
Count	1000	399	107	32	18	3	1	0
Success	520	205	61	18	10	3	1	0
Success Rate	52.00%	51.38%	57.01%	56.25%	55.56%	100.00%	100.00%	NaN

Figure 48: Certainty Index for the GNN ”cut and forecast” method

- CIS = 42.86.%
- HSRI = 0.00.
- CI30 = 56.25%.

We can also have a visual representation of our results and error distribution, respectively in Figure 49 and Figure 52.
For the ”rolling forecast” method we found the following results:

- RMSE = 1.88%.
- NRMSE = 100.26%.
- MAE = 1.09%.
- MAPE = 1.09E-04.
- SR = 49.30%.
- CI (Figure 50)

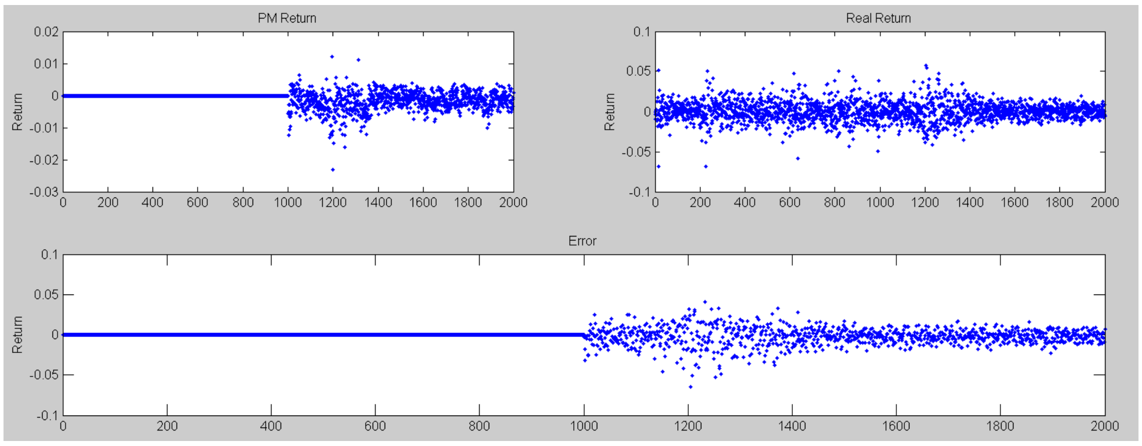
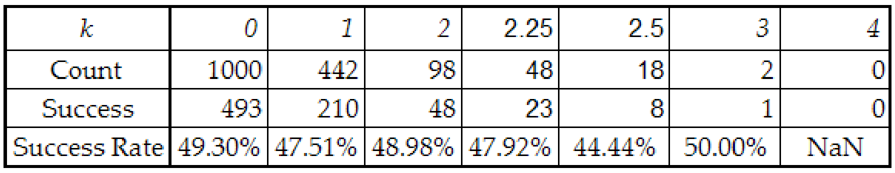


Figure 49: visual representation for the success of the GNN "cut and forecast" method



k	*0*	*1*	*2*	2.25	2.5	*3*	*4*
Count	1000	442	98	48	18	2	0
Success	493	210	48	23	8	1	0
Success Rate	49.30%	47.51%	48.98%	47.92%	44.44%	50.00%	NaN

Figure 50: Certainty Index for the GNN "rolling forecast" method

- CIS = 50.00.%
- HSRI = 1.07.
- CI30 = 47.92%.

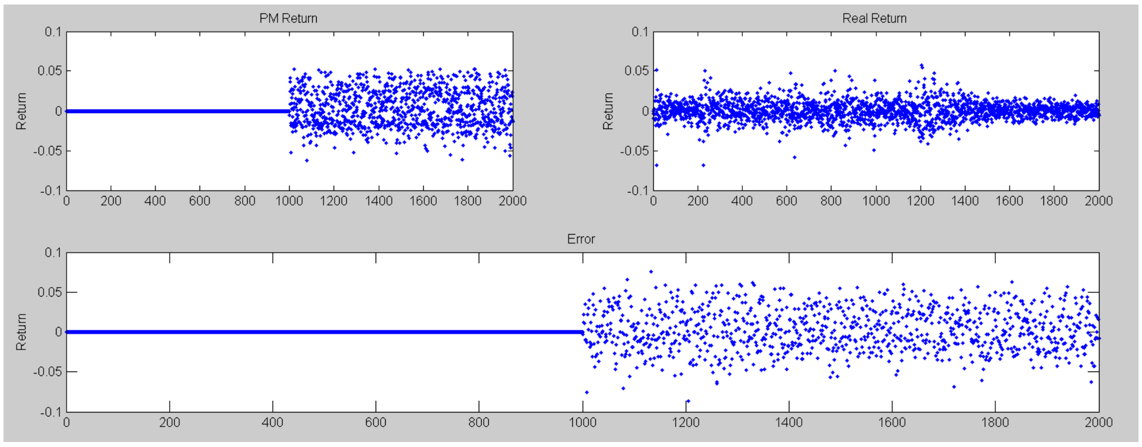


Figure 51: visual representation for the success of the GNN "rolling forecast" method

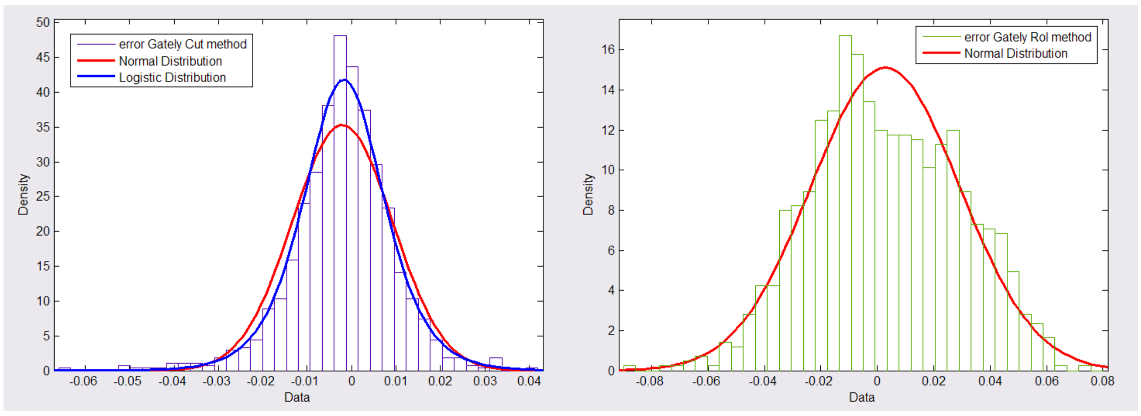
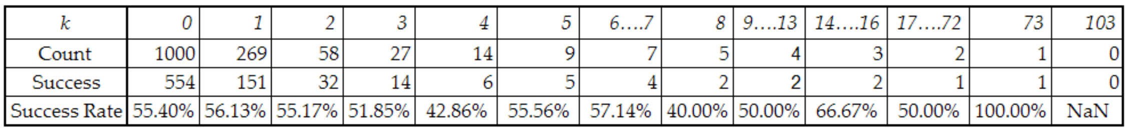


Figure 52: This Figure represents the error distribution using our two GNN models.

6.9 M-BNN

Using a multivariate Better Neural Network method we get the following results. For the "cut and forecast" method we found:

- RMSE = 0.95%.
- NRMSE = 99.98%.
- MAE = 0.40%.
- MAPE = 4.02E-05.
- SR = 55.70%.
- CI (see next figure)



k	0	1	2	3	4	5	6....7	8	9....13	14....16	17....72	73	103
Count	1000	269	58	27	14	9	7	5	4	3	2	1	0
Success	554	151	32	14	6	5	4	2	2	2	1	1	0
Success Rate	55.40%	56.13%	55.17%	51.85%	42.86%	55.56%	57.14%	40.00%	50.00%	66.67%	50.00%	100.00%	NaN

Figure 53: Certainty Index for the BNN "cut and forecast" method

- CIS = 50.00.%
- HSRI = 11.56.
- CI30 = 55.17%.

We can also have a visual representation of our results and error distribution, respectively in Figure 54 and Figure 57.

For the "rolling forecast" method we found the following results:

- RMSE = 0.83%.

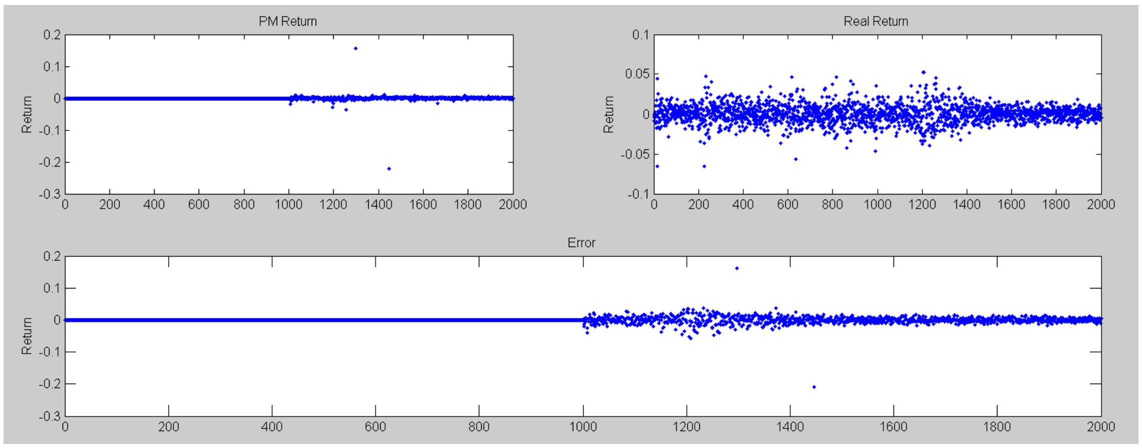
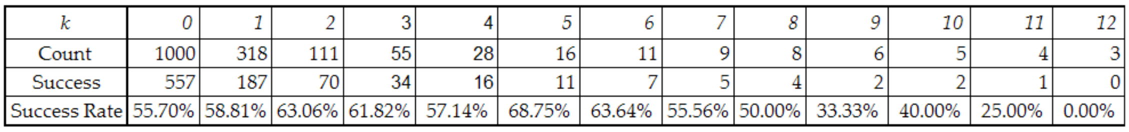


Figure 54: visual representation for the success of the BNN "cut and forecast" method

- NRMSE = 99.99%.
- MAE = 0.40%.
- MAPE = 4.05E-05.
- SR = 55.60%.
- CI (Figure 55)



k	*0*	*1*	*2*	3	4	*5*	*6*	*7*	*8*	*9*	*10*	*11*	*12*
Count	1000	318	111	55	28	16	11	9	8	6	5	4	3
Success	557	187	70	34	16	11	7	5	4	2	2	1	0
Success Rate	55.70%	58.81%	63.06%	61.82%	57.14%	68.75%	63.64%	55.56%	50.00%	33.33%	40.00%	25.00%	0.00%

Figure 55: Certainty Index for the BNN "rolling forecast" method

- CIS = 50.00.%
- HSRI = 1.07.
- CI30 = 47.92%.

We can also have a visual representation of our results and error distribution, respectively in Figure 56 and Figure 57.

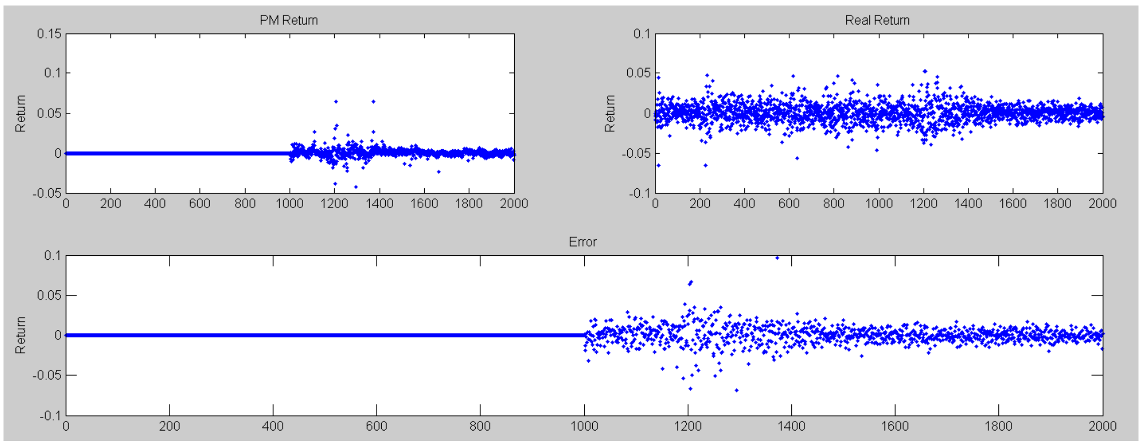


Figure 56: visual representation for the success of the BNN ”rolling forecast” method

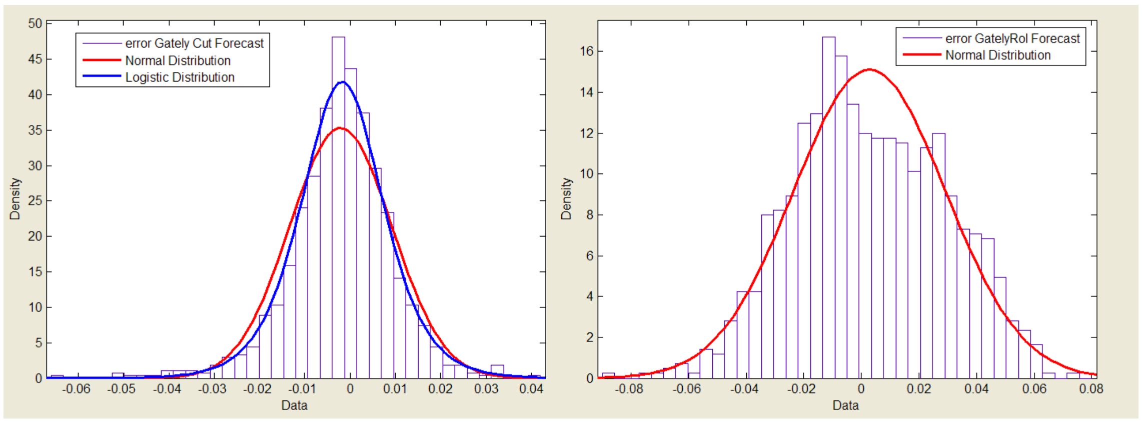
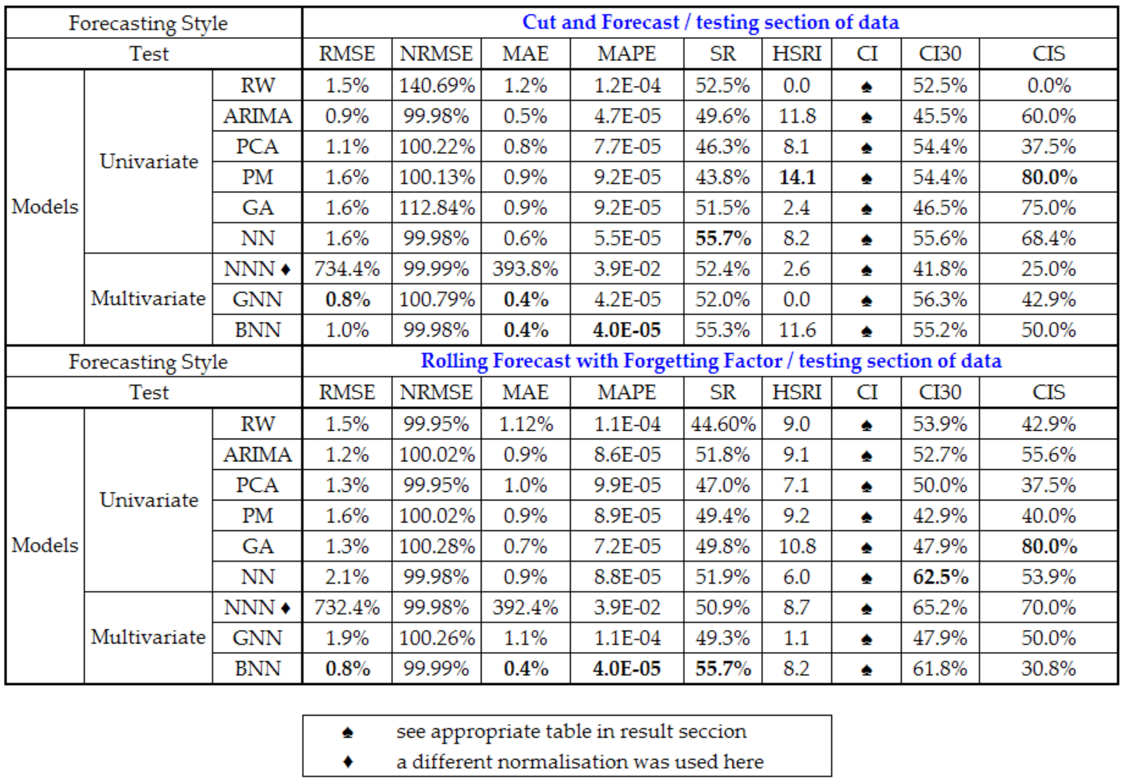


Figure 57: This Figure represents the error distribution using our two BNN models.

6.10 Summary

We can summarize our results in the table 58. Note that the M-NNN because of the fact that we have used a different normalization process was taken out of our future comparative results. The results for the RMSE and the NRMSE are summarized in



Forecasting Style			Cut and Forecast / testing section of data								
Test			RMSE	NRMSE	MAE	MAPE	SR	HSRI	CI	CI30	CIS
Models	Univariate	RW	1.5%	140.69%	1.2%	1.2E-04	52.5%	0.0	♠	52.5%	0.0%
		ARIMA	0.9%	99.98%	0.5%	4.7E-05	49.6%	11.8	♠	45.5%	60.0%
		PCA	1.1%	100.22%	0.8%	7.7E-05	46.3%	8.1	♠	54.4%	37.5%
		PM	1.6%	100.13%	0.9%	9.2E-05	43.8%	**14.1**	♠	54.4%	**80.0%**
		GA	1.6%	112.84%	0.9%	9.2E-05	51.5%	2.4	♠	46.5%	75.0%
		NN	1.6%	99.98%	0.6%	5.5E-05	**55.7%**	8.2	♠	55.6%	68.4%
	Multivariate	NNN ♦	734.4%	99.99%	393.8%	3.9E-02	52.4%	2.6	♠	41.8%	25.0%
		GNN	**0.8%**	100.79%	**0.4%**	4.2E-05	52.0%	0.0	♠	56.3%	42.9%
		BNN	1.0%	99.98%	**0.4%**	**4.0E-05**	55.3%	11.6	♠	55.2%	50.0%
Forecasting Style			Rolling Forecast with Forgetting Factor / testing section of data								
Test			RMSE	NRMSE	MAE	MAPE	SR	HSRI	CI	CI30	CIS
Models	Univariate	RW	1.5%	99.95%	1.12%	1.1E-04	44.60%	9.0	♠	53.9%	42.9%
		ARIMA	1.2%	100.02%	0.9%	8.6E-05	51.8%	9.1	♠	52.7%	55.6%
		PCA	1.3%	99.95%	1.0%	9.9E-05	47.0%	7.1	♠	50.0%	37.5%
		PM	1.6%	100.02%	0.9%	8.9E-05	49.4%	9.2	♠	42.9%	40.0%
		GA	1.3%	100.28%	0.7%	7.2E-05	49.8%	10.8	♠	47.9%	**80.0%**
		NN	2.1%	99.98%	0.9%	8.8E-05	51.9%	6.0	♠	**62.5%**	53.9%
	Multivariate	NNN ♦	732.4%	99.98%	392.4%	3.9E-02	50.9%	8.7	♠	65.2%	70.0%
		GNN	1.9%	100.26%	1.1%	1.1E-04	49.3%	1.1	♠	47.9%	50.0%
		BNN	**0.8%**	99.99%	**0.4%**	**4.0E-05**	**55.7%**	8.2	♠	61.8%	30.8%

♠	see appropriate table in result seccion
♦	a different normalisation was used here

Figure 58: Summary Table for the test results

Figure 59: First as one can see, the presence of the three of the Neural Network Designs in the first four place for the RMSE test, accounting for Neural Networks being the best know tool to model markets [24] because of their ability to model complex non linear models such as the markets. However, at the first glance we can see that amongst the same best four designs three are of the form "cut and forecast" rather than "rolling forecast" which does not verify the theory that older data is less relevnt than new data [22]. But one might speculate that if we look at models which RMSE is less than 1.5%, the split between the two type of forecasting model is equal. We may also add that our final design (BNN) which brings together everything we learnt and whose RMSE is the highest of all (equal to the GNNc) performs better with a rolling forecast. Finally note the presence in the elite of four models, the most simplistic model that is the ARIMA cut and forecast which is supposed to be our benchmark, a result that many people in the industry have oddly verified as well. The results of the MAE tests are mentionned in Figure 60. The MAE test confirms the RMSE test results. Once again three of the

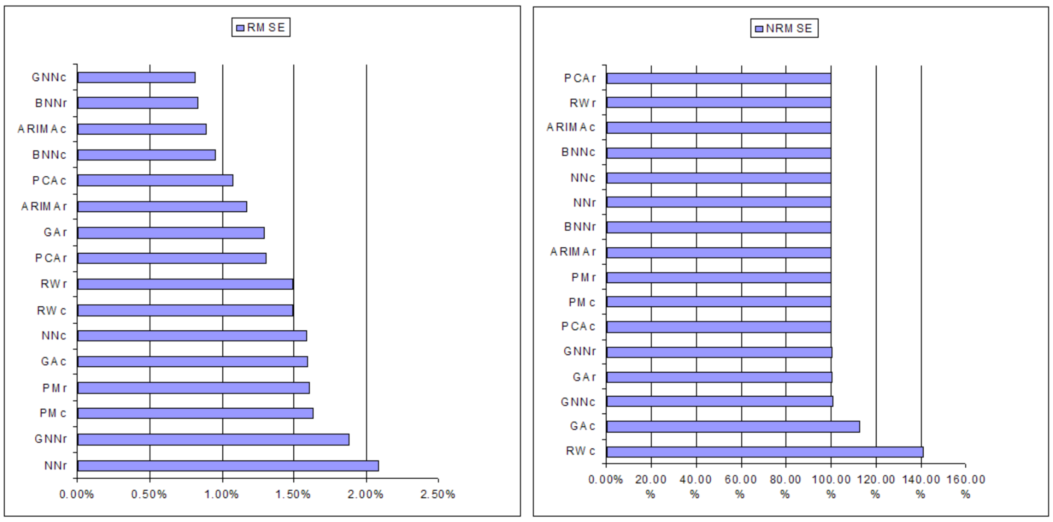


Figure 59: Visual Representation of the RMSE and the NRMSE for the test results

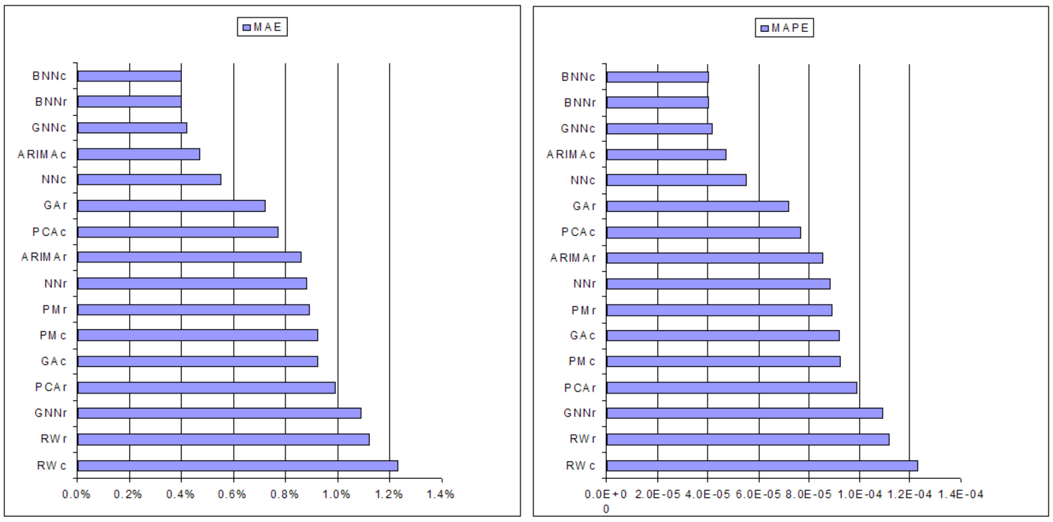


Figure 60: Visual Representation of the MAE and MAPE for the test results

four top designs are of NN type, with the BNN being this time clearly superior since the rolling and cut and forecast shares the top two positions. The ARIMAc is again in this group. Notice the very low ranking for the GNNr which was up to a couple of years ago still a reference. This is due that a good model's output becomes a contrarian indicator as the arbitrage opportunity is absorbed [34]. Now, we can look at the SR test results (Figure 61) . Remember that the SR model only looks at the success rate of the direction the forecast takes with respect of the real value. This time the five best models are of the type NN, with the ARIMAr coming at the sixth position. Note the poor results of our PM (probability models) which were a second order markov chain. These poor results compared to the other models might not be a coincidence. Here the model is better than the random walk in a theorethical term and simple enough for many people in the

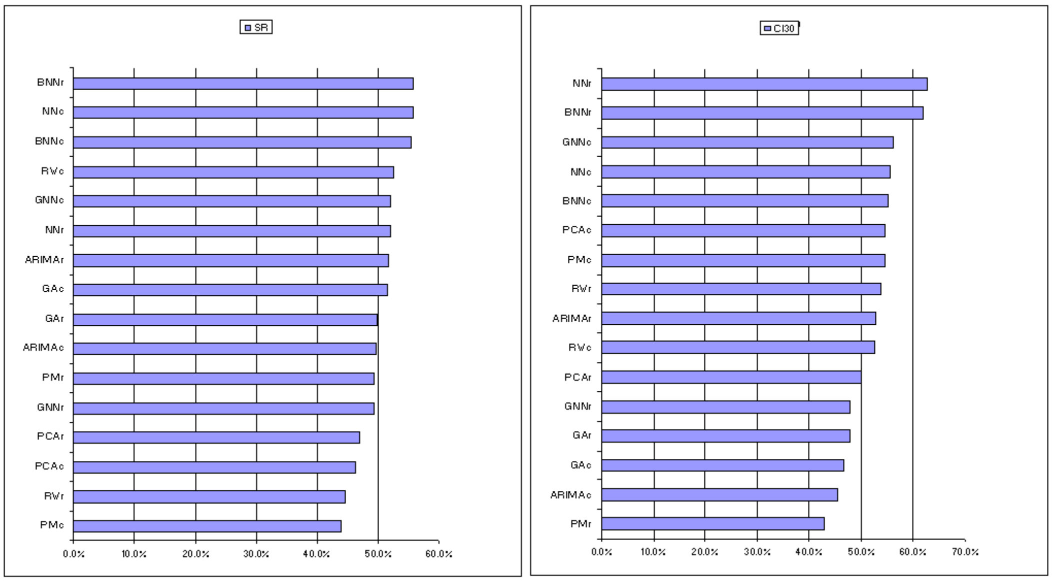


Figure 61: Visual Representation of the SR and CI30 for the test results

industry to perform but not complicated enough to really find an arbitrage opportunity which ultimatly performs poorly for the same argument we gave above [34].
As for the CI30, the test results are summarized in Figure 61. Remember that CI30 shows the SR of a model as it is increasingly sure of its future movement (up or down). The CI30 is a legitimate test because thirty is big enough for the results to have a statistical meaning (see the statistical methods chapter). The test confirms what we have seen with the RMSE and the MAE, an obvious domination from the NN models with our BNNr being once again in the top two positions. We have calculated a 95% confidence that the success rate of our design will fall in between 48.48% and 75.15% which can allow us to feel comfortable this BNNr model beats our ARIMAc benchmark whose SR mean is just below 50%.
We would now look at the CIS and HSRI results that were initially designed in case the models ranking change too drastically depending on the tests which is actually not the case so we will not look at them.

7 Conclusion

7.1 Summary of Aims & Methods

In this MSc dissertation, some financial forecasting methods were studied. In doing so, we looked at several univariate methods: ARIMA (autoregressive moving average), PCA (principal component analysis), PM (probabilistic modeling, second order markov chains), GA (genetic algorithms) and NN (neural networks).
It was concluded that in order to improve the project into a multivariate design, we had to choose the model that would be the best for this purpose. Neural networks were chosen for the multivariate purpose and, as a result, three new multivariate algorithms were created: NNN (naive neural network), GNN (Gately neural network) and BNN (better neural network).
Another key factor of the presented design was to decide what inputs to be used, that is, what set of leading indicators and optimum lag to use with respect to the target value (S&P500). In doing so, we had to look both at what the literature said about these leading indicators and also to the complexity issues involved by trying to find the best lag for each potential leading indicator.
We also improved each of our designs by processing the data through noise reduction, forgetting factor and normalization techniques as well as by studying two types of training techniques that are the ”cut & forecast” and the ”rolling forecast” (ARIMAc, ARIMAr, etc). The results were tested and compared by using few traditional and non-traditional statistical techniques.

7.2 Summary of the Results

The obtained results are encouraging The final design performed better than the random walk and the benchmark, and outperformed Gately design, a reference in the industry. We can mention a general use of neural network type design for the different models accounting for their ability to beat any type of linear model such as the ARIMA or PCA model. It also beats discrete models such as the PM or the GA models. All the main statistical tests defined, such as the RMSE, the MAE, the SR or the CI30 agree with these claims. We can also note that, with respect to the two best designs, the BNNr performs slightly better than BNNc, which supports the supposition that stock markets have usually short term memory. The BNNr uses a resilient backpropagation algorithm with the following leading indicators with shift between 0 and 5: Dow Jones, News, COT S&P500 stock index futures Traders Net, COT S&P500 stock index futures non Commercial Traders Net, Volume S&P500, Dow Jones Transportation, Dow Jones Utilities, AMEX Oil Index, Gold and Silver, COT Mini S&P500 Non Commercial Traders

Net, COT Mini S&P500 Commercial Traders Net, S&P500). It also uses a forgetting factor through the exponential decay technique as well as a noise reducer of the exponential moving average type.

7.3 Directions for future research

The potential for further research are immense for several reasons:

- There are many leading indicators and thus many possible combinations of them. I have personally not tried all the combinations. I was also not able to find the data for few of them that I wanted to try.
- The lag for each of these leading indicator is not optimum, and the design should rigorously check the combination of leading indicators at every time step, which was not possible to be done in depth because of time complexity issues.
- There are many different architectures and training algorithms for neural networks to be tested, and thus many different combinations of leading indicators and architectures. I have tried a resilient backpropagation which produced good results, but there is still room for improvement. Also, there are other types of machine learning techniques which could be considered for the multivariate design. For example, the theory behind genetic algorithms could fit perfectly to the way the stock market evolves. We have only seen a glimpse of this potential within our univariate model.
- The design should be checked at different time frames to see if our design is able to capture all the different behaviors the stock market may have, and also to see if our certainty index makes sense on these different intervals, which may be limited because of the lack of historical data of some of our leading indicators.
- We have applied the described techniques to the S&P 500. It could be interesting to see if the method would work with other indexes (in Europe and Asia for example) too.

8 Appendix

8.1 Matlab Codes for U-Random Walk

```
% function that trains and uses the random walk cut and forecast model

function M = randomWalkcut(InputLeadingIndicator,TargetIndicator);

% data manipulation
InputLeadingIndicator = flipud(InputLeadingIndicator);
TargetIndicator = flipud(TargetIndicator);

% capturing the returns
deltaI(1,1) = 0;
deltaT(1,1) = 0;
for(i=2:1:2000)
    deltaT(i,1) = (TargetIndicator(i,1)-TargetIndicator(i-1,1))/TargetIndicator(i-1,1);
    deltaI(i,1) = (InputLeadingIndicator(i,1)-InputLeadingIndicator(i-1,1))/InputLeadingIndicator(i-1,1);
end

% making and testing the forecast
forecat(1000,1) = deltaT(999,1);
success = 0;
cumulatifeError(1000,1) = TargetIndicator(999,1) - TargetIndicator(998,1);
for(i=1000:1:2000)
    forecast(i,1) = deltaT(999,1);
    cumulatifeError(i,1) = (forecast(i,1)+1) * TargetIndicator(999,1) + cumulatifeError(i-1);
    if (forecast(i,1) * deltaT(i,1) > 0)
        success = success + 1;
    end
    error(i,1) = forecast(i,1) - deltaT(i,1);
end

% printing the test results
successRate = success/1000
rmse = RMSE(error(1001:2000,1))
nrmse = NRMSE(error(1001:2000,1))
mae = MAE(error(1001:2000,1))
mape = MAPE(error(1001:2000,1))

% data manipulation for cumulative error
data = flipud(TargetIndicator);
datatemp = (data(1:2000));
Yt_temp(1,1) = datatemp(1,1);
for (i=2:1:2000)
    if(i<1000)
        Yt_temp(i,1) = TargetIndicator(999,1);
    else
        Yt_temp(i,1) =  cumulatifeError(i,1);
    end
end
x_1 = 1:2000;

% Plotting the results
figure
subplot(3,2,1)
plot(x_1, forecast, '.');
title('PM Return')
```

```
ylabel('Return')
subplot(3,2,2)
plot(x_1, deltaT(1:2000,1), '.');
title('Real Return')
ylabel('Return')
subplot(3,2,3:4)
plot(x_1, error, '.');
title('Error')
ylabel('Return')
TargetIndicator = flipud(TargetIndicator);
subplot(3,2,5:6)
plot(x_1, data(1:2000),'.', x_1, Yt_temp, '.');
title('Real Value Vs. Cumulative Error Value')
ylabel('Value')

M=[forecast(1001:2000,1) deltaT(1001:2000,1) error(1001:2000,1)];
end
```

```
% function that trains and uses the random walk rolling forecast model

function M = randomWalkRol(InputLeadingIndicator,TargetIndicator);

% data manipulation
InputLeadingIndicator = flipud(InputLeadingIndicator);
TargetIndicator = flipud(TargetIndicator);

% capturing the returns
deltaI(1,1) = 0;
deltaT(1,1) = 0;
for(i=2:1:2000)
    deltaT(i,1) = (TargetIndicator(i,1)-TargetIndicator(i-1,1))/TargetIndicator(i-1,1);
    deltaI(i,1) = (InputLeadingIndicator(i,1)-InputLeadingIndicator(i-1,1))/InputLeadingIndicator(i-1,1);
end
forecat(1000,1) = deltaT(999,1);
success = 0;

% making and testing the forecast
for(i=1001:1:2000)
    forecast(i,1) = deltaT(i-1,1);
    realForecast(i,1) = (forecast(i,1)+1)*TargetIndicator(i-1,1);
    if (forecast(i,1) * deltaT(i,1) > 0)
        success = success + 1;
    end
    error(i,1) = forecast(i,1) - deltaT(i,1);
end

% printing the test results
successRate = success/1000
rmse = RMSE(error(1001:2000,1))
nrmse = NRMSE(error(1001:2000,1))
mae = MAE(error(1001:2000,1))
mape = MAPE(error(1001:2000,1))

% data manipulation for cumulative error
data = flipud(TargetIndicator);
```

```
datatemp = (data(1:2000));
Yt_temp(1,1) = datatemp(1,1)
for (i=2:1:2000)
    if(i<999)
        Yt_temp(i,1) = TargetIndicator(999,1);
    else
    Yt_temp(i,1) =  realForecast(i,1);
    end
end

x_1 = 1:2000;
% Plotting the results
figure
subplot(3,2,1)
plot(x_1, forecast, '.');
title('PM Return')
ylabel('Return')
subplot(3,2,2)
plot(x_1, deltaT(1:2000,1), '.');
title('Real Return')
ylabel('Return')
subplot(3,2,3:4)
plot(x_1, error, '.');
title('Error')
ylabel('Return')
TargetIndicator = flipud(TargetIndicator);
subplot(3,2,5:6)
plot(x_1, data(1:2000),'.', x_1, Yt_temp, '.');
title('Real Value Vs. Cumulative Error Value')
ylabel('Value')

M=[forecast(1001:2000,1) deltaT(1001:2000,1) error(1001:2000,1)];
end
```

8.2 Matlab Codes for U-ARIMA

```
% function that uses ARIMA to make a forecast
% Test it using the following: ARIMAtestCutting(S_P_500)

function M = ARIMAtestCutting(LInd)
data = LInd;
dataReal = data(1000:2000);
period = 2;
LInd = flipud(LInd);
LInd = ShiftMatrix(LInd, 1, length(LInd)-1,1);
LInd = reduceNoise(LInd, 0.95);
trainingPeriod = period;
Yt(1,1) = 0;
deltaI(1,1) = 0;
ARtemp (1,1) = 0;
meanSP = mean(LInd);
sum = 0;
for(j=2:1:length(LInd))
    deltaI (j,1) = (LInd(j,1)-LInd(j-1,1))/LInd(j-1,1);
    if (deltaI (j,1)>0)
        sum = sum+1;
    end
    ARtemp (j,1) = 0;
end
Yt(1,1)=0;
for (i=2:1:1000)
    deltaItemp = deltaI';
    ARtemptemp = ARtemp';
    AR = CutBelow(deltaItemp(1,i-1:i), 1, period);   % coefficients for the AR part of the ARIMA
    MA = CutBelow(deltaItemp(1,i-1:i), 1, period);   % coefficients for the MA part of the ARIMA
    coeff_AR(i,1:2) = garchar(AR, MA, period);
    coeff_MA(i,1:2) = garchma(AR, MA, period);
    Yt(i,1)=0;
end
coeff_AR_av = mean(coeff_AR);
coeff_MA_av = mean(coeff_MA);
for (i=1001:1:2000)
    LInd = CutAbove(LInd, period+1,1);
    for (k=1:1:period)
        Yt(i,1) = Yt(i-1,1) + coeff_AR_av(1,k) * deltaI(k,1) + coeff_MA_av(1,k) * deltaI(k,1);
    end
    trainingPeriod = trainingPeriod +1;
    realReturn(i,1) = deltaI(trainingPeriod+1,1);
end
err(1,1) = Yt(1,1)-realReturn(1,1);
success = 0;
for (i=2:1:2000)
    forecast(i,1) = Yt(i,1)-Yt(i-1,1);
    err(i,1) = forecast(i,1)-realReturn(i,1);
    if (forecast(i,1)*realReturn(i,1) > 0)
        success = success +1;
    end
end
successRate = success/1000
rmse = RMSE(err(1001:2000,1))
```

```
nrmse = NRMSE(err(1001:2000,1))
mae = MAE(err(1001:2000,1))
mape = MAPE(err(1001:2000,1))
datatemp = flipud(data(1:2000));
Yt_temp(1,1) = datatemp(1,1)
for (i=2:1:2000)
   Yt_temp(i,1) =  Yt_temp(i-1,1)+Yt(i,1);
end
success = sum/j;
x_1 = 1:2000;
figure
subplot(3,2,1)
plot(x_1, forecast, '.');
title('ARIMA(0,1,0) Return')
ylabel('Return')
subplot(3,2,2)
plot(x_1, deltaI(1:2000,1), '.');
title('Real Return')
ylabel('Return')
subplot(3,2,3:4)
plot(x_1, err, '.');
title('Error')
ylabel('Return')
size(Yt);
size(data);
subplot(3,2,5:6)
plot(x_1, flipud(data(1:2000)),'.', x_1, Yt_temp, '.');
title('Real Value Vs. Forecasted Value')
ylabel('Value')
M = [forecast(1000:2000,1) deltaI(1000:2000,1) err(1000:2000,1)];
end
```

```
% function that uses ARIMA to make a forecast
% Test it using the following: ARIMAtestRolling(S_P_500)

function M = ARIMAtestRolling(LInd)
data = LInd;
dataReal = data(1000:2000);
period = 2;
LInd = flipud(LInd);
LInd = ShiftMatrix(LInd, 1, length(LInd)-1,1);
LInd = reduceNoise(LInd, 0.95);
trainingPeriod = period;
Yt(1,1) = 0;
deltaI(1,1) = 0;
ARtemp (1,1) = 0;
meanSP = mean(LInd);
sum = 0;
for(j=2:1:length(LInd))
    deltaI (j,1) = (LInd(j,1)-LInd(j-1,1))/LInd(j-1,1);
    if (deltaI (j,1)>0)
        sum = sum+1;
    end
    ARtemp (j,1) = 0;
end
```

```
Yt(1,1)=0;
for (i=2:1:2000)
    deltaItemp = deltaI';
    ARtemptemp = ARtemp';
    AR = CutBelow(deltaItemp(1,i-1:i), 1, period);   % coefficients for the AR part of the ARIMA
    MA = CutBelow(deltaItemp(1,i-1:i), 1, period);   % coefficients for the MA part of the ARIMA
    coeff_AR(i,1:2) = garchar(AR, MA, period);
    coeff_MA(i,1:2) = garchma(AR, MA, period);
    LInd = CutAbove(LInd, period+1,1);
    coeff_AR_temp = coeff_AR(i,1:2);
    coeff_MA_temp = coeff_MA(i,1:2);
    for (k=1:1:period)
        Yt(i,1) = Yt(i-1,1) + coeff_AR_temp(1,k) * deltaI(i,1) + coeff_MA_temp(1,k) * deltaI(i,1);
    end
    trainingPeriod = trainingPeriod +1;
    realReturn(i,1) = deltaI(trainingPeriod+1,1);
end
err(1,1) = Yt(1,1)-realReturn(1,1);
success = 0;
for (i=1000:1:2000)
    forecast(i,1) = Yt(i,1)-Yt(i-1,1);
    err(i,1) = forecast(i,1)-realReturn(i,1);
    if (forecast(i,1)*realReturn(i,1) > 0)
        success = success +1;
    end
end
successRate = success/1000
rmse = RMSE(err(1001:2000,1))
nrmse = NRMSE(err(1001:2000,1))
mae = MAE(err(1001:2000,1))
mape = MAPE(err(1001:2000,1))
datatemp = flipud(data(1:2000));
Yt_temp(1,1) = datatemp(1,1);
for (i=1:1:2000)
    if (i<1000)
        Yt_temp(i,1)= Yt_temp(1,1);
    else
        Yt_temp(i,1) =  Yt_temp(i-1,1)+Yt(i,1);
    end
end
siZe = size(Yt_temp)
x_1 = 1:2000;
figure
subplot(3,2,1)
plot(x_1, forecast, '.');
title('ARIMA(0,1,0) Return')
ylabel('Return')
subplot(3,2,2)
plot(x_1, deltaI(1:2000,1), '.');
title('Real Return')
ylabel('Return')
subplot(3,2,3:4)
plot(x_1, err, '.');
title('Error')
ylabel('Return')
subplot(3,2,5:6)
plot(x_1, flipud(data(1:2000)),'.', x_1, Yt_temp, '.');
```

```
title('Real Value Vs. Forecasted Value')
ylabel('Value')
M = [forecast(1000:2000,1) deltaI(1000:2000,1) err(1000:2000,1)];
end
```

8.3 Matlab Codes for the U-PM Model (second order markov chain)

```
% This function aims at training a second order markov chain:
% run this function using an initial scales = 0.485&0.53,and dataMean = 0.1

function M = trainMarkovWithWeight(L1,scale1,scale2,dataMean)

L1 = reduceNoise(flipud(L1), 0.95);
% the constant 0.95 creates a nice noise less curve that still picks the general
% movement of the S&P 500 and was suggested by the litterature (try 0.01
% next time)

perChange(1,1) = dataMean;
perChange(2,1) = dataMean;
flag = 0;

while (flag < 1)

    Sum = 0;
    %------------------------
    smallerThanMean=0;
    smallerThanMeanSTM=0;
    smallerThanMeanSTMstm=0;
    smallerThanMeanSTMetm=0;
    smallerThanMeanSTMbtm=0;
    smallerThanMeanETM=0;
    smallerThanMeanETMstm=0;
    smallerThanMeanETMetm=0;
    smallerThanMeanETMbtm=0;
    smallerThanMeanBTM=0;
    smallerThanMeanBTMstm=0;
    smallerThanMeanBTMetm=0;
    smallerThanMeanBTMbtm=0;
    %------------------------
    equalToMean=0;
    equalToMeanSTM=0;
    equalToMeanSTMstm=0;
    equalToMeanSTMetm=0;
    equalToMeanSTMbtm=0;
    equalToMeanETM=0;
    equalToMeanETMstm=0;
    equalToMeanETMetm=0;
    equalToMeanETMbtm=0;
    equalToMeanBTM=0;
    equalToMeanBTMstm=0;
    equalToMeanBTMetm=0;
    equalToMeanBTMbtm=0;
    %------------------------
    biggerThanMean=0;

    biggerThanMeanSTM=0;
    biggerThanMeanSTMstm=0;
    biggerThanMeanSTMetm=0;
    biggerThanMeanSTMbtm=0;
    biggerThanMeanETM=0;
    biggerThanMeanETMstm=0;
```

```
biggerThanMeanETMetm=0;
biggerThanMeanETMbtm=0;
biggerThanMeanBTM=0;
biggerThanMeanBTMstm=0;
biggerThanMeanBTMetm=0;
biggerThanMeanBTMbtm=0;
%------------------------
weightForB = 1.1;
% initially start as 1.1 as opposed to 1 because log (1) =0
% which would not be good for a weighted average
weightForE = 1.1;
weightForS = 1.1;
marKovLength = length(L1);

for(i=2:1:marKovLength-1)
    perChange(i,1)=(L1(i,1)-L1(i-1,1))/L1(i-1,1);
    perChange(i+1,1)=(L1(i+1,1)-L1(i,1))/L1(i,1);
    Sum = Sum + abs(perChange(i+1,1));
    if (perChange(i+1,1) < - dataMean * scale2)
        if (perChange(i,1) < - dataMean * scale2)
            if (perChange(i-1,1) < - dataMean * scale2)
                smallerThanMeanSTMstm = smallerThanMeanSTMstm+log(weightForS);
                weightForS = weightForS+1;
            elseif(perChange(i-1,1) > + dataMean * scale1)
                smallerThanMeanSTMbtm = smallerThanMeanSTMbtm+log(weightForB);
                weightForB = weightForB+1;
            else
                smallerThanMeanSTMetm = smallerThanMeanSTMetm+log(weightForE);
                weightForE = weightForE+1;
            end
        elseif(perChange(i,1) > + dataMean * scale1)
            if(perChange(i-1,1) < - dataMean * scale2)
                smallerThanMeanBTMstm = smallerThanMeanBTMstm+log(weightForS);
                weightForS = weightForS+1;
            elseif(perChange(i-1,1) > + dataMean * scale1)
                smallerThanMeanBTMbtm = smallerThanMeanBTMbtm+log(weightForB);
                weightForB = weightForB+1;
            else
                smallerThanMeanBTMetm = smallerThanMeanBTMetm+log(weightForE);
                weightForE = weightForE+1;
            end
        else
            if(perChange(i-1,1) < - dataMean * scale2)
                smallerThanMeanETMstm = smallerThanMeanETMstm+log(weightForS);
                weightForS = weightForS+1;
            elseif(perChange(i-1,1) > + dataMean * scale1)
                smallerThanMeanETMbtm = smallerThanMeanETMbtm+log(weightForB);
                weightForB = weightForB+1;
            else
                smallerThanMeanETMetm = smallerThanMeanETMetm+log(weightForE);
                weightForE = weightForE+1;
            end
        end
        %--------------------------------
    elseif(perChange(i+1,1) > + dataMean * scale1)
        if (perChange(i,1) < - dataMean * scale2)
            biggerThanMeanSTM = biggerThanMeanSTM+log(weightForS+weightForB+weightForE);
            if (perChange(i-1,1) < - dataMean * scale2)
                biggerThanMeanSTMstm = biggerThanMeanSTMstm+log(weightForS);
                weightForS = weightForS+1;
            elseif(perChange(i-1,1) > + dataMean * scale1)
                biggerThanMeanSTMbtm = biggerThanMeanSTMbtm+log(weightForB);
                weightForB = weightForB+1;
            else
                biggerThanMeanSTMetm = biggerThanMeanSTMetm+log(weightForE);
                weightForE = weightForE+1;
            end
        elseif (perChange(i,1) > + dataMean * scale1)
            if (perChange(i-1,1) < - dataMean * scale2)
                biggerThanMeanBTMstm = biggerThanMeanBTMstm+log(weightForS);
                weightForS = weightForS+1;
            elseif(perChange(i-1,1) > + dataMean * scale1)
                biggerThanMeanBTMbtm = biggerThanMeanBTMbtm+log(weightForB);
                weightForB = weightForB+1;
            else
                biggerThanMeanBTMetm = biggerThanMeanBTMetm+log(weightForE);
                weightForE = weightForE+1;
            end
        else
            if (perChange(i-1,1) < - dataMean * scale2)
                biggerThanMeanETMstm = biggerThanMeanETMstm+log(weightForS);
                weightForS = weightForS+1;
            elseif(perChange(i-1,1) > + dataMean * scale1)
                biggerThanMeanETMbtm = biggerThanMeanETMbtm+log(weightForB);
                weightForB = weightForB+1;
            else
                biggerThanMeanETMetm = biggerThanMeanETMetm+log(weightForE);
                weightForE = weightForE+1;
            end
        end
    else
        if (perChange(i,1) < - dataMean * scale2)
            if (perChange(i-1,1) < - dataMean * scale2)
                equalToMeanSTMstm = equalToMeanSTMstm+log(weightForS);
                weightForS = weightForS+1;
            elseif(perChange(i-1,1) > + dataMean * scale1)
                equalToMeanSTMbtm = equalToMeanSTMbtm+log(weightForB);
                weightForB = weightForB+1;
            else
                equalToMeanSTMetm = equalToMeanSTMetm+log(weightForE);
                weightForE = weightForE+1;
            end
        elseif(perChange(i,1) > + dataMean * scale1)
            if (perChange(i-1,1) < - dataMean * scale2)
                equalToMeanBTMstm = equalToMeanBTMstm+log(weightForS);
                weightForS = weightForS+1;
            elseif(perChange(i-1,1) > + dataMean * scale1)
                equalToMeanBTMbtm = equalToMeanBTMbtm+log(weightForB);
                weightForB = weightForB+1;
            else
                equalToMeanBTMetm = equalToMeanBTMetm+log(weightForE);
                weightForE = weightForE+1;
            end
        else
```

```
                if (perChange(i-1,1) < - dataMean * scale2)
                    equalToMeanETMstm = equalToMeanETMstm+log(weightForS);
                    weightForS = weightForS+1;
                elseif(perChange(i-1,1) > + dataMean * scale1)
                    equalToMeanETMbtm = equalToMeanETMbtm+log(weightForB);
                    weightForB = weightForB+1;
                else
                    equalToMeanETMetm = equalToMeanETMetm+log(weightForE);
                    weightForE = weightForE+1;
                end

            end
        end

    end
    dataMean = Sum/length(L1);
    smallerThanMeanSTM = smallerThanMeanSTMstm + smallerThanMeanSTMetm + smallerThanMeanSTMbtm;
    smallerThanMeanETM = smallerThanMeanETMstm + smallerThanMeanETMetm + smallerThanMeanETMbtm;
    smallerThanMeanBTM = smallerThanMeanBTMstm + smallerThanMeanBTMetm + smallerThanMeanBTMbtm;
    smallerThanMean = smallerThanMeanSTM + smallerThanMeanETM + smallerThanMeanBTM;
    equalToMeanSTM = equalToMeanSTMstm + equalToMeanSTMetm + equalToMeanSTMbtm;
    equalToMeanETM = equalToMeanETMstm + equalToMeanETMetm + equalToMeanETMbtm;
    equalToMeanBTM = equalToMeanBTMstm + equalToMeanBTMetm + equalToMeanBTMbtm;
    equalToMean = equalToMeanSTM + equalToMeanETM + equalToMeanBTM;
    biggerThanMeanSTM = biggerThanMeanSTMstm + biggerThanMeanSTMetm + biggerThanMeanSTMbtm;
    biggerThanMeanETM = biggerThanMeanETMstm + biggerThanMeanETMetm + biggerThanMeanETMbtm;
    biggerThanMeanBTM = biggerThanMeanBTMstm + biggerThanMeanBTMetm + biggerThanMeanBTMbtm;
    biggerThanMean = biggerThanMeanSTM + biggerThanMeanETM + biggerThanMeanBTM;
    total = biggerThanMean + equalToMean + smallerThanMean;
    constant = 25;
    dScale = 0.001;
    if((biggerThanMean > total/3) & (smallerThanMean > total/3))
        scale1 = scale1 + dScale;
        scale2 = scale2 + dScale;
    elseif((biggerThanMean > total/3) & (smallerThanMean <= total/3))
        scale1 = scale1 + dScale;
    elseif((biggerThanMean <= total/3) & (smallerThanMean > total/3))
        scale2 = scale2 + dScale;
    elseif(biggerThanMean <= total/3  - constant)
        scale1 = scale1 - dScale;
    elseif(smallerThanMean <= total/3  - constant)
        scale2 = scale2 - dScale;
    else
        flag = 1;
    end
end

secondOrderMarkovProbMatrix = [
biggerThanMean           biggerThanMeanSTM          biggerThanMeanETM       biggerThanMeanBTM
    0                        biggerThanMeanSTMstm       biggerThanMeanETMstm    biggerThanMeanBTMstm
    0                        biggerThanMeanSTMetm       biggerThanMeanETMetm    biggerThanMeanBTMetm
    0                        biggerThanMeanSTMbtm       biggerThanMeanETMbtm    biggerThanMeanBTMbtm
    equalToMean              equalToMeanSTM             equalToMeanETM          equalToMeanBTM
    0                        equalToMeanSTMstm          equalToMeanETMstm       equalToMeanBTMstm
    0                        equalToMeanSTMetm          equalToMeanETMetm       equalToMeanBTMetm
    0                        equalToMeanSTMbtm          equalToMeanETMbtm       equalToMeanBTMbtm
    smallerThanMean          smallerThanMeanSTM         smallerThanMeanETM      smallerThanMeanBTM
    0                          smallerThanMeanSTMstm       smallerThanMeanETMstm    smallerThanMeanBTMstm
    scale1                     smallerThanMeanSTMetm       smallerThanMeanETMetm    smallerThanMeanBTMetm
    scale2                     smallerThanMeanSTMbtm       smallerThanMeanETMbtm    smallerThanMeanBTMbtm
    perChange(1,1)             perChange(2,1)              0                        dataMean          ];

M=secondOrderMarkovProbMatrix;
end
```

```
% This function aims at testing the "cut and forecast" of our Probabilistic model
```

```
% This function aims at testing the "cut and forecast" of our Probabilistic model
```

```
function M = testMarkovWithWeightcut(L1)
data = L1;
dataTemp = data;
dataReal = data(1000:2000);
success=0;
k=0;
trainedMatrix = trainMarkovWithWeight(L1(1:1000,1),0.485,0.53, 0.01);
% 0.485,0.53, 0.01); are just number that makes the algorithm faster
dataTemp = flipud(dataTemp);
for(i=2:1:2000)
    perChange(i,1)=(dataTemp(i,1)-dataTemp(i-1,1))/dataTemp(i-1,1);
    perChange(i+1,1)=(dataTemp(i+1,1)-dataTemp(i,1))/dataTemp(i,1);
end
for (i=1001:1:2000)  %length(L1)instead 33
    k=k+1;
    biggerThanMean = trainedMatrix(1,1);
    biggerThanMeanSTM = trainedMatrix(1,2);
    biggerThanMeanETM = trainedMatrix(1,3);
    biggerThanMeanBTM = trainedMatrix(1,4);
    biggerThanMeanSTMstm = trainedMatrix(2,2);
    biggerThanMeanETMstm = trainedMatrix(2,3);
    biggerThanMeanBTMstm = trainedMatrix(2,4);
    biggerThanMeanSTMetm = trainedMatrix(3,2);
    biggerThanMeanETMetm = trainedMatrix(3,3);
    biggerThanMeanBTMetm = trainedMatrix(3,4);
    biggerThanMeanSTMbtm = trainedMatrix(4,2);
    biggerThanMeanETMbtm = trainedMatrix(4,3);
    biggerThanMeanBTMbtm = trainedMatrix(4,4);
    equalToMean = trainedMatrix(5,1);
    equalToMeanSTM = trainedMatrix(5,2);
    equalToMeanETM = trainedMatrix(5,3);
    equalToMeanBTM = trainedMatrix(5,4);
    equalToMeanSTMstm = trainedMatrix(6,2);
    equalToMeanETMstm = trainedMatrix(6,3);
    equalToMeanBTMstm = trainedMatrix(6,4);
    equalToMeanSTMetm = trainedMatrix(7,2);
    equalToMeanETMetm = trainedMatrix(7,3);
    equalToMeanBTMetm = trainedMatrix(7,4);
    equalToMeanSTMbtm = trainedMatrix(8,2);
    equalToMeanETMbtm = trainedMatrix(8,3);
    equalToMeanBTMbtm = trainedMatrix(8,4);
    smallerThanMean = trainedMatrix(9,1);
    smallerThanMeanSTM = trainedMatrix(9,2);
    smallerThanMeanETM = trainedMatrix(9,3);
```

```
smallerThanMeanBTM = trainedMatrix(9,4);
smallerThanMeanSTMstm = trainedMatrix(10,2);
smallerThanMeanETMstm = trainedMatrix(10,3);
smallerThanMeanBTMstm = trainedMatrix(10,4);
smallerThanMeanSTMetm = trainedMatrix(11,2);
smallerThanMeanETMetm = trainedMatrix(11,3);
smallerThanMeanBTMetm = trainedMatrix(11,4);
smallerThanMeanSTMbtm = trainedMatrix(12,2);
smallerThanMeanETMbtm = trainedMatrix(12,3);
smallerThanMeanBTMbtm = trainedMatrix(12,4);
dataMean = trainedMatrix(13,4);
scale1 = trainedMatrix(11,1);
scale2 = trainedMatrix(12,1);
SC1timesMean = scale1 * dataMean;
SC2timesMean = scale2 * dataMean;
%-----------------------------------------------------------------
if (perChange(i,1) < - dataMean * scale2)
    if (perChange(i-1,1) < - dataMean * scale2)
        %           1
        forecast(i,1) = max(trainedMatrix(10:12,2));    % smallerThanMeanSTM;
        for(j=1:1:3)
            if(forecast(i,1) == smallerThanMeanSTMstm)
                expectedRet(i,1) = mean(perChange(1:i-1,1)) - std(perChange(1:i-1,1));
            elseif(forecast(i,1) == smallerThanMeanSTMetm)
                expectedRet(i,1) = mean(perChange(1:i-1,1));
            else
                expectedRet(i,1) = mean(perChange(1:i-1,1)) + std(perChange(1:i-1,1));
            end
        end
        aRandomN = randn;
        forecastedValueReturn(i,1) = expectedRet(i,1) +  aRandomN * std(perChange(1:i-1,1));
        error(i,1) = forecastedValueReturn(i,1) - perChange(i,1);
    elseif (perChange(i-1,1)>dataMean*scale1)
        %           3
        forecast(i,1) = max(trainedMatrix(10:12,4));    % smallerThanMeanBTM
        for(j=1:1:3)
            if(forecast(i,1) == smallerThanMeanBTMstm)
                expectedRet(i,1) = mean(perChange(1:i-1,1)) - std(perChange(1:i-1,1));
            elseif(forecast(i,1) == smallerThanMeanBTMetm)
                expectedRet(i,1) = mean(perChange(1:i-1,1));
            else
                expectedRet(i,1) = mean(perChange(1:i-1,1)) + std(perChange(1:i-1,1));
            end
        end
        aRandomN = randn;
        forecastedValueReturn(i,1) = expectedRet(i,1) +  aRandomN * std(perChange(1:i-1,1));
        error(i,1) = forecastedValueReturn(i,1) - perChange(i,1);
    else
        %           2
        forecast(i,1) = max(trainedMatrix(10:12,3));    %   smallerThanMeanETM
        for(j=1:1:3)
            if(forecast(i,1) == smallerThanMeanETMstm)
                expectedRet(i,1) = mean(perChange(1:i-1,1)) - std(perChange(1:i-1,1));
            elseif(forecast(i,1) == smallerThanMeanETMetm)
                expectedRet(i,1) = mean(perChange(1:i-1,1));
            else
                expectedRet(i,1) = mean(perChange(1:i-1,1)) + std(perChange(1:i-1,1));
            end
        end
        aRandomN = randn;
        forecastedValueReturn(i,1) = expectedRet(i,1) +  aRandomN * std(perChange(1:i-1,1));
        error(i,1) = forecastedValueReturn(i,1) - perChange(i,1);
    end
    %---------------------------------------------------------------
elseif (perChange(i,1)>dataMean*scale1)
    if (perChange(i-1,1)< - dataMean*scale2)
        %             7
        forecast(i,1) = max(trainedMatrix(2:4,2));    % biggerThanMeanSTM
        for(j=1:1:3)
            if(forecast(i,1) == biggerThanMeanSTMstm)
                expectedRet(i,1) = mean(perChange(1:i-1,1)) - std(perChange(1:i-1,1));
            elseif(forecast(i,1) == biggerThanMeanSTMetm)
                expectedRet(i,1) = mean(perChange(1:i-1,1));
            else
                expectedRet(i,1) = mean(perChange(1:i-1,1)) + std(perChange(1:i-1,1));
            end
        end
        aRandomN = randn;
        forecastedValueReturn(i,1) = expectedRet(i,1) +  aRandomN * std(perChange(1:i-1,1));
        error(i,1) = forecastedValueReturn(i,1) - perChange(i,1);
    elseif (perChange(i-1,1)>dataMean*scale1)
        %             9
        forecast(i,1) = max(trainedMatrix(2:4,4));    % biggerThanMeanBTM
        for(j=1:1:3)
            if(forecast(i,1) == biggerThanMeanBTMstm)
                expectedRet(i,1) = mean(perChange(1:i-1,1)) - std(perChange(1:i-1,1));
            elseif(forecast(i,1) == biggerThanMeanBTMetm)
                expectedRet(i,1) = mean(perChange(1:i-1,1));
            else
                expectedRet(i,1) = mean(perChange(1:i-1,1)) + std(perChange(1:i-1,1));
            end
        end
        aRandomN = randn;
        forecastedValueReturn(i,1) = expectedRet(i,1) +  aRandomN * std(perChange(1:i-1,1));
        error(i,1) = forecastedValueReturn(i,1) - perChange(i,1);
    else
        %             8
        forecast(i,1) = max(trainedMatrix(2:4,3));    % biggerThanMeanETM
        for(j=1:1:3)
            if(forecast(i,1) == biggerThanMeanETMstm)
                expectedRet(i,1) = mean(perChange(1:i-1,1)) - std(perChange(1:i-1,1));
            elseif(forecast(i,1) == biggerThanMeanETMetm)
                expectedRet(i,1) = mean(perChange(1:i-1,1));
            else
                expectedRet(i,1) = mean(perChange(1:i-1,1)) + std(perChange(1:i-1,1));
            end
        end
        aRandomN = randn;
        forecastedValueReturn(i,1) = expectedRet(i,1) +  aRandomN * std(perChange(1:i-1,1));
        error(i,1) = forecastedValueReturn(i,1) - perChange(i,1);
    end
    %---------------------------------------------------------------
else
    if (perChange(i-1,1)< - dataMean*scale2)
```

```
            %           4
            forecast(i,1) = max(trainedMatrix(6:8,2));    % equalToMeanSTM
            for(j=1:1:3)
                if(forecast(i,1) == equalToMeanSTMstm)
                    expectedRet(i,1) = mean(perChange(1:i-1,1)) - std(perChange(1:i-1,1));
                elseif(forecast(i,1) == equalToMeanSTMetm)
                    expectedRet(i,1) = mean(perChange(1:i-1,1));
                else
                    expectedRet(i,1) = mean(perChange(1:i-1,1)) + std(perChange(1:i-1,1));
                end
            end
            aRandomN = randn;
            forecastedValueReturn(i,1) = expectedRet(i,1) +  aRandomN * std(perChange(1:i-1,1));
            error(i,1) = forecastedValueReturn(i,1) - perChange(i,1);
        elseif (perChange(i-1,1)>dataMean*scale1)
            %           6
            forecast(i,1) = max(trainedMatrix(6:8,4));    % equalToMeanBTM
            for(j=1:1:3)
                if(forecast(i,1) == equalToMeanBTMstm)
                    expectedRet(i,1) = mean(perChange(1:i-1,1)) - std(perChange(1:i-1,1));
                elseif(forecast(i,1) == equalToMeanBTMetm)
                    expectedRet(i,1) = mean(perChange(1:i-1,1));
                else
                    expectedRet(i,1) = mean(perChange(1:i-1,1)) + std(perChange(1:i-1,1));
                end
            end
            aRandomN = randn;
            forecastedValueReturn(i,1) = expectedRet(i,1) +  aRandomN * std(perChange(1:i-1,1));
            error(i,1) = forecastedValueReturn(i,1) - perChange(i,1);
        else
            %           5
            forecast(i,1) = max(trainedMatrix(6:8,3));    % equalToMeanETM
            for(j=1:1:3)
                if(forecast(i,1) == equalToMeanETMstm)
                    expectedRet(i,1) = mean(perChange(1:i-1,1)) - std(perChange(1:i-1,1));
                elseif(forecast(i,1) == equalToMeanETMetm)
                    expectedRet(i,1) = mean(perChange(1:i-1,1));
                else
                    expectedRet(i,1) = mean(perChange(1:i-1,1)) + std(perChange(1:i-1,1));
                end
            end
            aRandomN = randn;
            forecastedValueReturn(i,1) = expectedRet(i,1) +  aRandomN * std(perChange(1:i-1,1));
            error(i,1) = forecastedValueReturn(i,1) - perChange(i,1);
        end
    end
end
size(forecastedValueReturn)
size(perChange)
[forecastedValueReturn(1001:2000,1) perChange(1001:2000,1)];
for (i=1001:1:2000)
    if (forecastedValueReturn(i,1) * perChange(i,1) > 0)
        success = success + 1;
    end
end
successRate = success/1000
rmse = RMSE(error(1001:2000,1))
```

```
nrmse = NRMSE(error(1001:2000,1))
mae = MAE(error(1001:2000,1))
mape = MAPE(error(1001:2000,1))
datatemp = flipud(data(1:2000));
Yt_temp(1,1) = datatemp(1,1)
for (i=2:1:2000)
    Yt_temp(i,1) =  Yt_temp(i-1,1)+forecastedValueReturn(i,1);
end
x_1 = 1:2000;

figure
subplot(3,2,1)
plot(x_1, forecastedValueReturn, '.');
title('PM Return')
ylabel('Return')
subplot(3,2,2)
plot(x_1, perChange(1:2000,1), '.');
title('Real Return')
ylabel('Return')
subplot(3,2,3:4)
plot(x_1, error, '.');
title('Error')
ylabel('Return')
subplot(3,2,5:6)
plot(x_1, flipud(data(1:2000)),'.', x_1, Yt_temp, '.');
title('Real Value Vs. Forecasted Value')
ylabel('Value')

rBTM = biggerThanMean/biggerThanMean;
rETM = equalToMean/equalToMean;
rSTM = smallerThanMean/smallerThanMean;
rBTMSTM =  biggerThanMeanSTM/biggerThanMean;
rBTMETM = biggerThanMeanETM/biggerThanMean;
rBTMBTM = biggerThanMeanBTM/biggerThanMean;
rBTMSTMstm = biggerThanMeanSTMstm/biggerThanMeanSTM;
rBTMETMstm = biggerThanMeanETMstm/biggerThanMeanETM;
rBTMBTMstm = biggerThanMeanBTMstm/biggerThanMeanBTM;
rBTMSTMetm = biggerThanMeanSTMetm/biggerThanMeanSTM;
rBTMETMetm = biggerThanMeanETMetm/biggerThanMeanETM;
rBTMBTMetm = biggerThanMeanBTMetm/biggerThanMeanBTM;
rBTMSTMbtm = biggerThanMeanSTMbtm/biggerThanMeanSTM;
rBTMETMbtm = biggerThanMeanETMbtm/biggerThanMeanETM;
rBTMBTMbtm = biggerThanMeanBTMbtm/biggerThanMeanBTM;
rETMSTM = equalToMeanSTM/equalToMean;
rETMETM = equalToMeanETM/equalToMean;
rETMBTM = equalToMeanBTM/equalToMean;
rETMSTMstm = equalToMeanSTMstm/equalToMeanSTM;
rETMETMstm = equalToMeanETMstm/equalToMeanETM;
rETMBTMstm = equalToMeanBTMstm/equalToMeanBTM;
rETMSTMetm = equalToMeanSTMetm/equalToMeanSTM;
rETMETMetm = equalToMeanETMetm/equalToMeanETM;
rETMBTMetm = equalToMeanBTMetm/equalToMeanBTM;
rETMSTMbtm = equalToMeanSTMbtm/equalToMeanSTM;
rETMETMbtm = equalToMeanETMbtm/equalToMeanETM;
rETMBTMbtm = equalToMeanBTMbtm/equalToMeanBTM;
rSTMSTM = smallerThanMeanSTM/smallerThanMean;
rSTMETM = smallerThanMeanETM/smallerThanMean;
```

```
rSTMBTM = smallerThanMeanBTM/smallerThanMean;
rSTMSTMstm = smallerThanMeanSTMstm/smallerThanMeanSTM;
rSTMETMstm = smallerThanMeanETMstm/smallerThanMeanETM;
rSTMBTMstm = smallerThanMeanBTMstm/smallerThanMeanBTM;
rSTMSTMetm = smallerThanMeanSTMetm/smallerThanMeanSTM;
rSTMETMetm = smallerThanMeanETMetm/smallerThanMeanETM;
rSTMBTMetm = smallerThanMeanBTMetm/smallerThanMeanBTM;
rSTMSTMbtm = smallerThanMeanSTMbtm/smallerThanMeanSTM;
rSTMETMbtm = smallerThanMeanETMbtm/smallerThanMeanETM;
rSTMBTMbtm = smallerThanMeanBTMbtm/smallerThanMeanBTM;
ProbabilityMatrix  = [
    rBTM rBTMSTM          rBTMETM         rBTMBTM
    0    rBTMSTMstm       rBTMETMstm       rBTMBTMstm
    0    rBTMSTMetm       rBTMETMetm    rBTMBTMetm
    0    rBTMSTMbtm       rBTMETMbtm       rBTMBTMbtm
    rETM rETMSTM          rETMETM          rETMBTM
    0    rETMSTMstm       rETMETMstm       rETMBTMstm
    0    rETMSTMetm       rETMETMetm       rETMBTMetm
    0    rETMSTMbtm       rETMETMbtm       rETMBTMbtm
    rSTM rSTMSTM          rSTMETM          rSTMBTM
    0    rSTMSTMstm       rSTMETMstm       rSTMBTMstm
    0    rSTMSTMetm       rSTMETMetm       rSTMBTMetm
    0    rSTMSTMbtm       rSTMETMbtm       rSTMBTMbtm
    0    0                0                dataMean          ]
M=error;   % instead of forecast
end
```

```
% This function aims at testing the "rol and forecast" of our Probabilistic model

function M = testMarkovWithWeightrol(L1)
data = L1;
dataTemp = data;
dataReal = data(1000:2000);
success=0;
k=0;
trainedMatrix = trainMarkovWithWeight(L1(1:1000,1),0.485,0.53, 0.01);
% 0.485,0.53, 0.01); are just number that makes the algorithm faster
dataTemp = flipud(dataTemp);
for(i=2:1:2000)
    perChange(i,1)=(dataTemp(i,1)-dataTemp(i-1,1))/dataTemp(i-1,1);
    perChange(i+1,1)=(dataTemp(i+1,1)-dataTemp(i,1))/dataTemp(i,1);
end
for (i=1001:1:2000)  %length(L1)instead 33
    k=k+1;
    biggerThanMean = trainedMatrix(1,1);
    biggerThanMeanSTM = trainedMatrix(1,2);
    biggerThanMeanETM = trainedMatrix(1,3);
    biggerThanMeanBTM = trainedMatrix(1,4);
    biggerThanMeanSTMstm = trainedMatrix(2,2);
    biggerThanMeanETMstm = trainedMatrix(2,3);
    biggerThanMeanBTMstm = trainedMatrix(2,4);
    biggerThanMeanSTMetm = trainedMatrix(3,2);
    biggerThanMeanETMetm = trainedMatrix(3,3);
    biggerThanMeanBTMetm = trainedMatrix(3,4);
    biggerThanMeanSTMbtm = trainedMatrix(4,2);
```

```
biggerThanMeanETMbtm = trainedMatrix(4,3);
biggerThanMeanBTMbtm = trainedMatrix(4,4);
equalToMean = trainedMatrix(5,1);
equalToMeanSTM = trainedMatrix(5,2);
equalToMeanETM = trainedMatrix(5,3);
equalToMeanBTM = trainedMatrix(5,4);
equalToMeanSTMstm = trainedMatrix(6,2);
equalToMeanETMstm = trainedMatrix(6,3);
equalToMeanBTMstm = trainedMatrix(6,4);
equalToMeanSTMetm = trainedMatrix(7,2);
equalToMeanETMetm = trainedMatrix(7,3);
equalToMeanBTMetm = trainedMatrix(7,4);
equalToMeanSTMbtm = trainedMatrix(8,2);
equalToMeanETMbtm = trainedMatrix(8,3);
equalToMeanBTMbtm = trainedMatrix(8,4);
smallerThanMean = trainedMatrix(9,1);
smallerThanMeanSTM = trainedMatrix(9,2);
smallerThanMeanETM = trainedMatrix(9,3);
smallerThanMeanBTM = trainedMatrix(9,4);
smallerThanMeanSTMstm = trainedMatrix(10,2);
smallerThanMeanETMstm = trainedMatrix(10,3);
smallerThanMeanBTMstm = trainedMatrix(10,4);
smallerThanMeanSTMetm = trainedMatrix(11,2);
smallerThanMeanETMetm = trainedMatrix(11,3);
smallerThanMeanBTMetm = trainedMatrix(11,4);
smallerThanMeanSTMbtm = trainedMatrix(12,2);
smallerThanMeanETMbtm = trainedMatrix(12,3);
smallerThanMeanBTMbtm = trainedMatrix(12,4);
dataMean = trainedMatrix(13,4);
scale1 = trainedMatrix(11,1);
scale2 = trainedMatrix(12,1);
SC1timesMean = scale1 * dataMean;
SC2timesMean = scale2 * dataMean;
%-------------------------------------------------------------------
if (perChange(i,1) < - dataMean * scale2)
    if (perChange(i-1,1) < - dataMean * scale2)
        %             1
        forecast(i,1) = max(trainedMatrix(10:12,2));    % smallerThanMeanSTM;
        for(j=1:1:3)
            if(forecast(i,1) == smallerThanMeanSTMstm)
                expectedRet(i,1) = mean(perChange(1:i-1,1)) - std(perChange(1:i-1,1));
            elseif(forecast(i,1) == smallerThanMeanSTMetm)
                expectedRet(i,1) = mean(perChange(1:i-1,1));
            else
                expectedRet(i,1) = mean(perChange(1:i-1,1)) + std(perChange(1:i-1,1));
            end
        end
        aRandomN = randn;
        forecastedValueReturn(i,1) = expectedRet(i,1) +  aRandomN * std(perChange(1:i-1,1));
        error(i,1) = forecastedValueReturn(i,1) - perChange(i,1);
    elseif (perChange(i-1,1)>dataMean*scale1)
        %             3
        forecast(i,1) = max(trainedMatrix(10:12,4));    % smallerThanMeanBTM
        for(j=1:1:3)
            if(forecast(i,1) == smallerThanMeanBTMstm)
                expectedRet(i,1) = mean(perChange(1:i-1,1)) - std(perChange(1:i-1,1));
            elseif(forecast(i,1) == smallerThanMeanBTMetm)
```

```
                expectedRet(i,1) = mean(perChange(1:i-1,1));
            else
                expectedRet(i,1) = mean(perChange(1:i-1,1)) + std(perChange(1:i-1,1));
            end
        end
        aRandomN = randn;
        forecastedValueReturn(i,1) = expectedRet(i,1) +  aRandomN * std(perChange(1:i-1,1));
        error(i,1) = forecastedValueReturn(i,1) - perChange(i,1);
    else
        %            2
        forecast(i,1) = max(trainedMatrix(10:12,3));    %   smallerThanMeanETM
        for(j=1:1:3)
            if(forecast(i,1) == smallerThanMeanETMstm)
                expectedRet(i,1) = mean(perChange(1:i-1,1)) - std(perChange(1:i-1,1));
            elseif(forecast(i,1) == smallerThanMeanETMetm)
                expectedRet(i,1) = mean(perChange(1:i-1,1));
            else
                expectedRet(i,1) = mean(perChange(1:i-1,1)) + std(perChange(1:i-1,1));
            end
        end
        aRandomN = randn;
        forecastedValueReturn(i,1) = expectedRet(i,1) +  aRandomN * std(perChange(1:i-1,1));
        error(i,1) = forecastedValueReturn(i,1) - perChange(i,1);
    end
    %----------------------------------------------------------------
elseif (perChange(i,1)>dataMean*scale1)
    if (perChange(i-1,1)< - dataMean*scale2)
        %            7
        forecast(i,1) = max(trainedMatrix(2:4,2));    % biggerThanMeanSTM
        for(j=1:1:3)
            if(forecast(i,1) == biggerThanMeanSTMstm)
                expectedRet(i,1) = mean(perChange(1:i-1,1)) - std(perChange(1:i-1,1));
            elseif(forecast(i,1) == biggerThanMeanSTMetm)
                expectedRet(i,1) = mean(perChange(1:i-1,1));
            else
                expectedRet(i,1) = mean(perChange(1:i-1,1)) + std(perChange(1:i-1,1));
            end
        end
        aRandomN = randn;
        forecastedValueReturn(i,1) = expectedRet(i,1) +  aRandomN * std(perChange(1:i-1,1));
        error(i,1) = forecastedValueReturn(i,1) - perChange(i,1);
    elseif (perChange(i-1,1)>dataMean*scale1)
        %            9
        forecast(i,1) = max(trainedMatrix(2:4,4));    % biggerThanMeanBTM
        for(j=1:1:3)
            if(forecast(i,1) == biggerThanMeanBTMstm)
                expectedRet(i,1) = mean(perChange(1:i-1,1)) - std(perChange(1:i-1,1));
            elseif(forecast(i,1) == biggerThanMeanBTMetm)
                expectedRet(i,1) = mean(perChange(1:i-1,1));
            else
                expectedRet(i,1) = mean(perChange(1:i-1,1)) + std(perChange(1:i-1,1));
            end
        end
        aRandomN = randn;
        forecastedValueReturn(i,1) = expectedRet(i,1) +  aRandomN * std(perChange(1:i-1,1));
        error(i,1) = forecastedValueReturn(i,1) - perChange(i,1);
    else
```

```
        %            8
        forecast(i,1) = max(trainedMatrix(2:4,3));    % biggerThanMeanETM
        for(j=1:1:3)
            if(forecast(i,1) == biggerThanMeanETMstm)
                expectedRet(i,1) = mean(perChange(1:i-1,1)) - std(perChange(1:i-1,1));
            elseif(forecast(i,1) == biggerThanMeanETMetm)
                expectedRet(i,1) = mean(perChange(1:i-1,1));
            else
                expectedRet(i,1) = mean(perChange(1:i-1,1)) + std(perChange(1:i-1,1));
            end
        end
        aRandomN = randn;
        forecastedValueReturn(i,1) = expectedRet(i,1) +  aRandomN * std(perChange(1:i-1,1));
        error(i,1) = forecastedValueReturn(i,1) - perChange(i,1);
    end
    %-----------------------------------------------------------------
else
    if (perChange(i-1,1)< - dataMean*scale2)
        %            4
        forecast(i,1) = max(trainedMatrix(6:8,2));    % equalToMeanSTM
        for(j=1:1:3)
            if(forecast(i,1) == equalToMeanSTMstm)
                expectedRet(i,1) = mean(perChange(1:i-1,1)) - std(perChange(1:i-1,1));
            elseif(forecast(i,1) == equalToMeanSTMetm)
                expectedRet(i,1) = mean(perChange(1:i-1,1));
            else
                expectedRet(i,1) = mean(perChange(1:i-1,1)) + std(perChange(1:i-1,1));
            end
        end
        aRandomN = randn;
        forecastedValueReturn(i,1) = expectedRet(i,1) +  aRandomN * std(perChange(1:i-1,1));
        error(i,1) = forecastedValueReturn(i,1) - perChange(i,1);
    elseif (perChange(i-1,1)>dataMean*scale1)
        %            6
        forecast(i,1) = max(trainedMatrix(6:8,4));    % equalToMeanBTM
        for(j=1:1:3)
            if(forecast(i,1) == equalToMeanBTMstm)
                expectedRet(i,1) = mean(perChange(1:i-1,1)) - std(perChange(1:i-1,1));
            elseif(forecast(i,1) == equalToMeanBTMetm)
                expectedRet(i,1) = mean(perChange(1:i-1,1));
            else
                expectedRet(i,1) = mean(perChange(1:i-1,1)) + std(perChange(1:i-1,1));
            end
        end
        aRandomN = randn;
        forecastedValueReturn(i,1) = expectedRet(i,1) +  aRandomN * std(perChange(1:i-1,1));
        error(i,1) = forecastedValueReturn(i,1) - perChange(i,1);
    else
        %            5
        forecast(i,1) = max(trainedMatrix(6:8,3));    % equalToMeanETM
        for(j=1:1:3)
            if(forecast(i,1) == equalToMeanETMstm)
                expectedRet(i,1) = mean(perChange(1:i-1,1)) - std(perChange(1:i-1,1));
            elseif(forecast(i,1) == equalToMeanETMetm)
                expectedRet(i,1) = mean(perChange(1:i-1,1));
            else
                expectedRet(i,1) = mean(perChange(1:i-1,1)) + std(perChange(1:i-1,1));
                end
            end
            aRandomN = randn;
            forecastedValueReturn(i,1) = expectedRet(i,1) +  aRandomN * std(perChange(1:i-1,1));
            error(i,1) = forecastedValueReturn(i,1) - perChange(i,1);
        end
    end
    trainedMatrix = trainMarkovWithWeight(L1(1:i,1),0.485,0.53, 0.01);
end
size(forecastedValueReturn)
size(perChange)
[forecastedValueReturn(1001:2000,1) perChange(1001:2000,1)];
for (i=1001:1:2000)
    if (forecastedValueReturn(i,1) * perChange(i,1) > 0)
        success = success + 1;
    end
end
successRate = success/1000
rmse = RMSE(error(1001:2000,1))
nrmse = NRMSE(error(1001:2000,1))
mae = MAE(error(1001:2000,1))
mape = MAPE(error(1001:2000,1))
datatemp = flipud(data(1:2000));
Yt_temp(1,1) = datatemp(1,1)
for (i=2:1:2000)
    Yt_temp(i,1) =  Yt_temp(i-1,1)+forecastedValueReturn(i,1);
end
x_1 = 1:2000;

figure
subplot(3,2,1)
plot(x_1, forecastedValueReturn, '.');
title('PM Return')
ylabel('Return')
subplot(3,2,2)
plot(x_1, perChange(1:2000,1), '.');
title('Real Return')
ylabel('Return')
subplot(3,2,3:4)
plot(x_1, error, '.');
title('Error')
ylabel('Return')
subplot(3,2,5:6)
plot(x_1, flipud(data(1:2000)),'.', x_1, Yt_temp, '.');
title('Real Value Vs. Forecasted Value')
ylabel('Value')

rBTM = biggerThanMean/biggerThanMean;
rETM = equalToMean/equalToMean;
rSTM = smallerThanMean/smallerThanMean;
rBTMSTM =  biggerThanMeanSTM/biggerThanMean;
rBTMETM = biggerThanMeanETM/biggerThanMean;
rBTMBTM = biggerThanMeanBTM/biggerThanMean;
rBTMSTMstm = biggerThanMeanSTMstm/biggerThanMeanSTM;
rBTMETMstm = biggerThanMeanETMstm/biggerThanMeanETM;
rBTMBTMstm = biggerThanMeanBTMstm/biggerThanMeanBTM;
rBTMSTMetm = biggerThanMeanSTMetm/biggerThanMeanSTM;
```

```
rBTMETMetm = biggerThanMeanETMetm/biggerThanMeanETM;
rBTMBTMetm = biggerThanMeanBTMetm/biggerThanMeanBTM;
rBTMSTMbtm = biggerThanMeanSTMbtm/biggerThanMeanSTM;
rBTMETMbtm = biggerThanMeanETMbtm/biggerThanMeanETM;
rBTMBTMbtm = biggerThanMeanBTMbtm/biggerThanMeanBTM;
rETMSTM = equalToMeanSTM/equalToMean;
rETMETM = equalToMeanETM/equalToMean;
rETMBTM = equalToMeanBTM/equalToMean;
rETMSTMstm = equalToMeanSTMstm/equalToMeanSTM;
rETMETMstm = equalToMeanETMstm/equalToMeanETM;
rETMBTMstm = equalToMeanBTMstm/equalToMeanBTM;
rETMSTMetm = equalToMeanSTMetm/equalToMeanSTM;
rETMETMetm = equalToMeanETMetm/equalToMeanETM;
rETMBTMetm = equalToMeanBTMetm/equalToMeanBTM;
rETMSTMbtm = equalToMeanSTMbtm/equalToMeanSTM;
rETMETMbtm = equalToMeanETMbtm/equalToMeanETM;
rETMBTMbtm = equalToMeanBTMbtm/equalToMeanBTM;
rSTMSTM = smallerThanMeanSTM/smallerThanMean;
rSTMETM = smallerThanMeanETM/smallerThanMean;
rSTMBTM = smallerThanMeanBTM/smallerThanMean;
rSTMSTMstm = smallerThanMeanSTMstm/smallerThanMeanSTM;
rSTMETMstm = smallerThanMeanETMstm/smallerThanMeanETM;
rSTMBTMstm = smallerThanMeanBTMstm/smallerThanMeanBTM;
rSTMSTMetm = smallerThanMeanSTMetm/smallerThanMeanSTM;
rSTMETMetm = smallerThanMeanETMetm/smallerThanMeanETM;
rSTMBTMetm = smallerThanMeanBTMetm/smallerThanMeanBTM;
rSTMSTMbtm = smallerThanMeanSTMbtm/smallerThanMeanSTM;
rSTMETMbtm = smallerThanMeanETMbtm/smallerThanMeanETM;
rSTMBTMbtm = smallerThanMeanBTMbtm/smallerThanMeanBTM;
ProbabilityMatrix  = [
    rBTM rBTMSTM          rBTMETM         rBTMBTM
    0    rBTMSTMstm       rBTMETMstm       rBTMBTMstm
    0    rBTMSTMetm       rBTMETMetm    rBTMBTMetm
    0    rBTMSTMbtm       rBTMETMbtm       rBTMBTMbtm
    rETM rETMSTM          rETMETM          rETMBTM
    0    rETMSTMstm       rETMETMstm       rETMBTMstm
    0    rETMSTMetm       rETMETMetm       rETMBTMetm
    0    rETMSTMbtm       rETMETMbtm       rETMBTMbtm
    rSTM rSTMSTM          rSTMETM          rSTMBTM
    0    rSTMSTMstm       rSTMETMstm       rSTMBTMstm
    0    rSTMSTMetm       rSTMETMetm       rSTMBTMetm
    0    rSTMSTMbtm       rSTMETMbtm       rSTMBTMbtm
    0    0                0                dataMean          ]
M=error;   % instead of forecast
end
```

8.4 Matlab Codes for the U-NN Model

```
% function that uses Neural Networks with 2 input leading indicators in
% order to make a forecast. Print the following to make it work:
% Train_with_simple_FF_2Inputs(S_P_500, S_P_500, S_P_500,1,2,1000);

function M = Train_with_simple_FF_2Inputs
                  (InputLeadingIndicator_1, InputLeadingIndicator_2, TargetIndicator,shift_1,shift_2,selectionPeriod);

InputLeadingIndicator_1 = flipud(InputLeadingIndicator_1);
InputLeadingIndicator_2 = flipud(InputLeadingIndicator_2);
TargetIndicator = flipud(TargetIndicator);
InputLeadingIndicator_1 = reduceNoise(InputLeadingIndicator_1, 0.95);
InputLeadingIndicator_2 = reduceNoise(InputLeadingIndicator_2, 0.95);
TargetIndicator = reduceNoise(TargetIndicator, 0.95);
InputLeadingIndicator_1 = ShiftMatrix(InputLeadingIndicator_1, shift_1, selectionPeriod,1);
InputLeadingIndicator_2 = ShiftMatrix(InputLeadingIndicator_2, shift_2, selectionPeriod,1);

deltaI1(1,1) = 0;
deltaI2(1,1) = 0;
deltaT(1,1) = 0;

for(i=2:1:selectionPeriod)
    deltaT (i,1) = (TargetIndicator(i,1)-TargetIndicator(i-1,1))/TargetIndicator(i-1,1);
    deltaI1 (i,1) = (InputLeadingIndicator_1(i,1)-InputLeadingIndicator_1(i-1,1))/InputLeadingIndicator_1(i-1,1);
    deltaI2 (i,1) = (InputLeadingIndicator_2(i,1)-InputLeadingIndicator_2(i-1,1))/InputLeadingIndicator_2(i-1,1);
end
deltaI = [transpose(deltaI1);transpose(deltaI2)];
deltaT = [transpose(deltaT)];
minmax(deltaI);

net=newff(minmax(deltaI),[6, 3, 1],{'tansig', 'tansig', 'purelin'},'traingdm');
net = init(net);
net.trainParam.show = 50;
net.trainParam.lr = 0.05;
net.trainParam.mc = 0.9;
net.trainParam.epochs = 300;
net.trainParam.goal = 1e-3;
[net,tr]=train(net,deltaI,deltaT);
forecast = sim(net,deltaI);

size(deltaI1)
size(deltaI2)
size(deltaT)
plot3(deltaI1', deltaI2',deltaT,'.')

M = net;
end
```

```
% function that uses Neural Networks with 2 input leading indicators in
% order to make a "cut and forecast" model.
% Print the following to make it work:
% Train_with_simple_FF_2InputsCut(S_P_500, S_P_500, S_P_500,1,2,1000);
```

```
function M = test_Train_with_simple_FF_2InputsCut(InputLeadingIndicator_1,
InputLeadingIndicator_2, TargetIndicator,shift_1,shift_2, trainingPeriod);
numRun = length(TargetIndicator) - trainingPeriod;
periodLength=trainingPeriod+2 ;
InputLeadingIndicator_1_Mod = flipud(InputLeadingIndicator_1);
InputLeadingIndicator_2_Mod = flipud(InputLeadingIndicator_2);
TargetIndicator_Mod = flipud(TargetIndicator);
InputLeadingIndicator_1_Mod = reduceNoise(InputLeadingIndicator_1_Mod, 0.95);
InputLeadingIndicator_2_Mod = reduceNoise(InputLeadingIndicator_2_Mod, 0.95);
TargetIndicator_Mod = reduceNoise(TargetIndicator_Mod, 0.95);
InputLeadingIndicator_1_Mod = ShiftMatrix(InputLeadingIndicator_1_Mod, shift_1,
length(TargetIndicator)-2,1);
InputLeadingIndicator_2_Mod = ShiftMatrix(InputLeadingIndicator_2_Mod, shift_2,
length(TargetIndicator)-2,1);
deltaI1(1,1) = 0;
deltaI2(1,1) = 0;
realOutput(1,1) = 0;
deltaI(:,1) = [transpose(deltaI1(1,1)); transpose(deltaI2(1,1))];
forecast(1,1) = 0;
for(i=2:1:trainingPeriod)
    deltaT(i,1) = (TargetIndicator_Mod(i,1)-TargetIndicator_Mod(i-1,1))/TargetIndicator_Mod(i-1,1);
    deltaI1(i,1) = (InputLeadingIndicator_1_Mod(i,1)-InputLeadingIndicator_1_Mod(i-1,1))/
     InputLeadingIndicator_1_Mod(i-1,1);
    deltaI2(i,1) = (InputLeadingIndicator_2_Mod(i,1)-InputLeadingIndicator_2_Mod(i-1,1))/
     InputLeadingIndicator_2_Mod(i-1,1);
forecast(i,1) = 0;
end
count = 0;
net = Train_with_simple_FF_2Inputs(InputLeadingIndicator_1, InputLeadingIndicator_2,
TargetIndicator,shift_1,shift_2, 1000);
for(i=1001:1:2000)
    deltaT(i,1) = (TargetIndicator_Mod(i,1)-TargetIndicator_Mod(i-1,1))/TargetIndicator_Mod(i-1,1);
    deltaI1(i,1) = (InputLeadingIndicator_1_Mod(i,1)-InputLeadingIndicator_1_Mod(i-1,1))/
     InputLeadingIndicator_1_Mod(i-1,1);
    deltaI2(i,1) = (InputLeadingIndicator_2_Mod(i,1)-InputLeadingIndicator_2_Mod(i-1,1))/
     InputLeadingIndicator_2_Mod(i-1,1);
    deltaI = [transpose(deltaI1(i,1)); transpose(deltaI2(i,1))];
    forecast(i,1) = sim(net,deltaI);
    if (((forecast(i,1) > 0) & (deltaT(i,1)>0)) | ((forecast(i,1) < 0) & (deltaT(i,1)<0)))
        count = count+1;
    end
end
success = 0;
for (i=1001:1:2000)
    if (forecast(i,1) * deltaT(i,1) > 0)
        success = success + 1;
    end
    error(i,1) = forecast(i,1) - deltaT(i,1);
end
successRate = success/1000
rmse = RMSE(error(1001:2000,1))
nrmse = NRMSE(error(1001:2000,1))
mae = MAE(error(1001:2000,1))
mape = MAPE(error(1001:2000,1))
data = flipud(TargetIndicator);
datatemp = (data(1:2000));
Yt_temp(1,1) = datatemp(1,1)
```

```
for (i=2:1:2000)
    Yt_temp(i,1) =  Yt_temp(i-1,1)+forecast(i,1);
end
x_1 = 1:2000;
figure
subplot(3,2,1)
plot(x_1, forecast, '.');
title('PM Return')
ylabel('Return')
subplot(3,2,2)
plot(x_1, deltaT(1:2000,1), '.');
title('Real Return')
ylabel('Return')
subplot(3,2,3:4)
plot(x_1, error, '.');
title('Error')
ylabel('Return')
TargetIndicator = flipud(TargetIndicator);
subplot(3,2,5:6)
plot(x_1, data(1:2000),'.', x_1, Yt_temp, '.');
title('Real Value Vs. Forecasted Value')
ylabel('Value')

M=[forecast(1001:2000,1) deltaT(1001:2000,1) error(1001:2000,1)];
end
```

```
% function that uses Neural Networks with 2 input leading indicators in
% order to make a "rolling forecast" model.
% Print the following to make it work:
% Train_with_simple_FF_2InputsRol(S_P_500, S_P_500, S_P_500,1,2,1000);

function M = test_Train_with_simple_FF_2InputsRol(InputLeadingIndicator_1,
InputLeadingIndicator_2, TargetIndicator,shift_1,shift_2, trainingPeriod);

numRun = length(TargetIndicator) - trainingPeriod;
periodLength=trainingPeriod+2 ;
InputLeadingIndicator_1_Mod = flipud(InputLeadingIndicator_1);
InputLeadingIndicator_2_Mod = flipud(InputLeadingIndicator_2);
TargetIndicator_Mod = flipud(TargetIndicator);
InputLeadingIndicator_1_Mod = reduceNoise(InputLeadingIndicator_1_Mod, 0.95);
InputLeadingIndicator_2_Mod = reduceNoise(InputLeadingIndicator_2_Mod, 0.95);
TargetIndicator_Mod = reduceNoise(TargetIndicator_Mod, 0.95);

InputLeadingIndicator_1_Mod = ShiftMatrix(InputLeadingIndicator_1_Mod, shift_1, length(TargetIndicator)-2,1);
InputLeadingIndicator_2_Mod = ShiftMatrix(InputLeadingIndicator_2_Mod, shift_2, length(TargetIndicator)-2,1);
deltaI1(1,1) = 0;
deltaI2(1,1) = 0;
realOutput(1,1) = 0;
deltaI(:,1) = [transpose(deltaI1(1,1)); transpose(deltaI2(1,1))];
forecast(1,1) = 0;
for(i=2:1:trainingPeriod)
    deltaT(i,1) = (TargetIndicator_Mod(i,1)-TargetIndicator_Mod(i-1,1))/TargetIndicator_Mod(i-1,1);
    deltaI1(i,1) = (InputLeadingIndicator_1_Mod(i,1)-InputLeadingIndicator_1_Mod(i-1,1))/
     InputLeadingIndicator_1_Mod(i-1,1);
    deltaI2(i,1) = (InputLeadingIndicator_2_Mod(i,1)-InputLeadingIndicator_2_Mod(i-1,1))/
```

```
     InputLeadingIndicator_2_Mod(i-1,1);
    forecast(i,1) = 0;
end
count = 0;
for(i=1001:1:2000)
net = Train_with_simple_FF_2Inputs(InputLeadingIndicator_1, InputLeadingIndicator_2,
TargetIndicator,shift_1,shift_2, 1000);
    deltaT(i,1) = (TargetIndicator_Mod(i,1)-TargetIndicator_Mod(i-1,1))/TargetIndicator_Mod(i-1,1);
    deltaI1(i,1) = (InputLeadingIndicator_1_Mod(i,1)-InputLeadingIndicator_1_Mod(i-1,1))/
     InputLeadingIndicator_1_Mod(i-1,1);
    deltaI2(i,1) = (InputLeadingIndicator_2_Mod(i,1)-InputLeadingIndicator_2_Mod(i-1,1))/
     InputLeadingIndicator_2_Mod(i-1,1);

    deltaI = [transpose(deltaI1(i,1)); transpose(deltaI2(i,1))];
    forecast(i,1) = sim(net,deltaI);
    if (((forecast(i,1) > 0) & (deltaT(i,1)>0)) | ((forecast(i,1) < 0) & (deltaT(i,1)<0)))
        count = count+1;
    end
end
success = 0;
for (i=1001:1:2000)
    if (forecast(i,1) * deltaT(i,1) > 0)
        success = success + 1;
    end
    error(i,1) = forecast(i,1) - deltaT(i,1);
end
successRate = success/1000
rmse = RMSE(error(1001:2000,1))
nrmse = NRMSE(error(1001:2000,1))
mae = MAE(error(1001:2000,1))
mape = MAPE(error(1001:2000,1))
data = flipud(TargetIndicator);
datatemp = (data(1:2000));
Yt_temp(1,1) = datatemp(1,1)
for (i=2:1:2000)
    Yt_temp(i,1) =  Yt_temp(i-1,1)+forecast(i,1);
end
x_1 = 1:2000;

figure
subplot(3,2,1)
plot(x_1, forecast, '.');
title('PM Return')
ylabel('Return')
subplot(3,2,2)
plot(x_1, deltaT(1:2000,1), '.');
title('Real Return')
ylabel('Return')
subplot(3,2,3:4)
plot(x_1, error, '.');
title('Error')
ylabel('Return')

TargetIndicator = flipud(TargetIndicator);
subplot(3,2,5:6)
plot(x_1, data(1:2000),'.', x_1, Yt_temp, '.');
title('Real Value Vs. Forecasted Value')
```

```
ylabel('Value')
M=[forecast(1001:2000,1) deltaT(1001:2000,1) error(1001:2000,1)];
end
```

8.5 Matlab Codes for the U-PCA Model

```
% function that uses the PCA method to do a forecast.
% Use the following line to make it work: PCA(S_P_500, S_P_500, 1, 0, 2142)
function M = PAC(InputLeadingIndicator, TargetIndicator, shift, beta, trainingPeriod);

sumLI = 0;
sumTI = 0;

InputLeadingIndicator = flipud(InputLeadingIndicator);
TargetIndicator = flipud(TargetIndicator);
InputLeadingIndicator = reduceNoise(InputLeadingIndicator, 0.95);
TargetIndicator = reduceNoise(TargetIndicator, 0.95);
InputLeadingIndicator = ShiftMatrix(InputLeadingIndicator, shift, trainingPeriod,1);
TargetIndicator = ShiftMatrix(TargetIndicator, 0, trainingPeriod,1);
InputLeadingIndicator = CutBelow(InputLeadingIndicator, trainingPeriod,1);
TargetIndicator = CutBelow(TargetIndicator, trainingPeriod,1);
InputLeadingIndicatorCut = CutBelow(InputLeadingIndicator, trainingPeriod-1,1);
TargetIndicatorCut = CutBelow(TargetIndicator, trainingPeriod-1,1);

deltaI(1,1) = 0;
deltaT(1,1) = 0;
for(i=2:1:trainingPeriod-1)
    deltaT(i,1) = (TargetIndicatorCut(i,1)-TargetIndicatorCut(i-1,1))/TargetIndicatorCut(i-1,1);
    deltaI(i,1) = (InputLeadingIndicatorCut(i,1)-InputLeadingIndicatorCut(i-1,1))/InputLeadingIndicatorCut(i-1,1);
end

initialData = [deltaI, deltaT];
for (i=1:1:trainingPeriod-1)
    sumLI = sumLI + deltaI(i,1);
    sumTI = sumTI + deltaT(i,1);
end

meanLI = sumLI/(trainingPeriod-1);
meanTI = sumTI/(trainingPeriod-1);
for (i=1:1:trainingPeriod-1)
    deltaI(i,1) = deltaI(i,1) - meanLI;
    deltaT(i,1) = deltaT(i,1) - meanTI;
end

c = cov(deltaI(:,1), deltaT(:,1));
covLength = (c(1,1)^2 + c(2,2)^2)^(0.5);
cUnit = [   c(1,1)/covLength  c(1,2)/covLength;
    c(2,1)/covLength  c(2,2)/covLength];

[eigenVector, eigenValue] = eig(cUnit);
eigenValue = diag(eigenValue);

eigenValue1 = eigenValue(1,1);
eigenValue2 = eigenValue(2,1);
eigenValueMax = max(eigenValue);
eigenValueMin = min(eigenValue);
eigenVector1 = cUnit(:,1);
eigenVector2 = cUnit(:,2);
eigenVectors = [eigenVector1 eigenVector2];
```

```
lengthEV1 = (eigenVector1(1,1)^2 + eigenVector1(2,1)^2)^(0.5);
lengthEV2 = (eigenVector2(1,1)^2 + eigenVector2(2,1)^2)^(0.5);
eigenVector1 = [(eigenVector1(1,1)/lengthEV1); (eigenVector1(2,1)/lengthEV1)];  % the matrix need to be a unit matrix
eigenVector2 = [(eigenVector2(1,1)/lengthEV2); (eigenVector2(2,1)/lengthEV2)];  % the matrix need to be a unit matrix

if(eigenValue1 > eigenValue2)
    pca = eigenVector1;
    featureVector = [eigenVector1];
else
    pca = eigenVector2;
    featureVector = [eigenVector2];
end

eigenVectors = [eigenVector1 eigenVector2];
x1 = 1:1:trainingPeriod-1;
featureVector = transpose(featureVector); % we take the transpose of the Feature Vector
forecast = featureVector * [transpose(deltaI); transpose(deltaT)];
deltaT = transpose(deltaT);
for (i=1:1:trainingPeriod-1)
    error(1,i) = forecast(1,i) - transpose(deltaT(1,i)) - beta;
    % beta is just a correction factor that makes the results better.
end

deltaT(trainingPeriod,1) = (TargetIndicator(trainingPeriod,1)-TargetIndicator(trainingPeriod-1,1))
                                                           /TargetIndicator(trainingPeriod-1,1);
deltaI(trainingPeriod,1) = (InputLeadingIndicator(trainingPeriod,1)-InputLeadingIndicator(trainingPeriod-1,1))
                                                           /InputLeadingIndicator(trainingPeriod-1,1);

% featureVector
nextDayForecast = deltaI(trainingPeriod,1) * featureVector(1,1)/featureVector(1,2);
nextDayError = deltaT(trainingPeriod,1) - nextDayForecast;

if(i==1)
    thatDayError = 0;
else
    thatDayError = error(1,i-1);
end

meanErrors = mean(error,1);
meanError = meanErrors(1,trainingPeriod-1);
matrixSummary = [nextDayForecast, nextDayError, thatDayError, meanError];

M = matrixSummary;
end
```

```
% function that aims at testing the efficiency of the PCA "cutting and forecast" method
% Use it using the following line: testPCAcutting(S_P_500(1:2000,1), S_P_500(1:2000,1), 1, 0, 1000)

function M = testPCAcutting(InputLeadingIndicator, TargetIndicator, shift, beta, trainingPeriod);
% -- keeps a reference
data = InputLeadingIndicator;
dataReal = data(1000:2000);
% --
numRun = length(InputLeadingIndicator) - trainingPeriod;
realRet(1,1) = 0;
forecastRet(1,1) = 0;
```

```
error(1,1) = 0;
matrixResults = PCA(InputLeadingIndicator, TargetIndicator, shift, beta, trainingPeriod);
forecastRet(1,1) = matrixResults(1,1);
error(1,1) = matrixResults(1,2);
errorPreviousDay = matrixResults(1,3);
meanError = matrixResults(1,4);
matrixResults = PCA(InputLeadingIndicator, TargetIndicator, shift, meanError, trainingPeriod);
%--------- used only in this method
InputLeadingIndicator = flipud(InputLeadingIndicator);
TargetIndicator = flipud(TargetIndicator);
InputLeadingIndicator = ShiftMatrix(InputLeadingIndicator, shift, 1999,1);
TargetIndicator = ShiftMatrix(TargetIndicator, 0, 1999,1);
deltaI(1,1) = 0;
deltaT(1,1) = 0;
for(i=2:1:1999)
    deltaT(i,1) = (TargetIndicator(i,1)-TargetIndicator(i-1,1))/TargetIndicator(i-1,1);
    deltaI(i,1) = (InputLeadingIndicator(i,1)-InputLeadingIndicator(i-1,1))/InputLeadingIndicator(i-1,1);
end
%---------
realReturn(1,1) = deltaI(1,1);
err(1,1) = forecastRet(1,1)-realReturn(1,1);
for (i=1:1:1000)
    forecastRet(1000+i,1) = matrixResults(1,1);
end
success = 0;
for (i=2:1:999)
    realReturn(i,1)= deltaI(i,1);
    %     forecastRet(i,1) = Yt(i,1)-Yt(i-1,1);
    err(i,1) = forecastRet(i+1000,1)-deltaI(i+1000,1);
    if (forecastRet(i+1000,1)*deltaI(i+1000,1) > 0)
        success = success +1;
    end
end
successRate = success/999
rmse = RMSE(err(1001:2000,1))
nrmse = NRMSE(err(1001:2000,1))
mae = MAE(err(1001:2000,1))
mape = MAPE(err(1001:2000,1))
datatemp = flipud(data(1:2000));
Yt_temp(1,1) = datatemp(1,1)
for (i=2:1:1000)
    Yt_temp(i,1) =  Yt_temp(i-1,1)+forecastRet(i+1000,1);
end
x_1 = 1:1000;
x_2 = 1:999;
figure
subplot(3,2,1)
plot(forecastRet, '.');
title('PCA Return')
ylabel('Return')
subplot(3,2,2)
plot(x_1, deltaI(1:1000,1), '.');
title('Real Return')
ylabel('Return')
subplot(3,2,3:4)
plot(x_2, err(1:999), '.');
title('Error')
```

```
ylabel('Return')
subplot(3,2,5:6)
plot(x_1, flipud(data(1:1000)),'.', x_1, Yt_temp(1:1000), '.');
title('Real Value Vs. Forecasted Value')
ylabel('Value')
M = err;
end
```

```
% function that aims at testing the efficiency of the PCA "rolling forecast" method
% Use it using the following line: testPCArolling(S_P_500(1:2000,1), S_P_500(1:2000,1), 1, 0, 1000)

function M = testPCArolling(InputLeadingIndicator, TargetIndicator, shift, beta, trainingPeriod);
% -- keeps a reference
data = TargetIndicator;
dataReal = data(1000:2000,1);
% --
numRun = length(InputLeadingIndicator) - trainingPeriod;
realRet(1,1) = 0;
forecastRet(1,1) = 0;
error(1,1) = 0;
matrixResults = PCA(InputLeadingIndicator, TargetIndicator, shift, beta, trainingPeriod);
forecastRet(1,1) = matrixResults(1,1);
error(1,1) = matrixResults(1,2);
errorPreviousDay = matrixResults(1,3);
meanError = matrixResults(1,4);
for (i=1:1:1000)
    forecastRet(i,1) = 0;
end
for (i=1:1:1000)
    matrixResults = PCA(InputLeadingIndicator, TargetIndicator, shift, meanError, 1000+i-1);
    forecastRet(1000+i,1) = matrixResults(1,1);
    error(i,1) = matrixResults(1,2);
    errorPreviousDay = matrixResults(1,3);
    meanError = matrixResults(1,4);
end
%--------- used only in this method
InputLeadingIndicator = flipud(InputLeadingIndicator);
TargetIndicator = flipud(TargetIndicator);
InputLeadingIndicator = ShiftMatrix(InputLeadingIndicator, shift, 1999,1);
TargetIndicator = ShiftMatrix(TargetIndicator, 0, 1999,1);
deltaI(1,1) = 0;
deltaT(1,1) = 0;
for(i=2:1:1999)
    deltaT(i,1) = (TargetIndicator(i,1)-TargetIndicator(i-1,1))/TargetIndicator(i-1,1);
    deltaI(i,1) = (InputLeadingIndicator(i,1)-InputLeadingIndicator(i-1,1))/InputLeadingIndicator(i-1,1);
end
%---------
realReturn(1,1) = deltaI(1,1);
err(1,1) = forecastRet(1,1)-realReturn(1,1);
success = 0;
for (i=2:1:999)
    realReturn(i,1)= deltaI(i,1);
    %     forecastRet(i,1) = Yt(i,1)-Yt(i-1,1);
    err(i,1) = forecastRet(i+1000,1)-deltaI(i+1000,1);
    if (forecastRet(i+1000,1)*deltaI(i,1) > 0)
```

```
        success = success +1;
    end
end
successRate = success/999
rmse = RMSE(err(1001:2000,1))
nrmse = NRMSE(err(1001:2000,1))
mae = MAE(err(1001:2000,1))
mape = MAPE(err(1001:2000,1))
datatemp = flipud(data(1:2000));
Yt_temp(1,1) = datatemp(1,1)
for (i=2:1:1000)
    Yt_temp(i,1) =  Yt_temp(i-1,1)+forecastRet(i+1000,1);
end
x_1 = 1:1000;
x_2 = 1:999;
figure
subplot(3,2,1)
plot(forecastRet, '.');
title('PCA Return')
ylabel('Return')
subplot(3,2,2)
plot(x_1, deltaI(1:1000,1), '.');
title('Real Return')
ylabel('Return')
subplot(3,2,3:4)
plot(x_2, err(1:999), '.');
title('Error')
ylabel('Return')
subplot(3,2,5:6)
plot(x_1, flipud(data(1:1000)),'.', x_1, Yt_temp(1:1000), '.');
title('Real Value Vs. Forecasted Value')
ylabel('Value')

M = err;
end
```

8.6 Matlab Codes for the U-GA Model

```
% This a method used in our univariate GA algorithm
% It aims at translating a three digit binary into decimals

function translate = binaryToDecimal(a);
if a==000
    translate = 1;
elseif a==001
    translate = 2;
elseif a==010
    translate = 3;
elseif a==011
    translate = 4;
elseif a==100
    translate = 5;
elseif a==101
    translate = 6;
elseif a==110
    translate = 7;
elseif a==111
    translate = 8;
end
```

```
% This is a function used in the GA algorithm
% It is used in order to initialize a Population a possible results if our
% output function

function M = initializePopulation;
for (i=1:1:200)
    if i<26
        geneCode(i,1) = 000;
    elseif i<51 & i>25
        geneCode(i,1) = 001;
    elseif i<76 & i>50
        geneCode(i,1) = 010;
    elseif i<101 & i>75
        geneCode(i,1) = 011;
    elseif i<126 & i>100
        geneCode(i,1) = 100;
    elseif i<151 & i>125
        geneCode(i,1) = 101;
    elseif i<176 & i>150
        geneCode(i,1) = 110;
    elseif i<201
        geneCode(i,1) = 111;
    else
        666 % test
    end
end

M = [geneCode];
end
```

```
% This is a function used in the GA Algorithm
% Used to crossover parents

function M = crossover(parent1Binary, parent2Binary);

% takes the gene code for the w criteria ---
w1 = floor(parent1Binary/10);
w2 = floor(parent2Binary/10);
% -----------------------------------------
% takes the gene code for the s criteria ---
s1 = mod(parent1Binary,2);
s2 = mod(parent2Binary,2);
% -----------------------------------------
% crossovers the parents respective codes --
w11 = floor(w1/10);     % first digit of w1 genetic code
w12 = mod(w1,10);       % second digit of w1 genetic code
w21 = floor(w2/10);
w22 = mod(w2,10);
randa = rand;
randb = rand;
if randa > 0.5
    w31 = w11;
    w32 = w22;
else
    w31 = w21;
    w32 = w12;
end
if randb > 0.5
    s3 = s1;
else
    s3 = s2;
end
childGeneCode = 100 * w31 + 10 * w32 + s3;
% -----------------------------------------
M = childGeneCode;
end
```

```
% This function is used in the GA algorithm
% Used in mutating the children once the crossover between the parents occured
function M = mutation(childBinary);

% takes the gene code for the w criteria ---
w1 = floor(childBinary/10);
w11 = floor(w1/10);     % first digit of w1 genetic code
w12 = mod(w1,10);       % second digit of w1 genetic code
% -----------------------------------------

% takes the gene code for the s criteria ---
s1 = mod(childBinary,2);
% -----------------------------------------
randa = 1000 * rand;
randb = 1000 * rand;
```

```
randc = 1000 * rand;
if randa < 8
    if w11 == 1
        w11 = 0;
    else
        w11 = 1;
    end
elseif randb < 8
    if w12 == 1
        w12 = 0;
    else
        w12 = 1;
    end
elseif randc < 8
    if s1 == 1
        s1 = 0;
    else
        s1 = 1;
    end
end
childGeneCode = 100 * w11 + 10 * w12 + s1;
% -----------------------------------------
M = childGeneCode;
end
```

```
% Method used in the GA algorithm
% Used in order to transform the value of the input value into a DNA like
% sequence of 1's and 0's

function M = get_binaryRepfor8(input);
test =0;
period = length(input)-1;
deltaI(1,1) = 0;
for(i=2:1:period)
    deltaI(i,1) = (input(i,1)-input(i-1,1))/input(i-1,1);
end
equalSplit = period/8;
[A,indx] = sort(deltaI(:,1));
sortedInput = [A indx];
for i=1:1:floor(equalSplit)
    geneCodeI(i,1) = 000;
    sortedInput(i,3) = geneCodeI(i,1);
end
for i=ceil(equalSplit):1:floor(2*equalSplit)
    geneCodeI(i,1) = 001;
    sortedInput(i,3) = geneCodeI(i,1);
end
for i=ceil(2*equalSplit):1:floor(3*equalSplit)
    geneCodeI(i,1) = 010;
    sortedInput(i,3) = geneCodeI(i,1);
end
for i=ceil(3*equalSplit):1:floor(4*equalSplit)
    geneCodeI(i,1) = 011;
    sortedInput(i,3) = geneCodeI(i,1);
end
```

```
for i=ceil(4*equalSplit):1:floor(5*equalSplit)
    geneCodeI(i,1) = 100;
    sortedInput(i,3) = geneCodeI(i,1);
end
for i=ceil(5*equalSplit):1:floor(6*equalSplit)
    geneCodeI(i,1) = 101;
    sortedInput(i,3) = geneCodeI(i,1);
end
for i=ceil(6*equalSplit):1:floor(7*equalSplit)
    geneCodeI(i,1) = 110;
    sortedInput(i,3) = geneCodeI(i,1);
end
for i=ceil(7*equalSplit):1:period
    geneCodeI(i,1) = 111;
    sortedInput(i,3) = geneCodeI(i,1);
end
sortedGeneIndex(:,1) = sortedInput(:,2);
sortedGeneIndex(:,2) = sortedInput(:,3);
[B,idx] = sort(sortedGeneIndex(:,1));
%--------
for i=1:1:period
    if idx(i,1) < ceil(equalSplit)
        geneCodeI2(i,1) = 000;
    elseif (idx(i,1) >= ceil(equalSplit)) & (idx(i,1) < floor(2*equalSplit))
        geneCodeI2(i,1) = 001;
    elseif ((idx(i,1) >= floor(2*equalSplit)) & (idx(i,1) < ceil(3*equalSplit)))
        geneCodeI2(i,1) = 010;
    elseif ((idx(i,1) >= ceil(3*equalSplit)) & (idx(i,1) < floor(4*equalSplit)))
        geneCodeI2(i,1) = 011;
    elseif ((idx(i,1) >= floor(4*equalSplit)) & (idx(i,1) < ceil(5*equalSplit)))
        geneCodeI2(i,1) = 100;
    elseif ((idx(i,1) >= ceil(5*equalSplit)) & (idx(i,1) < floor(6*equalSplit)))
        geneCodeI2(i,1) = 101;
    elseif ((idx(i,1) >= floor(6*equalSplit)) & (idx(i,1) < ceil(7*equalSplit)))
        geneCodeI2(i,1) = 110;
    elseif ((idx(i,1) >= ceil(7*equalSplit)))
        geneCodeI2(i,1) = 111;
    end
end
%--------
M = geneCodeI2;
end
```

```
% Method used in the GA algorithm
% Gives the proportions of the individuals of different types of gene codes in a given population

function M = getProportions(population);
count000 = 0;
count001 = 0;
count010 = 0;
count011 = 0;
count100 = 0;
count101 = 0;
count110 = 0;
count111 = 0;
```

```
test12 = 0;
lengthPop = length(population);
for i=1:1:lengthPop
    geneCode = population(i,1);
    if geneCode == 000
        count000 = count000 + 1;
    elseif geneCode == 001
        count001 = count001 + 1;
    elseif geneCode == 010
        count010 = count010 + 1;
    elseif geneCode == 011
        count011 = count011 + 1;
    elseif geneCode == 100
        count100 = count100 + 1;
    elseif geneCode == 101
        count101 = count101 + 1;
    elseif geneCode == 110
        count110 = count110 + 1;
    elseif geneCode == 111
        count111 = count111 + 1;
    else
        test12 =test12+1;
    end
end
pr000 = count000/lengthPop;
pr001 = count001/lengthPop;
pr010 = count010/lengthPop;
pr011 = count011/lengthPop;
pr100 = count100/lengthPop;
pr101 = count101/lengthPop;
pr110 = count110/lengthPop;
pr111 = count111/lengthPop;
prtest = test12/lengthPop;
M = [pr000 pr001 pr010 pr011 pr100 pr101 pr110 pr111 prtest]';
end
```

```
% Method used in the GA algorithm
% The population changes given the new "enviroment". Couples are formed
% given the good ness of fit for the parents, childreen are born via this
% marriage and a new population is returned

function M = changePopulation(matrixPopulation, binaryInput, binaryTarget);
initialSize = length(matrixPopulation);
% this part of the program checks for the goodness of fit for each
% individual in the avalable population and alocates a survival rate
for i=1:1:length(matrixPopulation)
    % goodness of fit for the s parameter
    matrixPopulation(i,2) = mod(abs(matrixPopulation(i,1)-binaryTarget),2);
    % goodness of fit for the w parameter
    matrixPopulation(i,3) = abs(matrixPopulation(i,1)-binaryTarget)/10;
    if matrixPopulation(i,2) == 0 % means the model caught if we have a positive or negative correlation
        matrixPopulation(i,4) = .9; % survival/mating rate for this parent if the s parameter is good
    else
        matrixPopulation(i,4) = .8; % survival/mating rate for this parent if the s parameter is bad
    end
    if matrixPopulation(i,3) < 1
        matrixPopulation(i,5) = .9; % survival/mating rate for this parent if the w parameter is good
    else
        matrixPopulation(i,5) = .85; % survival/mating rate for this parent if the w parameter is bad
    end
    matrixPopulation(i,6) = matrixPopulation(i,4) * matrixPopulation(i,5) + rand;
    % this represents the total combined survival and fertility rate
end
% selection of parents depending on goodness of fit and chance -----------
j=0;
for i=1:1:length(matrixPopulation)
    if (matrixPopulation(i,6) > 1)
        j = j+1;
        setsOfParents(j,1) = matrixPopulation(i,1);
    end
end
% ------------------------------------------------------------------------
% makes couples amongst parents. The choice of couples are randomly chosen.
% Some parents can have multiple partners and some none depending on luck.
% resulting parents --------------------------------
for i=1:1:length(setsOfParents)
    % takes the closest higher integer and determines the parents index
    index1 = ceil(rand * length(setsOfParents));
    couple(i,1) = setsOfParents(index1);
    index2 = index1;
    while index1 == index2                       % while loop that avoids "self coupling"
        couple(i,2) = setsOfParents(index2);
        index2 = ceil(rand * length(setsOfParents));
    end
end
% --------------------------------------------------
% resulting childreen-------------------------------
for i=1:1:length(couple)
    child(i,1)= mutation(crossover(couple(i,1),couple(i,2)));
end
% --------------------------------------------------
% resulting total population -----------------------
for i=1:1:length(child)
    matrixPopulation(i,1) = child(i,1);
end
for j=1:1:length(setsOfParents)
    i = i+1;
    matrixPopulation(i,1) = setsOfParents(j,1);
end
% --------------------------------------------------

% resulting total surviving population---------------
for i=1:1:initialSize
    index = ceil(rand * initialSize);
    survivingPopulation(i,1) = matrixPopulation(i,1);
end
% --------------------------------------------------
M = survivingPopulation;
end
```

```
% Function that trains the GA algoruthm given an input, a target and the
% length of the training

function M = train_GA(input, target, period);
binaryInput = get_binaryRepfor8(input);
binaryTarget = get_binaryRepfor8(target);
population000 = initializePopulation;
population001 = initializePopulation;
population010 = initializePopulation;
population011 = initializePopulation;
population100 = initializePopulation;
population101 = initializePopulation;
population110 = initializePopulation;
population111 = initializePopulation;
for j=1:1:100
    for (i=1:1:period-1) % training part
        if binaryInput(i,1) == 000
            population000 = changePopulation(population000, binaryInput(i,1), binaryTarget(i,1));
        elseif binaryInput(i,1) == 001
            population001 = changePopulation(population001, binaryInput(i,1), binaryTarget(i,1));
        elseif binaryInput(i,1) == 010
            population010 = changePopulation(population010, binaryInput(i,1), binaryTarget(i,1));
        elseif binaryInput(i,1) == 011
            population011 = changePopulation(population011, binaryInput(i,1), binaryTarget(i,1));
        elseif binaryInput(i,1) == 100
            population100 = changePopulation(population100, binaryInput(i,1), binaryTarget(i,1));
        elseif binaryInput(i,1) == 101
            population101 = changePopulation(population101, binaryInput(i,1), binaryTarget(i,1));
        elseif binaryInput(i,1) == 110
            population110 = changePopulation(population110, binaryInput(i,1), binaryTarget(i,1));
        elseif binaryInput(i,1) == 111
            population111 = changePopulation(population111, binaryInput(i,1), binaryTarget(i,1));
        end

    end
    prop000 = getProportions(population000);
    prop001 = getProportions(population001);
    prop010 = getProportions(population010);
    prop011 = getProportions(population011);
    prop100 = getProportions(population100);
    prop101 = getProportions(population101);
    prop110 = getProportions(population110);
    prop111 = getProportions(population111);
    prop000_1(j,1) = prop000(1,1);
    prop000_2(j,1) = prop000(2,1);
    prop000_3(j,1) = prop000(3,1);
    prop000_4(j,1) = prop000(4,1);
    prop000_5(j,1) = prop000(5,1);
    prop000_6(j,1) = prop000(6,1);
    prop000_7(j,1) = prop000(7,1);
    prop000_8(j,1) = prop000(8,1);
    prop001_1(j,1) = prop001(1,1);
    prop001_2(j,1) = prop001(2,1);
    prop001_3(j,1) = prop001(3,1);
    prop001_4(j,1) = prop001(4,1);
    prop001_5(j,1) = prop001(5,1);
    prop001_6(j,1) = prop001(6,1);
    prop001_7(j,1) = prop001(7,1);
    prop001_8(j,1) = prop001(8,1);
    prop010_1(j,1) = prop010(1,1);
    prop010_2(j,1) = prop010(2,1);
    prop010_3(j,1) = prop010(3,1);
    prop010_4(j,1) = prop010(4,1);
    prop010_5(j,1) = prop010(5,1);
    prop010_6(j,1) = prop010(6,1);
    prop010_7(j,1) = prop010(7,1);
    prop010_8(j,1) = prop010(8,1);
    prop011_1(j,1) = prop011(1,1);
    prop011_2(j,1) = prop011(2,1);
    prop011_3(j,1) = prop011(3,1);
    prop011_4(j,1) = prop011(4,1);
    prop011_5(j,1) = prop011(5,1);
    prop011_6(j,1) = prop011(6,1);
    prop011_7(j,1) = prop011(7,1);
    prop011_8(j,1) = prop011(8,1);
    prop100_1(j,1) = prop100(1,1);
    prop100_2(j,1) = prop100(2,1);
    prop100_3(j,1) = prop100(3,1);
    prop100_4(j,1) = prop100(4,1);
    prop100_5(j,1) = prop100(5,1);
    prop100_6(j,1) = prop100(6,1);
    prop100_7(j,1) = prop100(7,1);
    prop100_8(j,1) = prop100(8,1);
    prop101_1(j,1) = prop101(1,1);
    prop101_2(j,1) = prop101(2,1);
    prop101_3(j,1) = prop101(3,1);
    prop101_4(j,1) = prop101(4,1);
    prop101_5(j,1) = prop101(5,1);
    prop101_6(j,1) = prop101(6,1);
    prop101_7(j,1) = prop101(7,1);
    prop101_8(j,1) = prop101(8,1);
    prop110_1(j,1) = prop110(1,1);
    prop110_2(j,1) = prop110(2,1);
    prop110_3(j,1) = prop110(3,1);
    prop110_4(j,1) = prop110(4,1);
    prop110_5(j,1) = prop110(5,1);
    prop110_6(j,1) = prop110(6,1);
    prop110_7(j,1) = prop110(7,1);
    prop110_8(j,1) = prop110(8,1);
    prop111_1(j,1) = prop111(1,1);
    prop111_2(j,1) = prop111(2,1);
    prop111_3(j,1) = prop111(3,1);
    prop111_4(j,1) = prop111(4,1);
    prop111_5(j,1) = prop111(5,1);
    prop111_6(j,1) = prop111(6,1);
    prop111_7(j,1) = prop111(7,1);
    prop111_8(j,1) = prop111(8,1);
end
prop000_1 = mean(prop000_1);
prop000_2 = mean(prop000_2);
prop000_3 = mean(prop000_3);
prop000_4 = mean(prop000_4);
prop000_5 = mean(prop000_5);
prop000_6 = mean(prop000_6);
```

```
prop000_7 = mean(prop000_7);
prop000_8 = mean(prop000_8);
prop001_1 = mean(prop001_1);
prop001_2 = mean(prop001_2);
prop001_3 = mean(prop001_3);
prop001_4 = mean(prop001_4);
prop001_5 = mean(prop001_5);
prop001_6 = mean(prop001_6);
prop001_7 = mean(prop001_7);
prop001_8 = mean(prop001_8);
prop010_1 = mean(prop010_1);
prop010_2 = mean(prop010_2);
prop010_3 = mean(prop010_3);
prop010_4 = mean(prop010_4);
prop010_5 = mean(prop010_5);
prop010_6 = mean(prop010_6);
prop010_7 = mean(prop010_7);
prop010_8 = mean(prop010_8);
prop011_1 = mean(prop011_1);
prop011_2 = mean(prop011_2);
prop011_3 = mean(prop011_3);
prop011_4 = mean(prop011_4);
prop011_5 = mean(prop011_5);
prop011_6 = mean(prop011_6);
prop011_7 = mean(prop011_7);
prop011_8 = mean(prop011_8);
prop100_1 = mean(prop100_1);
prop100_2 = mean(prop100_2);
prop100_3 = mean(prop100_3);
prop100_4 = mean(prop100_4);
prop100_5 = mean(prop100_5);
prop100_6 = mean(prop100_6);
prop100_7 = mean(prop100_7);
prop100_8 = mean(prop100_8);
prop101_1 = mean(prop101_1);
prop101_2 = mean(prop101_2);
prop101_3 = mean(prop101_3);
prop101_4 = mean(prop101_4);
prop101_5 = mean(prop101_5);
prop101_6 = mean(prop101_6);
prop101_7 = mean(prop101_7);
prop101_8 = mean(prop101_8);
prop110_1 = mean(prop110_1);
prop110_2 = mean(prop110_2);
prop110_3 = mean(prop110_3);
prop110_4 = mean(prop110_4);
prop110_5 = mean(prop110_5);
prop110_6 = mean(prop110_6);
prop110_7 = mean(prop110_7);
prop110_8 = mean(prop110_8);
prop111_1 = mean(prop111_1);
prop111_2 = mean(prop111_2);
prop111_3 = mean(prop111_3);
prop111_4 = mean(prop111_4);
prop111_5 = mean(prop111_5);
prop111_6 = mean(prop111_6);
prop111_7 = mean(prop111_7);
```

```
prop111_8 = mean(prop111_8);
M = [   prop000_1 prop001_1 prop010_1 prop011_1 prop100_1 prop101_1 prop110_1 prop111_1 binaryInput(i,1)
    prop000_2 prop001_2 prop010_2 prop011_2 prop100_2 prop101_2 prop110_2 prop111_2 binaryTarget(i,1)
    prop000_3 prop001_3 prop010_3 prop011_3 prop100_3 prop101_3 prop110_3 prop111_3 999
    prop000_4 prop001_4 prop010_4 prop011_4 prop100_4 prop101_4 prop110_4 prop111_4 999
    prop000_5 prop001_5 prop010_5 prop011_5 prop100_5 prop101_5 prop110_5 prop111_5 999
    prop000_6 prop001_6 prop010_6 prop011_6 prop100_6 prop101_6 prop110_6 prop111_6 999
    prop000_7 prop001_7 prop010_7 prop011_7 prop100_7 prop101_7 prop110_7 prop111_7 999
    prop000_8 prop001_8 prop010_8 prop011_8 prop100_8 prop101_8 prop110_8 prop111_8 999                 ];
end
```

```
% This function tests the efficiency or the GA cut and forecast algorithm in catching
% a pattern in the input and target stream of data that we input into our function

function M = test_train_GAcut(input, target);
data = flipud(input);
input = flipud(input);
target = flipud(target);
input = reduceNoise(input, 0.95);
target = reduceNoise(target, 0.95);
period = 100;
deltaI(1,1) = 0;
deltaT(1,1) = 0;
for(i=2:1:length(input))
    deltaI(i,1) = (input(i,1)-input(i-1,1))/input(i-1,1);
    deltaT(i,1) = (target(i,1)-target(i-1,1))/target(i-1,1);
end
perfectSuccess = 0; % is incremented by one everytime the model makes a perfect guess
signSuccess = 0;    % is incremented by one everytime the model makes a decent guess
testErrorInSuccess = 0; % debugging purpose
inputTemp = ShiftMatrix(input, 1, 1000, 1);
targetTemp = ShiftMatrix(target, 0, 1000, 1);
matrixForecastProb = train_GA(inputTemp(1:1000,1), targetTemp(1:1000,1), 1000)
for (i = 1000:1:2000)
    bIn = matrixForecastProb(1,9); % binary representation of input at position i
    bOu = matrixForecastProb(2,9); % binary representation of input at position i
    aRandomN = randn;
    if bIn == 000
        [forecastedValue,Index] = max(matrixForecastProb(:,1));
        if Index == 1
            perfectSuccess = perfectSuccess + 1;
            signSuccess = signSuccess + 1;
        end
        if  Index == 1 | Index == 2 | Index == 3 | Index == 4
            signSuccess = signSuccess + 1;
        end
        forecastedValueReturn(i,1) = mean(deltaI(1:i-1,1)) - 1.1503 * std(deltaI(1:i-1,1)) +
         aRandomN * std(deltaI(1:i-1,1));
        % - 1.1503 because if u look at the N distribution, this number
        % corresponds to p(1/8)
    elseif bIn == 001
        [forecastedValue,Index]  = max(matrixForecastProb(:,2));
        if Index == 2
            perfectSuccess = perfectSuccess + 1;
            signSuccess = signSuccess + 1;
        end
        if  Index == 2 | Index == 3 | Index == 4 | Index == 5
            signSuccess = signSuccess + 1;
        end
        forecastedValueReturn(i,1) = mean(deltaI(1:i-1,1)) - 0.67449 * std(deltaI(1:i-1,1)) +
         aRandomN * std(deltaI(1:i-1,1));
        % - 0.67449 because if u look at the N distribution, this number
        % corresponds to p(2/8)
    elseif bIn == 010
        [forecastedValue,Index]  = max(matrixForecastProb(:,3));
        if Index == 3
            perfectSuccess = perfectSuccess + 1;
            signSuccess = signSuccess + 1;
        end
        if  Index == 3 | Index == 4 | Index == 5 | Index == 6
            signSuccess = signSuccess + 1;
        end
        forecastedValueReturn(i,1) = mean(deltaI(1:i-1,1)) - 0.31864 * std(deltaI(1:i-1,1)) +
         aRandomN * std(deltaI(1:i-1,1));
        % - 0.31864 because if u look at the N distribution, this number
        % corresponds to p(3/8)
    elseif bIn == 011
        [forecastedValue,Index]  = max(matrixForecastProb(:,4));
        if Index == 4
            perfectSuccess = perfectSuccess + 1;
            signSuccess = signSuccess + 1;
        end
        if  Index == 4 | Index == 5 | Index == 6 | Index == 7
            signSuccess = signSuccess + 1;
        end
        forecastedValueReturn(i,1) = mean(deltaI(1:i-1,1)) - 0.01 * std(deltaI(1:i-1,1)) +
         aRandomN * std(deltaI(1:i-1,1));
        % - 0.01 (very small) because if u look at the N distribution, this number
        % corresponds to p(4/8)
    elseif bIn == 100
        [forecastedValue,Index]  = max(matrixForecastProb(:,5));
        if Index == 5
            perfectSuccess = perfectSuccess + 1;
            signSuccess = signSuccess + 1;
        end
        if  Index == 5 | Index == 6 | Index == 7 | Index == 8
            signSuccess = signSuccess + 1;
        end
        forecastedValueReturn(i,1) = mean(deltaI(1:i-1,1)) + 0.01 * std(deltaI(1:i-1,1)) +
         aRandomN * std(deltaI(1:i-1,1));
        % + 0.01 (very small) because if u look at the N distribution, this number
        % corresponds to p(4/8)
    elseif bIn == 101
        [forecastedValue,Index]  = max(matrixForecastProb(:,6));
        if Index == 6
            perfectSuccess = perfectSuccess + 1;
            signSuccess = signSuccess + 1;
        end
        forecastedValueReturn(i,1) = mean(deltaI(1:i-1,1)) + 0.31864 * std(deltaI(1:i-1,1)) +
         aRandomN * std(deltaI(1:i-1,1));
        % + 0.31864 because if u look at the N distribution, this number
        % corresponds to p(5/8)
        if  Index == 5 | Index == 6 | Index == 7 | Index == 8
            signSuccess = signSuccess + 1;
        end
    elseif bIn == 110
        [forecastedValue,Index]  = max(matrixForecastProb(:,7));
        if Index == 7
            perfectSuccess = perfectSuccess + 1;
            signSuccess = signSuccess + 1;
        end
        if  Index == 5 | Index == 6 | Index == 7 | Index == 8
            signSuccess = signSuccess + 1;
        end
        forecastedValueReturn(i,1) = mean(deltaI(1:i-1,1)) + 0.67449 * std(deltaI(1:i-1,1)) +
         aRandomN * std(deltaI(1:i-1,1));
        % + 0.67449 because if u look at the N distribution, this number
        % corresponds to p(6/8)
    elseif bIn == 111
        [forecastedValue,Index]  = max(matrixForecastProb(:,8));
        if Index == 8
            perfectSuccess = perfectSuccess + 1;
            signSuccess = signSuccess + 1;
        end
        if  Index == 5 | Index == 6 | Index == 7 | Index == 8
            signSuccess = signSuccess + 1;
        end
        forecastedValueReturn(i,1) = mean(deltaI(1:i-1,1)) + 1.1503 * std(deltaI(1:i-1,1)) +
         aRandomN * std(deltaI(1:i-1,1));
        % + 1.1503 because if u look at the N distribution, this number
        % corresponds to p(7/8)
    else
        testErrorInSuccess = testErrorInSuccess + 1;
    end
end
success = 0;
for (i=1001:1:2000)
    if (forecastedValueReturn(i,1) * deltaI(i,1) > 0)
        success = success + 1;
    end
    error(i,1) = forecastedValueReturn(i,1) - deltaI(i,1);
end
successRate = success/1000
rmse = RMSE(error(1001:2000,1))
nrmse = NRMSE(error(1001:2000,1))
mae = MAE(error(1001:2000,1))
mape = MAPE(error(1001:2000,1))
datatemp = (data(1:2000));
Yt_temp(1,1) = datatemp(1,1)
for (i=2:1:2000)
    Yt_temp(i,1) =  Yt_temp(i-1,1)+forecastedValueReturn(i,1);
end
x_1 = 1:2000;

figure
subplot(3,2,1)
plot(x_1, forecastedValueReturn, '.');
title('PM Return')
ylabel('Return')
```

```
subplot(3,2,2)
plot(x_1, deltaI(1:2000,1), '.');
title('Real Return')
ylabel('Return')
subplot(3,2,3:4)
plot(x_1, error, '.');
title('Error')
ylabel('Return')
subplot(3,2,5:6)
plot(x_1, data(1:2000),'.', x_1, Yt_temp, '.');
title('Real Value Vs. Forecasted Value')
ylabel('Value')

M=[forecastedValueReturn(1001:2000,1) deltaT(1001:2000,1)];
end
```

```
% This function tests the efficiency or the GA roll and forecast algorithm in catching
% a pattern in the input and target stream of data that we input into our function

function M = test_train_GArol(input, target);
data = flipud(input);
input = flipud(input);
target = flipud(target);
input = reduceNoise(input, 0.95);
target = reduceNoise(target, 0.95);
period = 100;
deltaI(1,1) = 0;
deltaT(1,1) = 0;
for(i=2:1:length(input))
    deltaI(i,1) = (input(i,1)-input(i-1,1))/input(i-1,1);
    deltaT(i,1) = (target(i,1)-target(i-1,1))/target(i-1,1);
end
perfectSuccess = 0; % is incremented by one everytime the model makes a perfect guess
signSuccess = 0;    % is incremented by one everytime the model makes a decent guess
testErrorInSuccess = 0; % debugging purpose
for (i = 1000:1:2000)
    inputTemp = ShiftMatrix(input, 1, i, 1);
    targetTemp = ShiftMatrix(target, 0, i, 1);
    if ((i==1000) | (i==1100) | (i==1200) | (i==1300) | (i==1400) | (i==1500) |
     (i==1600) | (i==1700) | (i==1800) | (i==1900) | (i==1999))
        % instead of doing a full rolling forecast we update the system
        % only 5 times for a time complexity issue
        i
        matrixForecastProb = train_GA(inputTemp(1:i,1), targetTemp(1:i,1), i)
    end
    bIn = matrixForecastProb(1,9); % binary representation of input at position i
    bOu = matrixForecastProb(2,9); % binary representation of input at position i
    aRandomN = randn;
    if bIn == 000
        [forecastedValue,Index] = max(matrixForecastProb(:,1));
        if Index == 1
            perfectSuccess = perfectSuccess + 1;
            signSuccess = signSuccess + 1;
        end
        if  Index == 1 | Index == 2 | Index == 3 | Index == 4
            signSuccess = signSuccess + 1;
        end
        forecastedValueReturn(i,1) = mean(deltaI(1:i-1,1)) - 1.1503 * std(deltaI(1:i-1,1)) +
         aRandomN * std(deltaI(1:i-1,1));
        % - 1.1503 because if u look at the N distribution, this number
        % corresponds to p(1/8)
    elseif bIn == 001
        [forecastedValue,Index]  = max(matrixForecastProb(:,2));
        if Index == 2
            perfectSuccess = perfectSuccess + 1;
            signSuccess = signSuccess + 1;
        end
        if  Index == 2 | Index == 3 | Index == 4 | Index == 5
            signSuccess = signSuccess + 1;
        end
        forecastedValueReturn(i,1) = mean(deltaI(1:i-1,1)) - 0.67449 * std(deltaI(1:i-1,1)) +
         aRandomN * std(deltaI(1:i-1,1));
        % - 0.67449 because if u look at the N distribution, this number
        % corresponds to p(2/8)
    elseif bIn == 010
        [forecastedValue,Index]  = max(matrixForecastProb(:,3));
        if Index == 3
            perfectSuccess = perfectSuccess + 1;
            signSuccess = signSuccess + 1;
        end
        if  Index == 3 | Index == 4 | Index == 5 | Index == 6
            signSuccess = signSuccess + 1;
        end
        forecastedValueReturn(i,1) = mean(deltaI(1:i-1,1)) - 0.31864 * std(deltaI(1:i-1,1)) +
         aRandomN * std(deltaI(1:i-1,1));
        % - 0.31864 because if u look at the N distribution, this number
        % corresponds to p(3/8)
    elseif bIn == 011
        [forecastedValue,Index]  = max(matrixForecastProb(:,4));
        if Index == 4
            perfectSuccess = perfectSuccess + 1;
            signSuccess = signSuccess + 1;
        end
        if  Index == 4 | Index == 5 | Index == 6 | Index == 7
            signSuccess = signSuccess + 1;
        end
        forecastedValueReturn(i,1) = mean(deltaI(1:i-1,1)) - 0.01 * std(deltaI(1:i-1,1)) +
         aRandomN * std(deltaI(1:i-1,1));
        % - 0.01 (very small) because if u look at the N distribution, this number
        % corresponds to p(4/8)
    elseif bIn == 100
        [forecastedValue,Index]  = max(matrixForecastProb(:,5));
        if Index == 5
            perfectSuccess = perfectSuccess + 1;
            signSuccess = signSuccess + 1;
        end
        if  Index == 5 | Index == 6 | Index == 7 | Index == 8
            signSuccess = signSuccess + 1;
        end
        forecastedValueReturn(i,1) = mean(deltaI(1:i-1,1)) + 0.01 * std(deltaI(1:i-1,1)) +
         aRandomN * std(deltaI(1:i-1,1));
        % + 0.01 (very small) because if u look at the N distribution, this number
```

```
        % corresponds to p(4/8)
    elseif bIn == 101
        [forecastedValue,Index]  = max(matrixForecastProb(:,6));
        if Index == 6
            perfectSuccess = perfectSuccess + 1;
            signSuccess = signSuccess + 1;
        end
        forecastedValueReturn(i,1) = mean(deltaI(1:i-1,1)) + 0.31864 * std(deltaI(1:i-1,1)) +
         aRandomN * std(deltaI(1:i-1,1));
        % + 0.31864 because if u look at the N distribution, this number
        % corresponds to p(5/8)
        if  Index == 5 | Index == 6 | Index == 7 | Index == 8
            signSuccess = signSuccess + 1;
        end
    elseif bIn == 110
        [forecastedValue,Index]  = max(matrixForecastProb(:,7));
        if Index == 7
            perfectSuccess = perfectSuccess + 1;
            signSuccess = signSuccess + 1;
        end
        if  Index == 5 | Index == 6 | Index == 7 | Index == 8
            signSuccess = signSuccess + 1;
        end
        forecastedValueReturn(i,1) = mean(deltaI(1:i-1,1)) + 0.67449 * std(deltaI(1:i-1,1)) +
         aRandomN * std(deltaI(1:i-1,1));
        % + 0.67449 because if u look at the N distribution, this number
        % corresponds to p(6/8)
    elseif bIn == 111
        [forecastedValue,Index]  = max(matrixForecastProb(:,8));
        if Index == 8
            perfectSuccess = perfectSuccess + 1;
            signSuccess = signSuccess + 1;
        end
        if  Index == 5 | Index == 6 | Index == 7 | Index == 8
            signSuccess = signSuccess + 1;
        end
        forecastedValueReturn(i,1) = mean(deltaI(1:i-1,1)) + 1.1503 * std(deltaI(1:i-1,1)) +
         aRandomN * std(deltaI(1:i-1,1));
        % + 1.1503 because if u look at the N distribution, this number
        % corresponds to p(7/8)
    else
        testErrorInSuccess = testErrorInSuccess + 1;
    end
end
success = 0;
for (i=1001:1:2000)
    if (forecastedValueReturn(i,1) * deltaI(i,1) > 0)
        success = success + 1;
    end
    error(i,1) = forecastedValueReturn(i,1) - deltaI(i,1);
end
successRate = success/1000
rmse = RMSE(error(1001:2000,1))
nrmse = NRMSE(error(1001:2000,1))
mae = MAE(error(1001:2000,1))
mape = MAPE(error(1001:2000,1))
datatemp = (data(1:2000));
```

```
Yt_temp(1,1) = datatemp(1,1)
for (i=2:1:2000)
    Yt_temp(i,1) =  Yt_temp(i-1,1)+forecastedValueReturn(i,1);
end
x_1 = 1:2000;
figure
subplot(3,2,1)
plot(x_1, forecastedValueReturn, '.');
title('PM Return')
ylabel('Return')
subplot(3,2,2)
plot(x_1, deltaI(1:2000,1), '.');
title('Real Return')
ylabel('Return')
subplot(3,2,3:4)
plot(x_1, error, '.');
title('Error')
ylabel('Return')
subplot(3,2,5:6)
plot(x_1, data(1:2000),'.', x_1, Yt_temp, '.');
title('Real Value Vs. Forecasted Value')
ylabel('Value')

M=[forecastedValueReturn(1001:2000,1) deltaT(1001:2000,1)];
end
```

8.7 Matlab Codes for the First Attempt to Solve Complexity Issue

```
% One of the function used in our time complexity issue for a multivariate
% approach.
% The function could be viewed as a hash table that gets a key and returns
% a unique string for the specific key.
% It is used to see what 2 indicators are being prossessed

function modMatrix = hashGet(key)
if (key>=1 & key<=39)
    modMatrix = ['L1 L', num2str(key+1)];
elseif (key>=40 & key<=77)
    modMatrix = ['L2 L', num2str(key-37)];
elseif (key>=78 & key<=114)
    modMatrix = ['L3 L', num2str(key-37-37)];
elseif (key>=115 & key<=150)
    modMatrix = ['L4 L', num2str(key-37-37-36)];
elseif (key>=151 & key<=185)
    modMatrix = ['L5 L', num2str(key-37-37-36-35)];
elseif (key>=186 & key<=219)
    modMatrix = ['L6 L', num2str(key-37-37-36-35-34)];
elseif (key>=220 & key<=252)
    modMatrix = ['L7 L', num2str(key-37-37-36-35-34-33)];
elseif (key>=253 & key<=284)
    modMatrix = ['L8 L', num2str(key-37-37-36-35-34-33-32)];
elseif (key>=285 & key<=315)
    modMatrix = ['L9 L', num2str(key-37-37-36-35-34-33-32-31)];
elseif (key>=316 & key<=345)
    modMatrix = ['L10 L', num2str(key-37-37-36-35-34-33-32-31-30)];
elseif (key>=346 & key<=374)
    modMatrix = ['L11 L', num2str(key-37-37-36-35-34-33-32-31-30-29)];
elseif (key>=375 & key<=402)
    modMatrix = ['L12 L', num2str(key-37-37-36-35-34-33-32-31-30-29-28)];
elseif (key>=403 & key<=429)
    modMatrix = ['L13 L', num2str(key-37-37-36-35-34-33-32-31-30-29-28-27)];
elseif (key>=430 & key<=455)
    modMatrix = ['L14 L', num2str(key-37-37-36-35-34-33-32-31-30-29-28-27-26)];
elseif (key>=456 & key<=480)
    modMatrix = ['L15 L', num2str(key-37-37-36-35-34-33-32-31-30-29-28-27-26-25)];
elseif (key>=481 & key<=504)
    modMatrix = ['L16 L', num2str(key-37-37-36-35-34-33-32-31-30-29-28-27-26-25-24)];
elseif (key>=505 & key<=527)
    modMatrix = ['L17 L', num2str(key-37-37-36-35-34-33-32-31-30-29-28-27-26-25-24-23)];
elseif (key>=528 & key<=549)
    modMatrix = ['L18 L', num2str(key-37-37-36-35-34-33-32-31-30-29-28-27-26-25-24-23-22)];
elseif (key>=550 & key<=570)
    modMatrix = ['L19 L', num2str(key-37-37-36-35-34-33-32-31-30-29-28-27-26-25-24-23-22-21)];
elseif (key>=571 & key<=590)
    modMatrix = ['L20 L', num2str(key-37-37-36-35-34-33-32-31-30-29-28-27-26-25-24-23-22-21-20)];
elseif (key>=591 & key<=609)
    modMatrix = ['L21 L', num2str(key-37-37-36-35-34-33-32-31-30-29-28-27-26-25-24-23-22-21-20-19)];
elseif (key>=610 & key<=627)
    modMatrix = ['L22 L', num2str(key-37-37-36-35-34-33-32-31-30-29-28-27-26-25-24-23-22-21-20-19-18)];
```

```
elseif (key>=628 & key<=644)
    modMatrix = ['L23 L', num2str(key-37-37-36-35-34-33-32-31-30-29-28-27-26-25-24-23-22-21-20-19-18-17)];
elseif (key>=645 & key<=660)
    modMatrix = ['L24 L', num2str(key-37-37-36-35-34-33-32-31-30-29-28-27-26-25-24-23-22-21-20-19-18-17-16)];
elseif (key>=661 & key<=675)
    modMatrix = ['L25 L', num2str(key-37-37-36-35-34-33-32-31-30-29-28-27-26-25-24-23-22-21-20-19-18-17-16-15)];
elseif (key>=676 & key<=689)
    modMatrix = ['L26 L', num2str(key-37-37-36-35-34-33-32-31-30-29-28-27-26-25-24-23-22-21-20-19-18-17-16-15-14)];
elseif (key>=690 & key<=702)
    modMatrix = ['L27 L', num2str(key-37-37-36-35-34-33-32-31-30-29-28-27-26-25-24-23-22-21-20-19-18-17-16-15-14-
     13)];
elseif (key>=703 & key<=714)
    modMatrix = ['L28 L', num2str(key-37-37-36-35-34-33-32-31-30-29-28-27-26-25-24-23-22-21-20-19-18-17-16-15-14-
     13-12)];
elseif (key>=715 & key<=725)
    modMatrix = ['L29 L', num2str(key-37-37-36-35-34-33-32-31-30-29-28-27-26-25-24-23-22-21-20-19-18-17-16-15-14-
     13-12-11)];
elseif (key>=726 & key<=735)
    modMatrix = ['L30 L', num2str(key-37-37-36-35-34-33-32-31-30-29-28-27-26-25-24-23-22-21-20-19-18-17-16-15-14-
     13-12-11-10)];
elseif (key>=736 & key<=744)
    modMatrix = ['L31 L', num2str(key-37-37-36-35-34-33-32-31-30-29-28-27-26-25-24-23-22-21-20-19-18-17-16-15-14-
     13-12-11-10-9)];
elseif (key>=745 & key<=752)
    modMatrix = ['L32 L', num2str(key-37-37-36-35-34-33-32-31-30-29-28-27-26-25-24-23-22-21-20-19-18-17-16-15-14-
     13-12-11-10-9-8)];
elseif (key>=753 & key<=759)
    modMatrix = ['L33 L', num2str(key-37-37-36-35-34-33-32-31-30-29-28-27-26-25-24-23-22-21-20-19-18-17-16-15-14-
     13-12-11-10-9-8-7)];
elseif (key>=760 & key<=765)
    modMatrix = ['L34 L', num2str(key-37-37-36-35-34-33-32-31-30-29-28-27-26-25-24-23-22-21-20-19-18-17-16-15-14-
     13-12-11-10-9-8-7-6)];
elseif (key>=765 & key<=770)
    modMatrix = ['L35 L', num2str(key-37-37-36-35-34-33-32-31-30-29-28-27-26-25-24-23-22-21-20-19-18-17-16-15-14-
     13-12-11-10-9-8-7-6-5)];
elseif (key>=770 & key<=774)
    modMatrix = ['L36 L', num2str(key-37-37-36-35-34-33-32-31-30-29-28-27-26-25-24-23-22-21-20-19-18-17-16-15-14-
     13-12-11-10-9-8-7-6-5-4)];
elseif (key>=775 & key<=777)
    modMatrix = ['L37 L', num2str(key-37-37-36-35-34-33-32-31-30-29-28-27-26-25-24-23-22-21-20-19-18-17-16-15-14-
     13-12-11-10-9-8-7-6-5-4-3)];
elseif (key>=778 & key<=779)
    modMatrix = ['L38 L', num2str(key-37-37-36-35-34-33-32-31-30-29-28-27-26-25-24-23-22-21-20-19-18-17-16-15-14-
     13-12-11-10-9-8-7-6-5-4-3-2)];
else (key>=780 & key<=780)
    modMatrix = ['L39 L', num2str(40)];
end
```

```
% One of the function used in our time complexity issue for a multivariate
% approach.
% The function looks at the historical value of two different indicators
% and retrieve the shift that makes the two data stream have the highest
% correlation

function M = CorrUsingBestShiftSingleton(L1, L2)
bestCorrTemp1=0;
```

```
bestCorrTemp2=0;
bestCorr=0;
newCorr=0;
boundary=4;
bestShift=0;
periodLength=length(L1)-boundary;
for(i=0:1:boundary)
    L1_mod = ShiftMatrix(L1, i, periodLength,1);
    L2_mod = CutAbove(L2, periodLength, 1);
    newCorr = corr2(L1_mod, L2_mod);
    if(abs(newCorr)>=abs(bestCorrTemp1))
        bestCorrTemp1 =newCorr;
        bestShift=i;
        matrix_Best_Shift = L1_mod;  %%%%%%%%%%%%%%% SEE Best_Period ()
        best_Shift_1 = i;
    end
end
for(i=0:1:boundary)
    L2_mod = ShiftMatrix(L2, i, periodLength,1);
    L1_mod = CutAbove(L1, periodLength, 1);
    newCorr = corr2(L1_mod, L2_mod);
    if(abs(newCorr)>=abs(bestCorrTemp2))
        bestCorrTemp2=newCorr;
        bestShift=i;
        matrix_Best_Shift = L1_mod;  %%%%%%%%%%%%%%% SEE Best_Period ()
        best_Shift_2 = i;
    end
end
if (bestCorrTemp1>bestCorrTemp2)
    bestCorr = bestCorrTemp1;
else
    bestCorr = bestCorrTemp2;
end
if (bestCorrTemp1==bestCorr)
    bs = + best_Shift_1;
else
    bs = - best_Shift_2;
end
M=[bestCorr, bs];
end
```

```
% One of the function used in our time complexity issue for a multivariate
% approach.
% The function looks at the historical value of all of our different indicators
% and retrieve the shift that makes the data of every two indicator the highest
% and returns a list of decreasing order of correlation as well as the
% shift
% Use it pasting the following:
% CorrUsingBestShift(AMEX_Oil_Index, COT_2_YEAR_U_S__T_NOTES_Commercial_Traders_Net,
%                        COT_2_YEAR_U_S__T_NOTES_Non_Commercial_Traders_Net,
%                            ....,  Volume_S_P_500)

function M = CorrUsingBestShift(L1,L2,L3,L4,L5,L6,L7,L8,L9,L10,L11,L12,L13,L14,L15,L16,
                                L17,L18,L19,L20,L21,L22,L23,L24,L25,L26,L27,L28,L29,L30,
                                                L31,L32,L33,L34,L35,L36,L37,L38,L39,L40)
```

```
A=[L1,L2,L3,L4,L5,L6,L7,L8,L9,L10,L11,L12,L13,L14,L15,L16,L17,L18,L19,L20,L21,L22,L23,
                    L24,L25,L26,L27,L28,L29,L30,L31,L32,L33,L34,L35,L36,L37,L38,L39,L40];
corrMatrix = transpose([1:780]);     % because (39+1)/2 * 39  = 780
matrixShifts = transpose([1:780]);
k=0;
count = 1;
for(i=1:1:39)
    for(j=2:1:40)
        if(i<j)
            k = k+1;
            temp = CorrUsingBestShiftSingleton(A(:,i), A(:,j));
            corrMatrix(k,1) = temp(1,1);
            matrixShifts(k,1) = temp(1,2);
        end
    end
end
temp1 = cell(780,1)
temp1(:,1) = {corrMatrix};
sortedCorrMatrix = sort(abs(corrMatrix)); % sort correlation in decreasing order of importance
temp2 = cell(780,4);
temp2(:,1) = num2cell(sortedCorrMatrix);
for(i=1:1:780)
    for(j=1:1:780)
        if(sortedCorrMatrix(i,1)==corrMatrix(j,1) | sortedCorrMatrix(i,1)==-corrMatrix(j,1))
            temp2(i,2) = {corrMatrix(j,1)};
            temp2(i,3) = {hashGet(j)};
            temp2(i,4) = {matrixShifts(j,1)};
        end
    end
end
M = temp2;
end
```

8.8 Matlab Codes for the M-NNN model

```
% test_Train_with_simple_FF_40InputsCut(5, AMEX_Oil_Index, COT_2_YEAR_U_S__T_NOTES_Commercial_Traders_Net,
% COT_2_YEAR_U_S__T_NOTES_Non_Commercial_Traders_Net, COT_2_YEAR_U_S__T_NOTES_Small_Traders_Net,
% COT_CRUDE_OIL__LIGHT_SWEET_Commercial_Traders_Net, COT_CRUDE_OIL__LIGHT_SWEET_Non_Commercial_Traders_Net,
% COT_CRUDE_OIL__LIGHT_SWEET_Small_Traders_Net, COT_Dow_Jones_Commercial_Traders_Net,
% COT_Dow_Jones_Non_Commercial_Traders_Net, COT_Dow_Jones_Small_Traders_Net,
% COT_GOLD_Commercial_Traders_Net, COT_GOLD_Non_Commercial_Traders_Net, COT_GOLD_Small_Traders_Net,
% COT_INTEREST_RATES__FED_FUNDS__Commercial_Traders_Net,
% COT_INTEREST_RATES__FED_FUNDS__Non_Commercial_Traders_Net,
% COT_INTEREST_RATES__FED_FUNDS__Small_Traders_Net, COT_Mini_S_P_500_Commercial_Traders_Net,
% COT_Mini_S_P_500_Non_Commercial_Traders_Net, COT_Mini_S_P_500_Small_Traders_Net,
% COT_SILVER_Commercial_Traders_Net, COT_SILVER_Non_Commercial_Traders_Net,
% COT_SILVER_Small_Traders_Net, COT_S_P_500_STOCK_INDEX_FUTURES_Commercial_Traders_Net,
% COT_S_P_500_STOCK_INDEX_FUTURES_Non_Commercial_Traders_Net,
% COT_S_P_500_STOCK_INDEX_FUTURES_Small_Traders_Net,
% COT_UNLEADED_GASOLINE__NY_Commercial_Traders_Net, COT_UNLEADED_GASOLINE__NY_Non_Commercial_Traders_Net,
% COT_UNLEADED_GASOLINE__NY_Small_Traders_Net, COT_U_S__DOLLAR_INDEX_Commercial_Traders_Net,
% COT_U_S__DOLLAR_INDEX_Non_Commercial_Traders_Net, COT_U_S__DOLLAR_INDEX_Small_Traders_Net,
% Dow_Jones, Dow_Jones_Moving_Average_30_days, Dow_Jones_Transportation, Dow_Jones_Utilities,
% Gold_and_Silver, High__S_P_500, Low__S_P_500, News, Volume_S_P_500, S_P_500)

function M = test_Train_with_simple_FF_40InputsCut(shift,  L1, L2, L3, L4, L5, L6, L7, L8, L9, L10,
L11, L12, L13, L14, L15, L16, L17, L18, L19, L20, L21, L22, L23, L24, L25, L26, L27,
L28, L29, L30, L31, L32, L33, L34, L35, L36, L37, L38, L39, L40, TargetIndicator);
data = flipud(TargetIndicator);
L1 = flipud(L1);
L2 = flipud(L2);
L3 = flipud(L3);
L4 = flipud(L4);
L5 = flipud(L5);
L6 = flipud(L6);
L7 = flipud(L7);
L8 = flipud(L8);
L9 = flipud(L9);
L10 = flipud(L10);
L11 = flipud(L11);
L12 = flipud(L12);
L13 = flipud(L13);
L14 = flipud(L14);
L15 = flipud(L15);
L16 = flipud(L16);
L17 = flipud(L17);
L18 = flipud(L18);
L19 = flipud(L19);
L20 = flipud(L20);
L21 = flipud(L21);
L22 = flipud(L22);
L23 = flipud(L23);
L24 = flipud(L24);
L25 = flipud(L25);
L26 = flipud(L26);
L27 = flipud(L27);
L28 = flipud(L28);
L29 = flipud(L29);
```

```
L30 = flipud(L30);
L31 = flipud(L31);
L32 = flipud(L32);
L33 = flipud(L33);
L34 = flipud(L34);
L35 = flipud(L35);
L36 = flipud(L36);
L37 = flipud(L37);
L38 = flipud(L38);
L39 = flipud(L39);
L40 = flipud(L40);
TargetIndicator = flipud(TargetIndicator);
L1 = reduceNoise(L1, 0.95);
L2 = reduceNoise(L2, 0.95);
L3 = reduceNoise(L3, 0.95);
L4 = reduceNoise(L4, 0.95);
L5 = reduceNoise(L5, 0.95);
L6 = reduceNoise(L6, 0.95);
L7 = reduceNoise(L7, 0.95);
L8 = reduceNoise(L8, 0.95);
L9 = reduceNoise(L9, 0.95);
L10 = reduceNoise(L10, 0.95);
L11 = reduceNoise(L11, 0.95);
L12 = reduceNoise(L12, 0.95);
L13 = reduceNoise(L13, 0.95);
L14 = reduceNoise(L14, 0.95);
L15 = reduceNoise(L15, 0.95);
L16 = reduceNoise(L16, 0.95);
L17 = reduceNoise(L17, 0.95);
L18 = reduceNoise(L18, 0.95);
L19 = reduceNoise(L19, 0.95);
L20 = reduceNoise(L20, 0.95);
L21 = reduceNoise(L21, 0.95);
L22 = reduceNoise(L22, 0.95);
L23 = reduceNoise(L23, 0.95);
L24 = reduceNoise(L24, 0.95);
L25 = reduceNoise(L25, 0.95);
L26 = reduceNoise(L26, 0.95);
L27 = reduceNoise(L27, 0.95);
L28 = reduceNoise(L28, 0.95);
L29 = reduceNoise(L29, 0.95);
L30 = reduceNoise(L30, 0.95);
L31 = reduceNoise(L31, 0.95);
L32 = reduceNoise(L32, 0.95);
L33 = reduceNoise(L33, 0.95);
L34 = reduceNoise(L34, 0.95);
L35 = reduceNoise(L35, 0.95);
L36 = reduceNoise(L36, 0.95);
L37 = reduceNoise(L37, 0.95);
L38 = reduceNoise(L38, 0.95);
L39 = reduceNoise(L39, 0.95); % news indicators
L40 = reduceNoise(L40, 0.95);
TargetIndicator = reduceNoise(TargetIndicator, 0.95);
boundary=10;
periodLength=length(L1)-boundary;
L1 = ShiftMatrix(L1, shift, periodLength,1);
L2 = ShiftMatrix(L2, shift, periodLength,1);
```

```
L3 = ShiftMatrix(L3, shift, periodLength,1);
L4 = ShiftMatrix(L4, shift, periodLength,1);
L5 = ShiftMatrix(L5, shift, periodLength,1);
L6 = ShiftMatrix(L6, shift, periodLength,1);
L7 = ShiftMatrix(L7, shift, periodLength,1);
L8 = ShiftMatrix(L8, shift, periodLength,1);
L9 = ShiftMatrix(L9, shift, periodLength,1);
L10 = ShiftMatrix(L10, shift, periodLength,1);
L11 = ShiftMatrix(L11, shift, periodLength,1);
L12 = ShiftMatrix(L12, shift, periodLength,1);
L13 = ShiftMatrix(L13, shift, periodLength,1);
L14 = ShiftMatrix(L14, shift, periodLength,1);
L15 = ShiftMatrix(L15, shift, periodLength,1);
L16 = ShiftMatrix(L16, shift, periodLength,1);
L17 = ShiftMatrix(L17, shift, periodLength,1);
L18 = ShiftMatrix(L18, shift, periodLength,1);
L19 = ShiftMatrix(L19, shift, periodLength,1);
L20 = ShiftMatrix(L20, shift, periodLength,1);
L21 = ShiftMatrix(L21, shift, periodLength,1);
L22 = ShiftMatrix(L22, shift, periodLength,1);
L23 = ShiftMatrix(L23, shift, periodLength,1);
L24 = ShiftMatrix(L24, shift, periodLength,1);
L25 = ShiftMatrix(L25, shift, periodLength,1);
L26 = ShiftMatrix(L26, shift, periodLength,1);
L27 = ShiftMatrix(L27, shift, periodLength,1);
L28 = ShiftMatrix(L28, shift, periodLength,1);
L29 = ShiftMatrix(L29, shift, periodLength,1);
L30 = ShiftMatrix(L30, shift, periodLength,1);
L31 = ShiftMatrix(L31, shift, periodLength,1);
L32 = ShiftMatrix(L32, shift, periodLength,1);
L33 = ShiftMatrix(L33, shift, periodLength,1);
L34 = ShiftMatrix(L34, shift, periodLength,1);
L35 = ShiftMatrix(L35, shift, periodLength,1);
L36 = ShiftMatrix(L36, shift, periodLength,1);
L37 = ShiftMatrix(L37, shift, periodLength,1);
L38 = ShiftMatrix(L38, shift, periodLength,1);
L39 = ShiftMatrix(L39, 0, periodLength,1); % news is a coincident indicator
L40 = ShiftMatrix(L40, shift, periodLength,1);
deltaL1(1,1) = 0;
deltaL2(1,1) = 0;
deltaL3(1,1) = 0;
deltaL4(1,1) = 0;
deltaL5(1,1) = 0;
deltaL6(1,1) = 0;
deltaL7(1,1) = 0;
deltaL8(1,1) = 0;
deltaL9(1,1) = 0;
deltaL10(1,1) = 0;
deltaL11(1,1) = 0;
deltaL12(1,1) = 0;
deltaL13(1,1) = 0;
deltaL14(1,1) = 0;
deltaL15(1,1) = 0;
deltaL16(1,1) = 0;
deltaL17(1,1) = 0;
deltaL18(1,1) = 0;
deltaL19(1,1) = 0;
```

```
deltaL20(1,1) = 0;
deltaL21(1,1) = 0;
deltaL22(1,1) = 0;
deltaL23(1,1) = 0;
deltaL24(1,1) = 0;
deltaL25(1,1) = 0;
deltaL26(1,1) = 0;
deltaL27(1,1) = 0;
deltaL28(1,1) = 0;
deltaL29(1,1) = 0;
deltaL30(1,1) = 0;
deltaL31(1,1) = 0;
deltaL32(1,1) = 0;
deltaL33(1,1) = 0;
deltaL34(1,1) = 0;
deltaL35(1,1) = 0;
deltaL36(1,1) = 0;
deltaL37(1,1) = 0;
deltaL38(1,1) = 0;
deltaL39(1,1) = 0;
deltaL40(1,1) = 0;
for(i=2:1:periodLength)
    deltaT (i,1) = (TargetIndicator(i,1)-TargetIndicator(i-1,1));
    deltaL1 (i,1) = (L1(i,1)-L1(i-1,1));
    deltaL2 (i,1) = (L2(i,1)-L2(i-1,1));
    deltaL3 (i,1) = (L3(i,1)-L3(i-1,1));
    deltaL4 (i,1) = (L4(i,1)-L4(i-1,1));
    deltaL5 (i,1) = (L5(i,1)-L5(i-1,1));
    deltaL6 (i,1) = (L6(i,1)-L6(i-1,1));
    deltaL7 (i,1) = (L7(i,1)-L7(i-1,1));
    deltaL8 (i,1) = (L8(i,1)-L8(i-1,1));
    deltaL9 (i,1) = (L9(i,1)-L9(i-1,1));
    deltaL10 (i,1) = (L10(i,1)-L10(i-1,1));
    deltaL11 (i,1) = (L11(i,1)-L11(i-1,1));
    deltaL12 (i,1) = (L12(i,1)-L12(i-1,1));
    deltaL13 (i,1) = (L13(i,1)-L13(i-1,1));
    deltaL14 (i,1) = (L14(i,1)-L14(i-1,1));
    deltaL15 (i,1) = (L15(i,1)-L15(i-1,1));
    deltaL16 (i,1) = (L16(i,1)-L16(i-1,1));
    deltaL17 (i,1) = (L17(i,1)-L17(i-1,1));
    deltaL18 (i,1) = (L18(i,1)-L18(i-1,1));
    deltaL19 (i,1) = (L19(i,1)-L19(i-1,1));
    deltaL20 (i,1) = (L20(i,1)-L20(i-1,1));
    deltaL21 (i,1) = (L21(i,1)-L21(i-1,1));
    deltaL22 (i,1) = (L22(i,1)-L22(i-1,1));
    deltaL23 (i,1) = (L23(i,1)-L23(i-1,1));
    deltaL24 (i,1) = (L24(i,1)-L24(i-1,1));
    deltaL25 (i,1) = (L25(i,1)-L25(i-1,1));
    deltaL26 (i,1) = (L26(i,1)-L26(i-1,1));
    deltaL27 (i,1) = (L27(i,1)-L27(i-1,1));
    deltaL28 (i,1) = (L28(i,1)-L28(i-1,1));
    deltaL29 (i,1) = (L29(i,1)-L29(i-1,1));
    deltaL30 (i,1) = (L30(i,1)-L30(i-1,1));
    deltaL31 (i,1) = (L31(i,1)-L31(i-1,1));
    deltaL32 (i,1) = (L32(i,1)-L32(i-1,1));
    deltaL33 (i,1) = (L33(i,1)-L33(i-1,1));
    deltaL34 (i,1) = (L34(i,1)-L34(i-1,1));
    deltaL35 (i,1) = (L35(i,1)-L35(i-1,1));
    deltaL36 (i,1) = (L36(i,1)-L36(i-1,1));
    deltaL37 (i,1) = (L37(i,1)-L37(i-1,1));
    deltaL38 (i,1) = (L38(i,1)-L38(i-1,1));
    deltaL39 (i,1) = (L39(i,1)-L39(i-1,1));
    deltaL40 (i,1) = (L40(i,1)-L40(i-1,1));
end
initialTrainingPeriod = 1000;
success = 0;
trainedNet = Train_with_simple_FF_40Inputs(deltaL1(1:1000,1),deltaL2(1:1000,1),deltaL3(1:1000,1),deltaL4(1:1000,1),
deltaL5(1:1000,1),deltaL6(1:1000,1),deltaL7(1:1000,1),deltaL8(1:1000,1),deltaL9(1:1000,1),deltaL10(1:1000,1),
deltaL11(1:1000,1),deltaL12(1:1000,1),deltaL13(1:1000,1),deltaL14(1:1000,1),deltaL15(1:1000,1),
deltaL16(1:1000,1),deltaL17(1:1000,1),deltaL18(1:1000,1),deltaL19(1:1000,1),deltaL20(1:1000,1),deltaL21(1:1000,1),
deltaL22(1:1000,1),deltaL23(1:1000,1),deltaL24(1:1000,1),deltaL25(1:1000,1),deltaL26(1:1000,1),deltaL27(1:1000,1),
deltaL28(1:1000,1),deltaL29(1:1000,1),deltaL30(1:1000,1),deltaL31(1:1000,1),deltaL32(1:1000,1),deltaL33(1:1000,1),
deltaL34(1:1000,1),deltaL35(1:1000,1),deltaL36(1:1000,1),deltaL37(1:1000,1),deltaL38(1:1000,1),deltaL39(1:1000,1),
deltaL40(1:1000,1),deltaT(1:1000,1));
for (i=1001:1:2000)
    I = [deltaL1( i+1,1),deltaL2( i+1,1),deltaL3( i+1,1),deltaL4( i+1,1),deltaL5( i+1,1),deltaL6( i+1,1),
     deltaL7( i+1,1),deltaL8( i+1,1),deltaL9( i+1,1),deltaL10( i+1,1),deltaL11( i+1,1),deltaL12( i+1,1),
     deltaL13( i+1,1),deltaL14( i+1,1),deltaL15( i+1,1),deltaL16( i+1,1),deltaL17( i+1,1),deltaL18( i+1,1),
     deltaL19( i+1,1),deltaL20( i+1,1),deltaL21( i+1,1),deltaL22( i+1,1),deltaL23( i+1,1),deltaL24( i+1,1),
     deltaL25( i+1,1),deltaL26( i+1,1),deltaL27( i+1,1),deltaL28( i+1,1),deltaL29( i+1,1),deltaL30( i+1,1),
     deltaL31( i+1,1),deltaL32( i+1,1),deltaL33( i+1,1),deltaL34( i+1,1),deltaL35( i+1,1),deltaL36( i+1,1),
     deltaL37( i+1,1),deltaL38( i+1,1),deltaL39( i+1,1),deltaL40( i+1,1)]';
    forecast(i,1) = sim(trainedNet,I);
end
success = 0;
for (i=1001:1:2000)
    if (forecast(i,1) * deltaT(i,1) > 0)
        success = success + 1;
    end
    error(i,1) = forecast(i,1) - deltaT(i,1);
end
successRate = success/1000
rmse = RMSE(error)
nrmse = NRMSE(error)
mae = MAE(error)
mape = MAPE(error)
datatemp = (data(1:2000));
Yt_temp(1,1) = datatemp(1,1)
for (i=2:1:2000)
    Yt_temp(i,1) =  Yt_temp(i-1,1)+forecast(i,1);
end
x_1 = 1:2000;
figure
subplot(3,2,1)
plot(x_1, forecast, '.');
title('PM Return')
ylabel('Return')
subplot(3,2,2)
plot(x_1, deltaT(1:2000,1), '.');
title('Real Return')
ylabel('Return')
subplot(3,2,3:4)
plot(x_1, error, '.');
title('Error')
```

```
ylabel('Return')
TargetIndicator = flipud(TargetIndicator);
subplot(3,2,5:6)
plot(x_1, data(1:2000),'.', x_1, Yt_temp, '.');
title('Real Value Vs. Forecasted Value')
ylabel('Value')

M=[forecast(1001:2000,1) deltaT(1001:2000,1) error(1001:2000,1)];
end
```

```
% test_Train_with_simple_FF_40InputsRol(5, AMEX_Oil_Index, COT_2_YEAR_U_S__T_NOTES_Commercial_Traders_Net,
% COT_2_YEAR_U_S__T_NOTES_Non_Commercial_Traders_Net, COT_2_YEAR_U_S__T_NOTES_Small_Traders_Net,
% COT_CRUDE_OIL__LIGHT_SWEET_Commercial_Traders_Net, COT_CRUDE_OIL__LIGHT_SWEET_Non_Commercial_Traders_Net,
% COT_CRUDE_OIL__LIGHT_SWEET_Small_Traders_Net, COT_Dow_Jones_Commercial_Traders_Net,
% COT_Dow_Jones_Non_Commercial_Traders_Net, COT_Dow_Jones_Small_Traders_Net,
% COT_GOLD_Commercial_Traders_Net, COT_GOLD_Non_Commercial_Traders_Net, COT_GOLD_Small_Traders_Net,
% COT_INTEREST_RATES__FED_FUNDS__Commercial_Traders_Net,
% COT_INTEREST_RATES__FED_FUNDS__Non_Commercial_Traders_Net,
% COT_INTEREST_RATES__FED_FUNDS__Small_Traders_Net, COT_Mini_S_P_500_Commercial_Traders_Net,
% COT_Mini_S_P_500_Non_Commercial_Traders_Net, COT_Mini_S_P_500_Small_Traders_Net,
% COT_SILVER_Commercial_Traders_Net, COT_SILVER_Non_Commercial_Traders_Net,
% COT_SILVER_Small_Traders_Net, COT_S_P_500_STOCK_INDEX_FUTURES_Commercial_Traders_Net,
% COT_S_P_500_STOCK_INDEX_FUTURES_Non_Commercial_Traders_Net,
% COT_S_P_500_STOCK_INDEX_FUTURES_Small_Traders_Net,
% COT_UNLEADED_GASOLINE__NY_Commercial_Traders_Net, COT_UNLEADED_GASOLINE__NY_Non_Commercial_Traders_Net,
% COT_UNLEADED_GASOLINE__NY_Small_Traders_Net, COT_U_S__DOLLAR_INDEX_Commercial_Traders_Net,
% COT_U_S__DOLLAR_INDEX_Non_Commercial_Traders_Net, COT_U_S__DOLLAR_INDEX_Small_Traders_Net,
% Dow_Jones, Dow_Jones_Moving_Average_30_days, Dow_Jones_Transportation, Dow_Jones_Utilities,
% Gold_and_Silver, High__S_P_500, Low__S_P_500, News, Volume_S_P_500, S_P_500)


function M = test_Train_with_simple_FF_40InputsRol(shift,  L1, L2, L3, L4, L5, L6, L7, L8, L9, L10, L11, L12,
L13, L14, L15, L16, L17, L18, L19, L20, L21, L22, L23, L24, L25, L26, L27, L28, L29, L30, L31, L32,
L33, L34, L35, L36, L37, L38, L39, L40, TargetIndicator);
data = flipud(TargetIndicator);
L1 = flipud(L1);
L2 = flipud(L2);
L3 = flipud(L3);
L4 = flipud(L4);
L5 = flipud(L5);
L6 = flipud(L6);
L7 = flipud(L7);
L8 = flipud(L8);
L9 = flipud(L9);
L10 = flipud(L10);
L11 = flipud(L11);
L12 = flipud(L12);
L13 = flipud(L13);
L14 = flipud(L14);
L15 = flipud(L15);
L16 = flipud(L16);
L17 = flipud(L17);
L18 = flipud(L18);
L19 = flipud(L19);
L20 = flipud(L20);
L21 = flipud(L21);
```

```
L22 = flipud(L22);
L23 = flipud(L23);
L24 = flipud(L24);
L25 = flipud(L25);
L26 = flipud(L26);
L27 = flipud(L27);
L28 = flipud(L28);
L29 = flipud(L29);
L30 = flipud(L30);
L31 = flipud(L31);
L32 = flipud(L32);
L33 = flipud(L33);
L34 = flipud(L34);
L35 = flipud(L35);
L36 = flipud(L36);
L37 = flipud(L37);
L38 = flipud(L38);
L39 = flipud(L39);
L40 = flipud(L40);
TargetIndicator = flipud(TargetIndicator);

L1 = reduceNoise(L1, 0.95);
L2 = reduceNoise(L2, 0.95);
L3 = reduceNoise(L3, 0.95);
L4 = reduceNoise(L4, 0.95);
L5 = reduceNoise(L5, 0.95);
L6 = reduceNoise(L6, 0.95);
L7 = reduceNoise(L7, 0.95);
L8 = reduceNoise(L8, 0.95);
L9 = reduceNoise(L9, 0.95);
L10 = reduceNoise(L10, 0.95);
L11 = reduceNoise(L11, 0.95);
L12 = reduceNoise(L12, 0.95);
L13 = reduceNoise(L13, 0.95);
L14 = reduceNoise(L14, 0.95);
L15 = reduceNoise(L15, 0.95);
L16 = reduceNoise(L16, 0.95);
L17 = reduceNoise(L17, 0.95);
L18 = reduceNoise(L18, 0.95);
L19 = reduceNoise(L19, 0.95);
L20 = reduceNoise(L20, 0.95);
L21 = reduceNoise(L21, 0.95);
L22 = reduceNoise(L22, 0.95);
L23 = reduceNoise(L23, 0.95);
L24 = reduceNoise(L24, 0.95);
L25 = reduceNoise(L25, 0.95);
L26 = reduceNoise(L26, 0.95);
L27 = reduceNoise(L27, 0.95);
L28 = reduceNoise(L28, 0.95);
L29 = reduceNoise(L29, 0.95);
L30 = reduceNoise(L30, 0.95);
L31 = reduceNoise(L31, 0.95);
L32 = reduceNoise(L32, 0.95);
L33 = reduceNoise(L33, 0.95);
L34 = reduceNoise(L34, 0.95);
L35 = reduceNoise(L35, 0.95);
L36 = reduceNoise(L36, 0.95);
```

```
L37 = reduceNoise(L37, 0.95);
L38 = reduceNoise(L38, 0.95);
L39 = reduceNoise(L39, 0.95); % news indicators
L40 = reduceNoise(L40, 0.95);
TargetIndicator = reduceNoise(TargetIndicator, 0.95);

boundary=10;
periodLength=length(L1)-boundary;


L1 = ShiftMatrix(L1, shift, periodLength,1);
L2 = ShiftMatrix(L2, shift, periodLength,1);
L3 = ShiftMatrix(L3, shift, periodLength,1);
L4 = ShiftMatrix(L4, shift, periodLength,1);
L5 = ShiftMatrix(L5, shift, periodLength,1);
L6 = ShiftMatrix(L6, shift, periodLength,1);
L7 = ShiftMatrix(L7, shift, periodLength,1);
L8 = ShiftMatrix(L8, shift, periodLength,1);
L9 = ShiftMatrix(L9, shift, periodLength,1);
L10 = ShiftMatrix(L10, shift, periodLength,1);
L11 = ShiftMatrix(L11, shift, periodLength,1);
L12 = ShiftMatrix(L12, shift, periodLength,1);
L13 = ShiftMatrix(L13, shift, periodLength,1);
L14 = ShiftMatrix(L14, shift, periodLength,1);
L15 = ShiftMatrix(L15, shift, periodLength,1);
L16 = ShiftMatrix(L16, shift, periodLength,1);
L17 = ShiftMatrix(L17, shift, periodLength,1);
L18 = ShiftMatrix(L18, shift, periodLength,1);
L19 = ShiftMatrix(L19, shift, periodLength,1);
L20 = ShiftMatrix(L20, shift, periodLength,1);
L21 = ShiftMatrix(L21, shift, periodLength,1);
L22 = ShiftMatrix(L22, shift, periodLength,1);
L23 = ShiftMatrix(L23, shift, periodLength,1);
L24 = ShiftMatrix(L24, shift, periodLength,1);
L25 = ShiftMatrix(L25, shift, periodLength,1);
L26 = ShiftMatrix(L26, shift, periodLength,1);
L27 = ShiftMatrix(L27, shift, periodLength,1);
L28 = ShiftMatrix(L28, shift, periodLength,1);
L29 = ShiftMatrix(L29, shift, periodLength,1);
L30 = ShiftMatrix(L30, shift, periodLength,1);
L31 = ShiftMatrix(L31, shift, periodLength,1);
L32 = ShiftMatrix(L32, shift, periodLength,1);
L33 = ShiftMatrix(L33, shift, periodLength,1);
L34 = ShiftMatrix(L34, shift, periodLength,1);
L35 = ShiftMatrix(L35, shift, periodLength,1);
L36 = ShiftMatrix(L36, shift, periodLength,1);
L37 = ShiftMatrix(L37, shift, periodLength,1);
L38 = ShiftMatrix(L38, shift, periodLength,1);
L39 = ShiftMatrix(L39, 0, periodLength,1); % news is a coincident indicator
L40 = ShiftMatrix(L40, shift, periodLength,1);

deltaL1(1,1) = 0;
deltaL2(1,1) = 0;
deltaL3(1,1) = 0;
deltaL4(1,1) = 0;
deltaL5(1,1) = 0;
deltaL6(1,1) = 0;
```

```
deltaL7(1,1) = 0;
deltaL8(1,1) = 0;
deltaL9(1,1) = 0;
deltaL10(1,1) = 0;
deltaL11(1,1) = 0;
deltaL12(1,1) = 0;
deltaL13(1,1) = 0;
deltaL14(1,1) = 0;
deltaL15(1,1) = 0;
deltaL16(1,1) = 0;
deltaL17(1,1) = 0;
deltaL18(1,1) = 0;
deltaL19(1,1) = 0;
deltaL20(1,1) = 0;
deltaL21(1,1) = 0;
deltaL22(1,1) = 0;
deltaL23(1,1) = 0;
deltaL24(1,1) = 0;
deltaL25(1,1) = 0;
deltaL26(1,1) = 0;
deltaL27(1,1) = 0;
deltaL28(1,1) = 0;
deltaL29(1,1) = 0;
deltaL30(1,1) = 0;
deltaL31(1,1) = 0;
deltaL32(1,1) = 0;
deltaL33(1,1) = 0;
deltaL34(1,1) = 0;
deltaL35(1,1) = 0;
deltaL36(1,1) = 0;
deltaL37(1,1) = 0;
deltaL38(1,1) = 0;
deltaL39(1,1) = 0;
deltaL40(1,1) = 0;

for(i=2:1:periodLength)
    deltaT (i,1) = (TargetIndicator(i,1)-TargetIndicator(i-1,1));
    deltaL1 (i,1) = (L1(i,1)-L1(i-1,1));
    deltaL2 (i,1) = (L2(i,1)-L2(i-1,1));
    deltaL3 (i,1) = (L3(i,1)-L3(i-1,1));
    deltaL4 (i,1) = (L4(i,1)-L4(i-1,1));
    deltaL5 (i,1) = (L5(i,1)-L5(i-1,1));
    deltaL6 (i,1) = (L6(i,1)-L6(i-1,1));
    deltaL7 (i,1) = (L7(i,1)-L7(i-1,1));
    deltaL8 (i,1) = (L8(i,1)-L8(i-1,1));
    deltaL9 (i,1) = (L9(i,1)-L9(i-1,1));
    deltaL10 (i,1) = (L10(i,1)-L10(i-1,1));
    deltaL11 (i,1) = (L11(i,1)-L11(i-1,1));
    deltaL12 (i,1) = (L12(i,1)-L12(i-1,1));
    deltaL13 (i,1) = (L13(i,1)-L13(i-1,1));
    deltaL14 (i,1) = (L14(i,1)-L14(i-1,1));
    deltaL15 (i,1) = (L15(i,1)-L15(i-1,1));
    deltaL16 (i,1) = (L16(i,1)-L16(i-1,1));
    deltaL17 (i,1) = (L17(i,1)-L17(i-1,1));
    deltaL18 (i,1) = (L18(i,1)-L18(i-1,1));
    deltaL19 (i,1) = (L19(i,1)-L19(i-1,1));
    deltaL20 (i,1) = (L20(i,1)-L20(i-1,1));
    deltaL21 (i,1) = (L21(i,1)-L21(i-1,1));
    deltaL22 (i,1) = (L22(i,1)-L22(i-1,1));
    deltaL23 (i,1) = (L23(i,1)-L23(i-1,1));
    deltaL24 (i,1) = (L24(i,1)-L24(i-1,1));
    deltaL25 (i,1) = (L25(i,1)-L25(i-1,1));
    deltaL26 (i,1) = (L26(i,1)-L26(i-1,1));
    deltaL27 (i,1) = (L27(i,1)-L27(i-1,1));
    deltaL28 (i,1) = (L28(i,1)-L28(i-1,1));
    deltaL29 (i,1) = (L29(i,1)-L29(i-1,1));
    deltaL30 (i,1) = (L30(i,1)-L30(i-1,1));
    deltaL31 (i,1) = (L31(i,1)-L31(i-1,1));
    deltaL32 (i,1) = (L32(i,1)-L32(i-1,1));
    deltaL33 (i,1) = (L33(i,1)-L33(i-1,1));
    deltaL34 (i,1) = (L34(i,1)-L34(i-1,1));
    deltaL35 (i,1) = (L35(i,1)-L35(i-1,1));
    deltaL36 (i,1) = (L36(i,1)-L36(i-1,1));
    deltaL37 (i,1) = (L37(i,1)-L37(i-1,1));
    deltaL38 (i,1) = (L38(i,1)-L38(i-1,1));
    deltaL39 (i,1) = (L39(i,1)-L39(i-1,1));
    deltaL40 (i,1) = (L40(i,1)-L40(i-1,1));
end

initialTrainingPeriod = 1000;
success = 0;
for (i=1001:1:2000)
trainedNet = Train_with_simple_FF_40Inputs(deltaL1(1:1000,1),deltaL2(1:1000,1),deltaL3(1:1000,1),deltaL4(1:1000,1),
deltaL5(1:1000,1),deltaL6(1:1000,1),deltaL7(1:1000,1),deltaL8(1:1000,1),deltaL9(1:1000,1),deltaL10(1:1000,1),
deltaL11(1:1000,1),deltaL12(1:1000,1),deltaL13(1:1000,1),deltaL14(1:1000,1),deltaL15(1:1000,1),deltaL16(1:1000,1),
deltaL17(1:1000,1),deltaL18(1:1000,1),deltaL19(1:1000,1),deltaL20(1:1000,1),deltaL21(1:1000,1),deltaL22(1:1000,1),
deltaL23(1:1000,1),deltaL24(1:1000,1),deltaL25(1:1000,1),deltaL26(1:1000,1),deltaL27(1:1000,1),deltaL28(1:1000,1),
deltaL29(1:1000,1),deltaL30(1:1000,1),deltaL31(1:1000,1),deltaL32(1:1000,1),deltaL33(1:1000,1),deltaL34(1:1000,1),
deltaL35(1:1000,1),deltaL36(1:1000,1),deltaL37(1:1000,1),deltaL38(1:1000,1),deltaL39(1:1000,1),deltaL40(1:1000,1),
deltaT(1:1000,1));
  I = [deltaL1( i+1,1),deltaL2( i+1,1),deltaL3( i+1,1),deltaL4( i+1,1),deltaL5( i+1,1),deltaL6( i+1,1),
   deltaL7( i+1,1),deltaL8( i+1,1),deltaL9( i+1,1),deltaL10( i+1,1),deltaL11( i+1,1),deltaL12( i+1,1),
   deltaL13( i+1,1),deltaL14( i+1,1),deltaL15( i+1,1),deltaL16( i+1,1),deltaL17( i+1,1),deltaL18( i+1,1),
   deltaL19( i+1,1),deltaL20( i+1,1),deltaL21( i+1,1),deltaL22( i+1,1),deltaL23( i+1,1),deltaL24( i+1,1),
   deltaL25( i+1,1),deltaL26( i+1,1),deltaL27( i+1,1),deltaL28( i+1,1),deltaL29( i+1,1),deltaL30( i+1,1),
   deltaL31( i+1,1),deltaL32( i+1,1),deltaL33( i+1,1),deltaL34( i+1,1),deltaL35( i+1,1),deltaL36( i+1,1),
   deltaL37( i+1,1),deltaL38( i+1,1),deltaL39( i+1,1),deltaL40( i+1,1)]';
  forecast(i,1) = sim(trainedNet,I);
end
success = 0;
for (i=1001:1:2000)
    if (forecast(i,1) * deltaT(i,1) > 0)
        success = success + 1;
    end
    error(i,1) = forecast(i,1) - deltaT(i,1);
end
successRate = success/1000
rmse = RMSE(error)
nrmse = NRMSE(error)
mae = MAE(error)
mape = MAPE(error)
datatemp = (data(1:2000));
Yt_temp(1,1) = datatemp(1,1)
for (i=2:1:2000)
```

```
    Yt_temp(i,1) =  Yt_temp(i-1,1)+forecast(i,1);
end
x_1 = 1:2000;
figure
subplot(3,2,1)
plot(x_1, forecast, '.');
title('PM Return')
ylabel('Return')
subplot(3,2,2)
plot(x_1, deltaT(1:2000,1), '.');
title('Real Return')
ylabel('Return')
subplot(3,2,3:4)
plot(x_1, error, '.');
title('Error')
ylabel('Return')
TargetIndicator = flipud(TargetIndicator);
subplot(3,2,5:6)
plot(x_1, data(1:2000),'.', x_1, Yt_temp, '.');
title('Real Value Vs. Forecasted Value')
ylabel('Value')

M=[forecast(1001:2000,1) deltaT(1001:2000,1) error(1001:2000,1)];
end
```

8.9 Matlab Codes for the M-GNN model

```
% Method used in our multivariate approach with the application to neural
% netwroks. The indicators are the same Gately has used in his book plus an
% additional indicator that is coincident instead of being leading.

function trainedNet = train_Gately(L1,L2,L3,L4,L5,L6,L7,L8,L9,L10,T);

I = [(L1)';(L2)';(L3)';(L4)';(L5)';(L6)';(L7)';(L8)';(L9)';(L10)'];
T = (T)';
net = newff([min(L1) max(L1);min(L2) max(L2);min(L3) max(L3);min(L4) max(L4);min(L5) max(L5);min(L6) max(L6);
min(L7) max(L7);min(L8) max(L8);min(L9) max(L9);min(L10) max(L10)],[10 15 1],{'tansig' 'tansig' 'tansig'},'traingd');

net = init(net);
net.trainParam.epochs = 100;
net = train(net,I,T);
% net.trainParam.show = 50;
net.trainParam.lr = 0.30;
net.trainParam.mc = 0.65;
net.trainParam.goal = 1e-3;
[net,tr]=train(net,I,T);

trainedNet = net;
end
```

```
% Method used in our multivariate approach with the application to neural
% netwroks. The indicators are the same Gately has used in his book plus an
% additional indicator that is coincident instead of being leading.
% test_train_GatelyCut(10,  Low__S_P_500, High__S_P_500, S_P_500, Volume_S_P_500,
% Dow_Jones_Moving_Average_30_days, Dow_Jones, Dow_Jones_Transportation,
% Dow_Jones_Utilities, AMEX_Oil_Index, Gold_and_Silver, S_P_500);

function M = test_train_GatelyCut(shift,  L1, L2, L3, L4, L5, L6, L7, L8, L9, L10, TargetIndicator);

data = flipud(TargetIndicator)
L1 = flipud(L1);
L2 = flipud(L2);
L3 = flipud(L3);
L4 = flipud(L4);
L5 = flipud(L5);
L6 = flipud(L6);
L7 = flipud(L7);
L8 = flipud(L8);
L9 = flipud(L9);
L10 = flipud(L10);
TargetIndicator = flipud(TargetIndicator);
periodLength=length(L1)-shift
L1 = ShiftMatrix(L1, shift, periodLength,1);
L2 = ShiftMatrix(L2, shift, periodLength,1);
L3 = ShiftMatrix(L3, shift, periodLength,1);
L4 = ShiftMatrix(L4, shift, periodLength,1);
L5 = ShiftMatrix(L5, shift, periodLength,1);
L6 = ShiftMatrix(L6, shift, periodLength,1);
```

```
L7 = ShiftMatrix(L7, shift, periodLength,1);
L8 = ShiftMatrix(L8, shift, periodLength,1);
L9 = ShiftMatrix(L9, shift, periodLength,1);
L10 = ShiftMatrix(L10, shift, periodLength,1);
deltaL1(1,1) = 0;
deltaL2(1,1) = 0;
deltaL3(1,1) = 0;
deltaL4(1,1) = 0;
deltaL5(1,1) = 0;
deltaL6(1,1) = 0;
deltaL7(1,1) = 0;
deltaL8(1,1) = 0;
deltaL9(1,1) = 0;
deltaL10(1,1) = 0;
deltaT(1,1) = 0;
for(i=2:1:periodLength)
    deltaT (i,1) = (TargetIndicator(i,1)-TargetIndicator(i-1,1))/TargetIndicator(i-1,1);
    deltaL1 (i,1) = (L1(i,1)-L1(i-1,1))/L1(i-1,1);
    deltaL2 (i,1) = (L2(i,1)-L2(i-1,1))/L2(i-1,1);
    deltaL3 (i,1) = (L3(i,1)-L3(i-1,1))/L3(i-1,1);
    deltaL4 (i,1) = (L4(i,1)-L4(i-1,1))/L4(i-1,1);
    deltaL5 (i,1) = (L5(i,1)-L5(i-1,1))/L5(i-1,1);
    deltaL6 (i,1) = (L6(i,1)-L6(i-1,1))/L6(i-1,1);
    deltaL7 (i,1) = (L7(i,1)-L7(i-1,1))/L7(i-1,1);
    deltaL8 (i,1) = (L8(i,1)-L8(i-1,1))/L8(i-1,1);
    deltaL9 (i,1) = (L9(i,1)-L9(i-1,1))/L9(i-1,1);
    deltaL10 (i,1) = (L10(i,1)-L10(i-1,1))/L10(i-1,1);
end
[L1n,minL1,maxL1,tn,mint,maxt] = premnmx(deltaL1,deltaT);
[L2n,minL2,maxL2,tn,mint,maxt] = premnmx(deltaL2,deltaT);
[L3n,minL3,maxL3,tn,mint,maxt] = premnmx(deltaL3,deltaT);
[L4n,minL4,maxL4,tn,mint,maxt] = premnmx(deltaL4,deltaT);
[L5n,minL5,maxL5,tn,mint,maxt] = premnmx(deltaL5,deltaT);
[L6n,minL6,maxL6,tn,mint,maxt] = premnmx(deltaL6,deltaT);
[L7n,minL7,maxL7,tn,mint,maxt] = premnmx(deltaL7,deltaT);
[L8n,minL8,maxL8,tn,mint,maxt] = premnmx(deltaL8,deltaT);
[L9n,minL9,maxL9,tn,mint,maxt] = premnmx(deltaL9,deltaT);
[L10n,minL10,maxL10,tn,mint,maxt] = premnmx(deltaL10,deltaT);
a= [L1n L2n L3n L4n L5n L6n L7n L8n L9n L10n];
b= [L1n, L2n, L3n, L4n, L5n, L6n, L7n, L8n, L9n, L10n];
size(a);
initialTrainingPeriod = 1000;
itp = initialTrainingPeriod;
success = 0;
trainedNet = train_Gately(L1n(1:1000,1), L2n(1:1000,1), L3n(1:1000,1), L4n(1:1000,1), L5n(1:1000,1),
L6n(1:1000,1), L7n(1:1000,1), L8n(1:1000,1), L9n(1:1000,1), L10n(1:1000,1), tn(1:1000,1));
for(i=1:1:1001)
    forecasted(i,1)=0;
end
for (i=initialTrainingPeriod:1:2000)
    I = [L1n(i,1), L2n(i,1), L3n(i,1), L4n(i,1), L5n(i,1), L6n(i,1), L7n(i,1), L8n(i,1), L9n(i,1), L10n(i,1)]';
    forecastedn(i+1,1) = sim(trainedNet,I);
    forecasted(i+1,1) = postmnmx(forecastedn(i,1),mint,maxt);
end
visualTest = [forecasted(1:2000,1)  deltaT(1:2000,1)]
success = 0;
for (i=1001:1:2000)
```

```
    if (forecasted(i,1) * deltaT(i,1) > 0)
        success = success + 1;
    end
    error(i,1) = forecasted(i,1) - deltaT(i,1);

end
successRate = success/1000
rmse = RMSE(error(1001:2000,1))
nrmse = NRMSE(error(1001:2000,1))
mae = MAE(error(1001:2000,1))
mape = MAPE(error(1001:2000,1))
datatemp = (data(1:2000,1));
Yt_temp(1,1) = datatemp(1,1);
for (i=2:1:2000)
    Yt_temp(i,1) =  Yt_temp(i-1,1)+forecasted(i,1);
end
x_1 = 1:2000;
size(forecasted)
figure
subplot(3,2,1)
plot(x_1, forecasted(1:2000,1), '.');
title('PM Return')
ylabel('Return')
subplot(3,2,2)
plot(x_1, deltaT(1:2000,1), '.');
title('Real Return')
ylabel('Return')
subplot(3,2,3:4)
plot(x_1, error, '.');
title('Error')
ylabel('Return')
subplot(3,2,5:6)
plot(x_1, data(1:2000),'.', x_1, Yt_temp, '.');
title('Real Value Vs. Forecasted Value')
ylabel('Value')
M=[forecasted(1001:2000,1) deltaT(1001:2000,1) error(1001:2000,1)];
end
```

```
% Method used in our multivariate approach with the application to neural
% netwroks. The indicators are the same Gately has used in his book plus an
% additional indicator that is coincident instead of being leading.
% test_train_GatelyRol(10,  Low__S_P_500, High__S_P_500, S_P_500, Volume_S_P_500,
% Dow_Jones_Moving_Average_30_days, Dow_Jones, % Dow_Jones_Transportation,
% Dow_Jones_Utilities, AMEX_Oil_Index, Gold_and_Silver, S_P_500);

function M = test_train_GatelyRol(shift,  L1, L2, L3, L4, L5, L6, L7, L8, L9, L10, TargetIndicator);

data = flipud(TargetIndicator)
L1 = flipud(L1);
L2 = flipud(L2);
L3 = flipud(L3);
L4 = flipud(L4);
L5 = flipud(L5);
L6 = flipud(L6);
L7 = flipud(L7);
```

```
L8 = flipud(L8);
L9 = flipud(L9);
L10 = flipud(L10);
TargetIndicator = flipud(TargetIndicator);
periodLength=length(L1)-shift
L1 = ShiftMatrix(L1, shift, periodLength,1);
L2 = ShiftMatrix(L2, shift, periodLength,1);
L3 = ShiftMatrix(L3, shift, periodLength,1);
L4 = ShiftMatrix(L4, shift, periodLength,1);
L5 = ShiftMatrix(L5, shift, periodLength,1);
L6 = ShiftMatrix(L6, shift, periodLength,1);
L7 = ShiftMatrix(L7, shift, periodLength,1);
L8 = ShiftMatrix(L8, shift, periodLength,1);
L9 = ShiftMatrix(L9, shift, periodLength,1);
L10 = ShiftMatrix(L10, shift, periodLength,1);
deltaL1(1,1) = 0;
deltaL2(1,1) = 0;
deltaL3(1,1) = 0;
deltaL4(1,1) = 0;
deltaL5(1,1) = 0;
deltaL6(1,1) = 0;
deltaL7(1,1) = 0;
deltaL8(1,1) = 0;
deltaL9(1,1) = 0;
deltaL10(1,1) = 0;
deltaT(1,1) = 0;
for(i=2:1:periodLength)
    deltaT (i,1) = (TargetIndicator(i,1)-TargetIndicator(i-1,1))/TargetIndicator(i-1,1);
    deltaL1 (i,1) = (L1(i,1)-L1(i-1,1))/L1(i-1,1);
    deltaL2 (i,1) = (L2(i,1)-L2(i-1,1))/L2(i-1,1);
    deltaL3 (i,1) = (L3(i,1)-L3(i-1,1))/L3(i-1,1);
    deltaL4 (i,1) = (L4(i,1)-L4(i-1,1))/L4(i-1,1);
    deltaL5 (i,1) = (L5(i,1)-L5(i-1,1))/L5(i-1,1);
    deltaL6 (i,1) = (L6(i,1)-L6(i-1,1))/L6(i-1,1);
    deltaL7 (i,1) = (L7(i,1)-L7(i-1,1))/L7(i-1,1);
    deltaL8 (i,1) = (L8(i,1)-L8(i-1,1))/L8(i-1,1);
    deltaL9 (i,1) = (L9(i,1)-L9(i-1,1))/L9(i-1,1);
    deltaL10 (i,1) = (L10(i,1)-L10(i-1,1))/L10(i-1,1);
end
% normalisation
[L1n,minL1,maxL1,tn,mint,maxt] = premnmx(deltaL1,deltaT);
[L2n,minL2,maxL2,tn,mint,maxt] = premnmx(deltaL2,deltaT);
[L3n,minL3,maxL3,tn,mint,maxt] = premnmx(deltaL3,deltaT);
[L4n,minL4,maxL4,tn,mint,maxt] = premnmx(deltaL4,deltaT);
[L5n,minL5,maxL5,tn,mint,maxt] = premnmx(deltaL5,deltaT);
[L6n,minL6,maxL6,tn,mint,maxt] = premnmx(deltaL6,deltaT);
[L7n,minL7,maxL7,tn,mint,maxt] = premnmx(deltaL7,deltaT);
[L8n,minL8,maxL8,tn,mint,maxt] = premnmx(deltaL8,deltaT);
[L9n,minL9,maxL9,tn,mint,maxt] = premnmx(deltaL9,deltaT);
[L10n,minL10,maxL10,tn,mint,maxt] = premnmx(deltaL10,deltaT);
a= [L1n L2n L3n L4n L5n L6n L7n L8n L9n L10n];
b= [L1n, L2n, L3n, L4n, L5n, L6n, L7n, L8n, L9n, L10n];
size(a);
initialTrainingPeriod = 1000;
itp = initialTrainingPeriod;
success = 0;
for(i=1:1:1001)
```

```
    forecasted(i,1)=0;
end
for (i=initialTrainingPeriod:1:2000)
trainedNet = train_Gately(L1n(1:i,1), L2n(1:i,1), L3n(1:i,1), L4n(1:i,1), L5n(1:i,1),
L6n(1:i,1), L7n(1:i,1), L8n(1:i,1), L9n(1:i,1), L10n(1:i,1), tn(1:i,1));
    I = [L1n(i,1), L2n(i,1), L3n(i,1), L4n(i,1), L5n(i,1), L6n(i,1), L7n(i,1), L8n(i,1), L9n(i,1),
     L10n(i,1)]';
    forecastedn(i+1,1) = sim(trainedNet,I);
    forecasted(i+1,1) = postmnmx(forecastedn(i,1),mint,maxt);
end
visualTest = [forecasted(1:2000,1)  deltaT(1:2000,1)]
success = 0;
for (i=1001:1:2000)
    if (forecasted(i,1) * deltaT(i,1) > 0)
        success = success + 1;
    end
    error(i,1) = forecasted(i,1) - deltaT(i,1);
end
successRate = success/1000
rmse = RMSE(error(1001:2000,1))
nrmse = NRMSE(error(1001:2000,1))
mae = MAE(error(1001:2000,1))
mape = MAPE(error(1001:2000,1))
datatemp = (data(1:2000,1));
Yt_temp(1,1) = datatemp(1,1);
for (i=2:1:2000)
    Yt_temp(i,1) =  Yt_temp(i-1,1)+forecasted(i,1);
end
x_1 = 1:2000;
size(forecasted)
figure
subplot(3,2,1)
plot(x_1, forecasted(1:2000,1), '.');
title('PM Return')
ylabel('Return')
subplot(3,2,2)
plot(x_1, deltaT(1:2000,1), '.');
title('Real Return')
ylabel('Return')
subplot(3,2,3:4)
plot(x_1, error, '.');
title('Error')
ylabel('Return')
subplot(3,2,5:6)
plot(x_1, data(1:2000),'.', x_1, Yt_temp, '.');
title('Real Value Vs. Forecasted Value')
ylabel('Value')
M=[forecasted(1001:2000,1) deltaT(1001:2000,1) error(1001:2000,1)];
end
```

8.10 Matlab Codes for the M-BNN model

```
% Method used in our multivariate approach with the application to neural
% netwroks. The indicators are the same Gately has used in his book plus an
% additional indicator that is coincident instead of being leading.
% train_Final_BetterDesign(Dow_Jones, News,
% COT_S_P_500_STOCK_INDEX_FUTURES_Commercial_Traders_Net,
% COT_S_P_500_STOCK_INDEX_FUTURES_Non_Commercial_Traders_Net,
% Volume_S_P_500, Dow_Jones_Transportation, Dow_Jones_Utilities,
% AMEX_Oil_Index, Gold_and_Silver, COT_Mini_S_P_500_Non_Commercial_Traders_Net,
% COT_Mini_S_P_500_Commercial_Traders_Net, S_P_500)

function trainedNet = train_Final_BetterDesign(deltaL1,deltaL2,deltaL3,deltaL4,deltaL5,deltaL6,
deltaL7,deltaL8,deltaL9,deltaL10,deltaL11,deltaL12,deltaL13,deltaT);

I = [transpose(deltaL1);transpose(deltaL2);transpose(deltaL3);transpose(deltaL4);transpose(deltaL5);
transpose(deltaL6);transpose(deltaL7);transpose(deltaL8);transpose(deltaL9);transpose(deltaL10);
transpose(deltaL11);transpose(deltaL12);transpose(deltaL13)];
T = transpose(deltaT);

trainingStyle = 'trainscg'; % Resilient Backpropagation
net = newff([min(deltaL1) max(deltaL1);min(deltaL2) max(deltaL2);min(deltaL3) max(deltaL3);
min(deltaL4) max(deltaL4);min(deltaL5) max(deltaL5);min(deltaL6) max(deltaL6);
min(deltaL7) max(deltaL7);min(deltaL8) max(deltaL8);min(deltaL9) max(deltaL9);
min(deltaL10) max(deltaL10);min(deltaL11) max(deltaL11);min(deltaL12) max(deltaL12);
min(deltaL13) max(deltaL13)],[10 15 1],{'tansig' 'tansig' 'tansig'},trainingStyle);
net = init(net);
net.trainParam.epochs = 300;
net = train(net,I,T);
net.trainParam.show =1000;
net.trainParam.lr = 0.35;
% net.trainParam.mc = 0.65;
net.trainParam.goal = 1e-3;
[net,tr]=train(net,I,T);
trainedNet = net;
end
```

```
% Method used in our multivariate approach with the application to neural
% netwroks. The indicators are the same Gately has used in his book plus an
% additional indicator that is coincident instead of being leading.
% test_Final_BetterDesignCut(Dow_Jones, News, COT_S_P_500_STOCK_INDEX_FUTURES_Commercial_Traders_Net,
% COT_S_P_500_STOCK_INDEX_FUTURES_Non_Commercial_Traders_Net, Volume_S_P_500,
% Dow_Jones_Transportation, Dow_Jones_Utilities, AMEX_Oil_Index, Gold_and_Silver,
% COT_Mini_S_P_500_Non_Commercial_Traders_Net, COT_Mini_S_P_500_Commercial_Traders_Net, S_P_500);

function M = test_Final_BetterDesignCut(L1, L2, L3, L4, L5, L6, L7, L8, L9, L10, L11, TargetIndicator);
data = flipud(TargetIndicator);
L1 = flipud(L1);
L2 = flipud(L2);
L3 = flipud(L3);
L4 = flipud(L4);
L5 = flipud(L5);
L6 = flipud(L6);
```

```
L7 = flipud(L7);
L8 = flipud(L8);
L9 = flipud(L9);
L10 = flipud(L10);
L11 = flipud(L11);
L12 = flipud(TargetIndicator); %SP day-1
L13 = flipud(TargetIndicator); %SP day-2
TargetIndicator = flipud(TargetIndicator);
shiftL1 = 5;
shiftL2 = 0; % news indicator is a coincident indicator.
shiftL3 = 5;
shiftL4 = 5;
shiftL5 = 5;
shiftL6 = 5;
shiftL7 = 5;
shiftL8 = 5;
shiftL9 = 5;
shiftL10 = shiftForMaxCorr(TargetIndicator, L10, 1000);
shiftL11 = shiftForMaxCorr(TargetIndicator, L11, 1000);
shiftL12 = 1;
shiftL13 = 2;
L1 = reduceNoise(L1, 0.05);  % the noise reduction for the Dow Jones is modified to
% ressemble a 30 day moving average
% L2 = reduceNoise(L2, 0.95); % no noise reduction for L2
L3 = reduceNoise(L3, 0.95);
L4 = reduceNoise(L4, 0.95);
L5 = reduceNoise(L5, 0.95);
L6 = reduceNoise(L6, 0.95);
L7 = reduceNoise(L7, 0.95);
L8 = reduceNoise(L8, 0.95);
L9 = reduceNoise(L9, 0.95);
L10 = reduceNoise(L10, 0.95);
L11 = reduceNoise(L11, 0.95);
L12 = reduceNoise(L12, 0.95);
L13 = reduceNoise(L13, 0.95);
TargetIndicator = reduceNoise(TargetIndicator, 0.95);
maxShift = 15;
periodLength=2100;
L1 = ShiftMatrix(L1, shiftL1, periodLength,1);
L2 = ShiftMatrix(L2, shiftL2, periodLength,1);
L3 = ShiftMatrix(L3, shiftL3, periodLength,1);
L4 = ShiftMatrix(L4, shiftL4, periodLength,1);
L5 = ShiftMatrix(L5, shiftL5, periodLength,1);
L6 = ShiftMatrix(L6, shiftL6, periodLength,1);
L7 = ShiftMatrix(L7, shiftL7, periodLength,1);
L8 = ShiftMatrix(L8, shiftL8, periodLength,1);
L9 = ShiftMatrix(L9, shiftL9, periodLength,1);
L10 = ShiftMatrix(L10, shiftL10, periodLength,1);
L11 = ShiftMatrix(L11, shiftL11, periodLength,1);
L12 = ShiftMatrix(L12, shiftL12, periodLength,1);
L13 = ShiftMatrix(L13, shiftL13, periodLength,1);
deltaL1(1,1) = 0;
deltaL2(1,1) = 0;
deltaL3(1,1) = 0;
deltaL4(1,1) = 0;
deltaL5(1,1) = 0;
deltaL6(1,1) = 0;
```

```
deltaL7(1,1) = 0;
deltaL8(1,1) = 0;
deltaL9(1,1) = 0;
deltaL10(1,1) = 0;
deltaL11(1,1) = 0;
deltaL12(1,1) = 0;
deltaL13(1,1) = 0;
deltaT(1,1) = 0;
for(i=2:1:periodLength)
    deltaT (i,1) = (TargetIndicator(i,1)-TargetIndicator(i-1,1))/TargetIndicator(i-1,1);
    deltaL1 (i,1) = (L1(i,1)-L1(i-1,1))/L1(i-1,1);
    if(i<1001)
        deltaL2 (i,1) = (L2(i,1)-L2(i-1,1)); %news cant devide by 0
    else
        deltaL2 (i,1)= 0; % since news in coincident we have to assume = 0
    end
    deltaL3 (i,1) = (L3(i,1)-L3(i-1,1))/L3(i-1,1);
    deltaL4 (i,1) = (L4(i,1)-L4(i-1,1))/L4(i-1,1);
    deltaL5 (i,1) = (L5(i,1)-L5(i-1,1))/L5(i-1,1);
    deltaL6 (i,1) = (L6(i,1)-L6(i-1,1))/L6(i-1,1);
    deltaL7 (i,1) = (L7(i,1)-L7(i-1,1))/L7(i-1,1);
    deltaL8 (i,1) = (L8(i,1)-L8(i-1,1))/L8(i-1,1);
    deltaL9 (i,1) = (L9(i,1)-L9(i-1,1))/L9(i-1,1);
    deltaL10 (i,1) = (L10(i,1)-L10(i-1,1))/L10(i-1,1);
    deltaL11 (i,1) = (L11(i,1)-L11(i-1,1))/L11(i-1,1);
    deltaL12 (i,1) = (L12(i,1)-L12(i-1,1))/L12(i-1,1);
    deltaL13 (i,1) = (L12(i,1)-L12(i-1,1))/L13(i-1,1);

    tn (i,1) = (TargetIndicator(i,1)-TargetIndicator(i-1,1))/TargetIndicator(i-1,1);
    L1n (i,1) = (L1(i,1)-L1(i-1,1))/L1(i-1,1);
    L2n (i,1) = (L2(i,1)-L2(i-1,1)); %news cant devide by 0
    L3n (i,1) = (L3(i,1)-L3(i-1,1))/L3(i-1,1);
    L4n (i,1) = (L4(i,1)-L4(i-1,1))/L4(i-1,1);
    L5n (i,1) = (L5(i,1)-L5(i-1,1))/L5(i-1,1);
    L6n (i,1) = (L6(i,1)-L6(i-1,1))/L6(i-1,1);
    L7n (i,1) = (L7(i,1)-L7(i-1,1))/L7(i-1,1);
    L8n (i,1) = (L8(i,1)-L8(i-1,1))/L8(i-1,1);
    L9n (i,1) = (L9(i,1)-L9(i-1,1))/L9(i-1,1);
    L10n (i,1) = (L10(i,1)-L10(i-1,1))/L10(i-1,1);
    L11n (i,1) = (L11(i,1)-L11(i-1,1))/L11(i-1,1);
    L12n (i,1) = (L12(i,1)-L12(i-1,1))/L12(i-1,1);
    L13n (i,1) = (L12(i,1)-L12(i-1,1))/L13(i-1,1);
end
tn(1:1000, 1) = forgettingFactor(tn(1:1000, 1),.9999999);
L1n(1:1000, 1) = forgettingFactor(L1n(1:1000, 1),.9999999);
L2n(1:1000, 1) = forgettingFactor(L2n(1:1000, 1),.9999999);
L3n(1:1000, 1) = forgettingFactor(L3n(1:1000, 1),.9999999);
L4n(1:1000, 1) = forgettingFactor(L4n(1:1000, 1),.9999999);
L5n(1:1000, 1) = forgettingFactor(L5n(1:1000, 1),.9999999);
L6n(1:1000, 1) = forgettingFactor(L6n(1:1000, 1),.9999999);
L7n(1:1000, 1) = forgettingFactor(L7n(1:1000, 1),.9999999);
L8n(1:1000, 1) = forgettingFactor(L8n(1:1000, 1),.9999999);
L9n(1:1000, 1) = forgettingFactor(L9n(1:1000, 1),.9999999);
L10n(1:1000, 1) = forgettingFactor(L10n(1:1000, 1),.9999999);
L11n(1:1000, 1) = forgettingFactor(L11n(1:1000, 1),.9999999);
L12n(1:1000, 1) = forgettingFactor(L12n(1:1000, 1),.9999999);
L13n(1:1000, 1) = forgettingFactor(L13n(1:1000, 1),.9999999);
```

```
initialTrainingPeriod = 1000;
itp = initialTrainingPeriod;
success = 0;
trainedNet = train_Final_BetterDesign(L1n(1:1000,1), L2n(1:1000,1), L3n(1:1000,1),
L4n(1:1000,1), L5n(1:1000,1), L6n(1:1000,1), L7n(1:1000,1), L8n(1:1000,1),
L9n(1:1000,1), L10n(1:1000,1), L11n(1:1000,1), L12n(1:1000,1), L13n(1:1000,1),
tn(1:1000,1));
for(i=1:1:1001)
    forecasted(i,1)=0;
end
for (i=initialTrainingPeriod:1:2000)
    I = [L1n(i,1), L2n(i,1), L3n(i,1), L4n(i,1), L5n(i,1), L6n(i,1), L7n(i,1), L8n(i,1),
     L9n(i,1), L10n(i,1), L11n(i,1), L12n(i,1), L13n(i,1)]';
    forecastedn(i+1,1) = sim(trainedNet,I);
    forecasted(i+1,1) = forecastedn(i,1);
end
success = 0;
for (i=1001:1:2000)
    if (forecasted(i,1) * deltaT(i,1) > 0)
        success = success + 1;
    end
    error(i,1) = forecasted(i,1) - deltaT(i,1);
end
successRate = success/1000
rmse = RMSE(error(1001:2000,1))
nrmse = NRMSE(error(1001:2000,1))
mae = MAE(error(1001:2000,1))
mape = MAPE(error(1001:2000,1))
datatemp = (data(1:2000,1));
Yt_temp(1,1) = datatemp(1,1);
for (i=2:1:2000)
    Yt_temp(i,1) =  Yt_temp(i-1,1)+forecasted(i,1);
end
x_1 = 1:2000;
size(forecasted)
figure
subplot(3,2,1)
plot(x_1, forecasted(1:2000,1), '.');
title('PM Return')
ylabel('Return')
subplot(3,2,2)
plot(x_1, deltaT(1:2000,1), '.');
title('Real Return')
ylabel('Return')
subplot(3,2,3:4)
plot(x_1, error, '.');
title('Error')
ylabel('Return')
subplot(3,2,5:6)
plot(x_1, data(1:2000),'.', x_1, Yt_temp, '.');
title('Real Value Vs. Forecasted Value')
ylabel('Value')
M=[forecasted(1001:2000,1) deltaT(1001:2000,1) error(1001:2000,1)];
end
```

```
% Method used in our multivariate approach with the application to neural
% netwroks. The indicators are the same Gately has used in his book plus an
% additional indicator that is coincident instead of being leading.
% test_Final_BetterDesignRol(Dow_Jones, News, COT_S_P_500_STOCK_INDEX_FUTURES_Commercial_Traders_Net,
% COT_S_P_500_STOCK_INDEX_FUTURES_Non_Commercial_Traders_Net, Volume_S_P_500, Dow_Jones_Transportation,
% Dow_Jones_Utilities, AMEX_Oil_Index, Gold_and_Silver, COT_Mini_S_P_500_Non_Commercial_Traders_Net,
% COT_Mini_S_P_500_Commercial_Traders_Net, S_P_500);

function M = test_Final_BetterDesignRol(L1, L2, L3, L4, L5, L6, L7, L8, L9, L10, L11, TargetIndicator);

data = flipud(TargetIndicator);

L1 = flipud(L1);
L2 = flipud(L2);
L3 = flipud(L3);
L4 = flipud(L4);
L5 = flipud(L5);
L6 = flipud(L6);
L7 = flipud(L7);
L8 = flipud(L8);
L9 = flipud(L9);
L10 = flipud(L10);
L11 = flipud(L11);
L12 = flipud(TargetIndicator); %SP day-1
L13 = flipud(TargetIndicator); %SP day-2
TargetIndicator = flipud(TargetIndicator);

shiftL1 = 5;
shiftL2 = 0; % news indicator is a coincident indicator.
shiftL3 = 5;
shiftL4 = 5;
shiftL5 = 5;
shiftL6 = 5;
shiftL7 = 5;
shiftL8 = 5;
shiftL9 = 5;
shiftL10 = shiftForMaxCorr(TargetIndicator, L10, 1000);
shiftL11 = shiftForMaxCorr(TargetIndicator, L11, 1000);
shiftL12 = 1;
shiftL13 = 2;

L1 = reduceNoise(L1, 0.05); % the noise reduction for the Dow Jones is modified to ressemble a 30 day moving average
% L2 = reduceNoise(L2, 0.95);
L3 = reduceNoise(L3, 0.95);
L4 = reduceNoise(L4, 0.95);
L5 = reduceNoise(L5, 0.95);
L6 = reduceNoise(L6, 0.95);
L7 = reduceNoise(L7, 0.95);
L8 = reduceNoise(L8, 0.95);
L9 = reduceNoise(L9, 0.95);
L10 = reduceNoise(L10, 0.95);
L11 = reduceNoise(L11, 0.95);
L12 = reduceNoise(L12, 0.95);
L13 = reduceNoise(L13, 0.95);
TargetIndicator = reduceNoise(TargetIndicator, 0.95);

maxShift = 15;
```

```
periodLength=2100;

L1 = ShiftMatrix(L1, shiftL1, periodLength,1);
L2 = ShiftMatrix(L2, shiftL2, periodLength,1);
L3 = ShiftMatrix(L3, shiftL3, periodLength,1);
L4 = ShiftMatrix(L4, shiftL4, periodLength,1);
L5 = ShiftMatrix(L5, shiftL5, periodLength,1);
L6 = ShiftMatrix(L6, shiftL6, periodLength,1);
L7 = ShiftMatrix(L7, shiftL7, periodLength,1);
L8 = ShiftMatrix(L8, shiftL8, periodLength,1);
L9 = ShiftMatrix(L9, shiftL9, periodLength,1);
L10 = ShiftMatrix(L10, shiftL10, periodLength,1);
L11 = ShiftMatrix(L11, shiftL11, periodLength,1);
L12 = ShiftMatrix(L12, shiftL12, periodLength,1);
L13 = ShiftMatrix(L13, shiftL13, periodLength,1);

deltaL1(1,1) = 0;
deltaL2(1,1) = 0;
deltaL3(1,1) = 0;
deltaL4(1,1) = 0;
deltaL5(1,1) = 0;
deltaL6(1,1) = 0;
deltaL7(1,1) = 0;
deltaL8(1,1) = 0;
deltaL9(1,1) = 0;
deltaL10(1,1) = 0;
deltaL11(1,1) = 0;
deltaL12(1,1) = 0;
deltaL13(1,1) = 0;
deltaT(1,1) = 0;

for(i=2:1:periodLength)
    deltaT (i,1) = (TargetIndicator(i,1)-TargetIndicator(i-1,1))/TargetIndicator(i-1,1);
    deltaL1 (i,1) = (L1(i,1)-L1(i-1,1))/L1(i-1,1);
    if(i<1001)
        deltaL2 (i,1) = (L2(i,1)-L2(i-1,1)); %news cant devide by 0
    else
        deltaL2 (i,1)= 0; % since news in coincident we have to assume = 0
    end
    deltaL3 (i,1) = (L3(i,1)-L3(i-1,1))/L3(i-1,1);
    deltaL4 (i,1) = (L4(i,1)-L4(i-1,1))/L4(i-1,1);
    deltaL5 (i,1) = (L5(i,1)-L5(i-1,1))/L5(i-1,1);
    deltaL6 (i,1) = (L6(i,1)-L6(i-1,1))/L6(i-1,1);
    deltaL7 (i,1) = (L7(i,1)-L7(i-1,1))/L7(i-1,1);
    deltaL8 (i,1) = (L8(i,1)-L8(i-1,1))/L8(i-1,1);
    deltaL9 (i,1) = (L9(i,1)-L9(i-1,1))/L9(i-1,1);
    deltaL10 (i,1) = (L10(i,1)-L10(i-1,1))/L10(i-1,1);
    deltaL11 (i,1) = (L11(i,1)-L11(i-1,1))/L11(i-1,1);
    deltaL12 (i,1) = (L12(i,1)-L12(i-1,1))/L12(i-1,1);
    deltaL13 (i,1) = (L12(i,1)-L12(i-1,1))/L13(i-1,1);

    tn (i,1) = (TargetIndicator(i,1)-TargetIndicator(i-1,1))/TargetIndicator(i-1,1);
    L1n (i,1) = (L1(i,1)-L1(i-1,1))/L1(i-1,1);
    L2n (i,1) = (L2(i,1)-L2(i-1,1)); %news cant devide by 0
    L3n (i,1) = (L3(i,1)-L3(i-1,1))/L3(i-1,1);
    L4n (i,1) = (L4(i,1)-L4(i-1,1))/L4(i-1,1);
    L5n (i,1) = (L5(i,1)-L5(i-1,1))/L5(i-1,1);
    L6n (i,1) = (L6(i,1)-L6(i-1,1))/L6(i-1,1);
    L7n (i,1) = (L7(i,1)-L7(i-1,1))/L7(i-1,1);
    L8n (i,1) = (L8(i,1)-L8(i-1,1))/L8(i-1,1);
    L9n (i,1) = (L9(i,1)-L9(i-1,1))/L9(i-1,1);
    L10n (i,1) = (L10(i,1)-L10(i-1,1))/L10(i-1,1);
    L11n (i,1) = (L11(i,1)-L11(i-1,1))/L11(i-1,1);
    L12n (i,1) = (L12(i,1)-L12(i-1,1))/L12(i-1,1);
    L13n (i,1) = (L12(i,1)-L12(i-1,1))/L13(i-1,1);

end

tn(1:1000, 1) = forgettingFactor(tn(1:1000, 1),.9999999);
L1n(1:1000, 1) = forgettingFactor(L1n(1:1000, 1),.9999999);
L2n(1:1000, 1) = forgettingFactor(L2n(1:1000, 1),.9999999);
L3n(1:1000, 1) = forgettingFactor(L3n(1:1000, 1),.9999999);
L4n(1:1000, 1) = forgettingFactor(L4n(1:1000, 1),.9999999);
L5n(1:1000, 1) = forgettingFactor(L5n(1:1000, 1),.9999999);
L6n(1:1000, 1) = forgettingFactor(L6n(1:1000, 1),.9999999);
L7n(1:1000, 1) = forgettingFactor(L7n(1:1000, 1),.9999999);
L8n(1:1000, 1) = forgettingFactor(L8n(1:1000, 1),.9999999);
L9n(1:1000, 1) = forgettingFactor(L9n(1:1000, 1),.9999999);
L10n(1:1000, 1) = forgettingFactor(L10n(1:1000, 1),.9999999);
L11n(1:1000, 1) = forgettingFactor(L11n(1:1000, 1),.9999999);
L12n(1:1000, 1) = forgettingFactor(L12n(1:1000, 1),.9999999);
L13n(1:1000, 1) = forgettingFactor(L13n(1:1000, 1),.9999999);

initialTrainingPeriod = 1000;
itp = initialTrainingPeriod;
success = 0;

for(i=1:1:1001)
    trainedNet = train_Final_BetterDesign(L1n(1:1000+i,1), L2n(1:1000+i,1), L3n(1:1000+i,1),
      L4n(1:1000+i,1), L5n(1:1000+i,1), L6n(1:1000+i,1), L7n(1:1000+i,1), L8n(1:1000+i,1),
      L9n(1:1000+i,1), L10n(1:1000+i,1), L11n(1:1000+i,1), L12n(1:1000+i,1), L13n(1:1000+i,1),
      tn(1:1000+i,1));
    forecasted(i,1)=0;
end
for (i=initialTrainingPeriod:1:2000)
    I = [L1n(i,1), L2n(i,1), L3n(i,1), L4n(i,1), L5n(i,1), L6n(i,1), L7n(i,1), L8n(i,1),
       L9n(i,1), L10n(i,1), L11n(i,1), L12n(i,1), L13n(i,1)]';
    forecastedn(i+1,1) = sim(trainedNet,I);
    forecasted(i+1,1) = forecastedn(i,1);
end

success = 0;
for (i=1001:1:2000)
    if (forecasted(i,1) * deltaT(i,1) > 0)
        success = success + 1;
    end
    error(i,1) = forecasted(i,1) - deltaT(i,1);
end

successRate = success/1000
rmse = RMSE(error)
nrmse = NRMSE(error)
mae = MAE(error)
mape = MAPE(error)
```

```
datatemp = (data(1:2000,1));
Yt_temp(1,1) = datatemp(1,1);
for (i=2:1:2000)
    Yt_temp(i,1) =  Yt_temp(i-1,1)+forecasted(i,1);
end

figure
subplot(3,2,1)
plot(x_1, forecasted(1:2000,1), '.');
title('PM Return')
ylabel('Return')
subplot(3,2,2)
plot(x_1, deltaT(1:2000,1), '.');
title('Real Return')
ylabel('Return')
subplot(3,2,3:4)
plot(x_1, error, '.');
title('Error')
ylabel('Return')
subplot(3,2,5:6)
plot(x_1, data(1:2000),'.', x_1, Yt_temp, '.');
title('Real Value Vs. Forecasted Value')
ylabel('Value')

M=[forecasted(1001:2000,1) deltaT(1001:2000,1) error(1001:2000,1)];
end
```

8.11 Matlab Codes for Statistical Tests

```
% Function that aims at showing how it's prediction reaction as it is
% increasingly sure of its bet
% Use it: CertaintyIndex(MatrixReturnOfNeuralNet(:,1),MatrixReturnOfNeuralNet(:,2),2)

function M = CertaintyIndex(InputLeadingIndicator, TargetIndicator, k);
dataLength = length(InputLeadingIndicator);
count1 = 0;
count2 = 0;
for (i=1:1:dataLength)
    if (abs(InputLeadingIndicator(i,1)) >= k*mean(abs(InputLeadingIndicator)))
        count1 = count1+1;
        if (((InputLeadingIndicator(i,1) >= 0) & (TargetIndicator(i,1)>=0)) |
         ((InputLeadingIndicator(i,1) <= 0) & (TargetIndicator(i,1)<=0)))
            count2 = count2+1;
        end
    end
end
success = count2/count1;
counts = [count1 count2]
M = success;
end
```

```
% function that calculates the success rate

function M = SR(matrix2columnsOfReturns);

target = matrix2columnsOfReturns(:,1);
input = matrix2columnsOfReturns(:,2);
Length = length(input)
success = 0;
for (i=1:1:Length)
    if (target(i,1) * input(i,1) > 0)
        success = success + 1;
    end
end

M = success/Length;
```

```
% This function aims at looking at how successful was our design through
% time. This is a method used in our behavioral finance analysis.
function M = historical_Success(input, target);

success_1 = 0;
success_2 = 0;
success_3 = 0;
success_4 = 0;
success_5 = 0;
success_6 = 0;
```

```
success_7 = 0;
success_8 = 0;
success_9 = 0;
success_10 = 0;
success_11 = 0;
success_12 = 0;
success_13 = 0;
success_14 = 0;
success_15 = 0;
success_16 = 0;
success_17 = 0;
success_18 = 0;
success_19 = 0;
success_20 = 0;

interval_1 = 0;
interval_2 = 0;
interval_3 = 0;
interval_4 = 0;
interval_5 = 0;
interval_6 = 0;
interval_7 = 0;
interval_8 = 0;
interval_9 = 0;
interval_10 = 0;
interval_11 = 0;
interval_12 = 0;
interval_13 = 0;
interval_14 = 0;
interval_15 = 0;
interval_16 = 0;
interval_17 = 0;
interval_18 = 0;
interval_19 = 0;
interval_20 = 0;

TotalSuccess = 0;
totL = length(input);
j = 0;
k = 0;
for (i=1:1:totL)
    if (input(i,1)*target(i,1)>0)
        TotalSuccess = TotalSuccess +1;
        j=j+1;
    end
    %------
    if (input(i,1)*target(i,1)>0)&(i<totL/20)
        success_1 = success_1 + 1;
        k=k+1;
    end
    if (i<totL/20)
        interval_1 = interval_1 +1;
    end
    %------
    if (input(i,1)*target(i,1)>0)&(i>=totL/20)&(i<2*totL/20)
        success_2 = success_2 + 1;
        k=k+1;
```

```
end
if (i>=totL/20)&(i<2*totL/20)
    interval_2 = interval_2 +1;
end
%------
if (input(i,1)*target(i,1)>0)&(i>=2*totL/20)&(i<3*totL/20)
    success_3 = success_3 + 1;
    k=k+1;
end
if (i>=2*totL/20)&(i<3*totL/20)
    interval_3 = interval_3 +1;
end
%------
if (input(i,1)*target(i,1)>0)&(i>=3*totL/20)&(i<4*totL/20)
    success_4 = success_4 + 1;
    k=k+1;
end
if (i>=3*totL/20)&(i<4*totL/20)
    interval_4 = interval_4 +1;
end
%------
if (input(i,1)*target(i,1)>0)&(i>=4*totL/20)&(i<5*totL/20)
    success_5 = success_5 + 1;
    k=k+1;
end
if (i>=4*totL/20)&(i<5*totL/20)
    interval_5 = interval_5 +1;
end
%------
if (input(i,1)*target(i,1)>0)&(i>=5*totL/20)&(i<6*totL/20)
    success_6 = success_6 + 1;
    k=k+1;
end
if (i>=5*totL/20)&(i<6*totL/20)
    interval_6 = interval_6 +1;
end
%------
if (input(i,1)*target(i,1)>0)&(i>=6*totL/20)&(i<7*totL/20)
    success_7 = success_7 + 1;
    k=k+1;
end
if (i>=6*totL/20)&(i<7*totL/20)
    interval_7 = interval_7 +1;
end
%------
if (input(i,1)*target(i,1)>0)&(i>=7*totL/20)&(i<8*totL/20)
    success_8 = success_8 + 1;
    k=k+1;
end
if (i>=7*totL/20)&(i<8*totL/20)
    interval_8 = interval_8 +1;
end
%------
if (input(i,1)*target(i,1)>0)&(i>=8*totL/20)&(i<9*totL/20)
    success_9 = success_9 + 1;
    k=k+1;
end
```

```
if (i>=8*totL/20)&(i<9*totL/20)
    interval_9 = interval_9 +1;
end
%------
if (input(i,1)*target(i,1)>0)&(i>=9*totL/20)&(i<10*totL/20)
    success_10 = success_10 + 1;
    k=k+1;
end
if (i>=9*totL/20)&(i<10*totL/20)
    interval_10 = interval_10 +1;
end
%------
if (input(i,1)*target(i,1)>0)&(i>=10*totL/20)&(i<11*totL/20)
    success_11 = success_11 + 1;
    k=k+1;
end
if (i>=10*totL/20)&(i<11*totL/20)
    interval_11 = interval_11 +1;
end
%------
if (input(i,1)*target(i,1)>0)&(i>=11*totL/20)&(i<12*totL/20)
    success_12 = success_12 + 1;
    k=k+1;
end
if (i>=11*totL/20)&(i<12*totL/20)
    interval_12 = interval_12 +1;
end
%------
if (input(i,1)*target(i,1)>0)&(i>=12*totL/20)&(i<13*totL/20)
    success_13 = success_13 + 1;
    k=k+1;
end
if (i>=12*totL/20)&(i<13*totL/20)
    interval_13 = interval_13 +1;
end
%------
if (input(i,1)*target(i,1)>0)&(i>=13*totL/20)&(i<14*totL/20)
    success_14 = success_14 + 1;
    k=k+1;
end
if (i>=13*totL/20)&(i<14*totL/20)
    interval_14 = interval_14 +1;
end
%------
if (input(i,1)*target(i,1)>0)&(i>=14*totL/20)&(i<15*totL/20)
    success_15 = success_15 + 1;
    k=k+1;
end
if (i>=14*totL/20)&(i<15*totL/20)
    interval_15 = interval_15 +1;
end
%------
if (input(i,1)*target(i,1)>0)&(i>=15*totL/20)&(i<16*totL/20)
    success_16 = success_16 + 1;
    k=k+1;
end
if (i>=15*totL/20)&(i<16*totL/20)
```

```
        interval_16 = interval_16 +1;
    end
    %------
    if (input(i,1)*target(i,1)>0)&(i>=16*totL/20)&(i<17*totL/20)
        success_17 = success_17 + 1;
        k=k+1;
    end
    if (i>=16*totL/20)&(i<17*totL/20)
        interval_17 = interval_17 +1;
    end
    %------
    if (input(i,1)*target(i,1)>0)&(i>=17*totL/20)&(i<18*totL/20)
        success_18 = success_18 + 1;
        k=k+1;
    end
    if (i>=17*totL/20)&(i<18*totL/20)
        interval_18 = interval_18 +1;
    end
    %------
    if (input(i,1)*target(i,1)>0)&(i>=18*totL/20)&(i<19*totL/20)
        success_19 = success_19 + 1;
        k=k+1;
    end
    if (i>=18*totL/20)&(i<19*totL/20)
        interval_19 = interval_19 +1;
    end
    %------
    if (input(i,1)*target(i,1)>0)&(i>=19*totL/20)
        success_20 = success_20 + 1;
        k=k+1;
    end
    if (i>=19*totL/20)
        interval_20 = interval_20 +1;
    end
    %------
end

interval_1
interval_2
interval_3
interval_4
interval_5
interval_6
interval_7
interval_8
interval_9
interval_10
interval_11
interval_12
interval_13
interval_14
interval_15
interval_16
interval_17
interval_18
interval_19
interval_20
```

```
sR1 = success_1 / interval_1;
sR2 = success_2 / interval_2;
sR3 = success_3 / interval_3;
sR4 = success_4 / interval_4;
sR5 = success_5 / interval_5;
sR6 = success_6 / interval_6;
sR7 = success_7 / interval_7;
sR8 = success_8 / interval_8;
sR9 = success_9 / interval_9;
sR10 = success_10 / interval_10;
sR11 = success_11 / interval_11;
sR12 = success_12 / interval_12;
sR13 = success_13 / interval_13;
sR14 = success_14 / interval_14;
sR15 = success_15 / interval_15;
sR16 = success_16 / interval_16;
sR17 = success_17 / interval_17;
sR18 = success_18 / interval_18;
sR19 = success_19 / interval_19;
sR20 = success_20 / interval_20;

sRt = TotalSuccess / totL;
M = [sR1 sR2 sR3 sR4 sR5 sR6 sR7 sR8 sR9 sR10 sR11 sR12 sR13 sR14 sR15 sR16 sR17 sR18 sR19 sR20 sRt totL/1000];
end
```

```
% Function that aims at calculating the historical success rate index.

function M = HSRI(array);
success = 0;
for (i=1:1:length(array)-1)
    if ((array(1,i)>=array(1,i+1))&((array(1,i)>0.5)))
        success = success + array(1,i) - array(1,i+1);
    end
    if ((array(1,i)>=array(1,i+1))&((array(1,i)<0.5)))
        success = success - array(1,i) + array(1,i+1);
    end
    if ((array(1,i)<=array(1,i+1))&((array(1,i)>0.5)))
        success = success - array(1,i) + array(1,i+1);
    end
    if ((array(1,i)<=array(1,i+1))&((array(1,i)<0.5)))
        success = success + array(1,i) - array(1,i+1);
    end
end
M = success/length(array);
end
```

```
% function that tests the root mean square error

function M = RMSE(error)
for (i=1:1:length(error))
    errSq(i,1) = error(i,1)*error(i,1);
```

```
end
M = (mean(errSq))^(1/2);
end
```

```
% function that tests the normalized root mean square error

function M = NRMSE(error)
M = RMSE(error)/std(error);
end
```

```
% function that tests the mean absolute error

function M = MAE(error)
for (i=1:1:length(error))
    ae(i,1) = abs(error(i,1));
end
M = mean(ae);
end
```

```
% function that tests the mean absolute percentage error

function M = MAPE(error)
for (i=1:1:length(error))
    ape(i,1) = abs(error(i,1))/100;
end
SUMape = 0;
for (i=1:1:length(error))
    SUMape = SUMape + ape(i,1);
end
M = SUMape/length(error);
end
```

```
% Function that calculates a 95% confidence interval for a given CI


function M = ConfidenceIntervalCI30(InputLeadingIndicator, TargetIndicator);

counTemp = length(InputLeadingIndicator);
tempResult = CertaintyIndex(InputLeadingIndicator, TargetIndicator, 0);
countTemp = tempResult(1,2);
k=1;
while (countTemp >30)
    if (countTemp >30)
        result = CertaintyIndex(InputLeadingIndicator, TargetIndicator, k)
        count1 = result(1,2);
    end
    k=k+1;
    tempResult = CertaintyIndex(InputLeadingIndicator, TargetIndicator, k);
    countTemp = tempResult(1,2);
end

k=k-1; %correct the k from previous step
j=0;
for (i=1:1:length(InputLeadingIndicator))
    if (abs(InputLeadingIndicator(i,1)) >= k*mean(abs(InputLeadingIndicator)))
        j=j+1;
        MatrixResult(j,1) = InputLeadingIndicator(i,1) ;
        MatrixResult(j,2) = TargetIndicator(i,1);
    end
end

for(i=1:1:length(MatrixResult))
    if (InputLeadingIndicator(i,1) * TargetIndicator(i,1) >0)
        sucess(i,1)=1;
    else
        sucess(i,1)=0;
    end
end

meanSuccess=result(1,1);
standarDev = std(sucess(:,1));

lowerInterval =meanSuccess-1.96*standarDev/(count1)^(.5);
upperInterval =meanSuccess+1.96*standarDev/(count1)^(.5);

M = [lowerInterval upperInterval];
end
```

```
% Function that aims at calculating a summary for the CI index

function M = CertaintyIndexSum(array);
success = 0;
for (i=1:1:length(array)-1)
    if (array(1,i)>=array(1,i+1))
        success = success + 1;
    end
end
M = success/length(array);
end
```

8.12 Random Matlab Codes

```
% function that aims at cutting an array from above

function modMatrix=CutAbove(originalMatrix, rows, columns);
for row=1:rows
    for column=1:columns
        temp(row,column)=originalMatrix(row, column);
    end
end
modMatrix=temp;
end
```

```
% function that aims at cutting an array from below

function modMatrix=CutBelow(originalMatrix, rows, columns);
start=CutAbove(size(originalMatrix),1,1)-rows+1;
for row=start:CutAbove(size(originalMatrix),1,1)
    for column=1:columns
        temp(row-start+1,column)=originalMatrix(row, column);
    end
end
modMatrix=temp;
end
```

```
% function that aims at shifting the data

function modMatrix = ShiftMatrix(originalMatrix, shiftLength, rows,columns);
temp = CutAbove(originalMatrix,(shiftLength+rows),columns);
temp2 = CutBelow(temp,rows,columns);
modMatrix=temp2;
end
```

```
% this function aims at making past data less relevant. It uses the exponential decay formula
% How it works:
% paste the following in the terminal to make it work (S_P_500 and .99 are only arguments and could be replaced
% by respectivly a Matrix and a number between 0 and 1): forgettingFactor(S_P_500, .99)
% Note that the smaler is the number, the faster the data becomes irrelevent.

function modMatrix = forgettingFactor(originalMatrix, a);
temp = originalMatrix;
row = CutAbove(size(originalMatrix), 1, 1);
column = CutAbove(size(transpose(originalMatrix)), 1, 1);
for (i=1:1:row)
    for (j=1:1:column)
        temp(i,j) = temp(i,j)*a^i;
    end
end
plot(temp,'.');  % test
modMatrix=temp;
```

```
% this function aims at reducing the noise in a set of data.
% it uses the weighted moving average formula

% How it works:
% paste the following in the terminal to make it work
% (B and .8 are only arguments and could be replaced by respectivly a Matrix and a number between 0 and 1):
% reduceNoise(B, .8)
% Note that the smaller is the number, the more noise is taken away.

function modMatrix = reduceNoise(originalMatrix, a);

temp = originalMatrix;
row = CutAbove(size(originalMatrix), 1, 1);
column = CutAbove(size(transpose(originalMatrix)), 1, 1);

for (i=1:1:row)
    for (j=1:1:column)
        if (i==1)
            temp(i,j) = originalMatrix(i, j);
        else
            temp(i,j) = (1-a)*temp(i-1,j) + a*originalMatrix(i, j);
        end
    end
end
%  plot(temp,'.');  % test
modMatrix=temp;
end
```

```
% Function that aims at finding the shift the makes the correlation between
% 2 indicators maximum.
% shiftForMaxCorr(AMEX_Oil_Index, S_P_500)

function M = shiftForMaxCorr(L1, L2, studyPeriod);

L1 = flipud(L1);
L2 = flipud(L2);
R = 0;
bestShift = 1;
for (i=1:1:15)
    matrixAnswer = abs(corrcoef(L1(1:studyPeriod,1),L2(i:studyPeriod+i-1,1)));
    Rtemp = matrixAnswer(2,1);
    shift = i;
    if (Rtemp > R)
        R = Rtemp;
        bestShift = shift;
    end
end
bestResult = [R  bestShift];
M = bestShift;
end
```

References

[1] Patrick McSharry Interview in Oxford, UK., 2006.

[2] Pinnacle Data Corp., Historical Data for few Leading Indicators: http://www.pinnacledata.com/cot.html.

[3] Matlab Neural Net Toolbox, The Mathworks Inc.

[4] Riedmiller, M., Braun, H., in: Ruspini, H. (Ed.), IEEE Int. Conf. Neural Networks, IEEE, San Francisco 1993, pp. 586 to 591.

[5] Bloomberg.

[6] B. Abraham, A. & Nath. A neuro-fuzzy approach for modelling electricity demand in victoria. *Applied Soft Computing Journal*, 1:127–138, 2001.

[7] Report by a Group of Independent Experts. External evaluation of imf surveillance. 1999.

[8] Chatfield C. Calculating interval forecasts. *Journal of Business and Economics Statistics*, 11:121–135, 1993.

[9] Winkler RL Clemen RT, Murphy AH. Screening probability forecasts: Contrasts between choosing and combining. *International Journal of Forecasting*, 11:133–146, 1995.

[10] M. Darbellay, G.A. & Slama. Forecasting the short-term demand for electricity - do neural networks stand a better chance? *International Journal of Forecasting*, 16:71–83, 2000.

[11] Charles Darwin. *The Origin of Species*. Gramercy, 1995.

[12] Jay. L. Devore. *Probability and Statistics for Engineering and the Sciences*. Duxbury, Thomson Learning, 2000.

[13] J. D. Schaffer R. A. Caruana L.J. Eshelman and R. Das. A study of control parameters affecting online performance of genetic algorithms for function optimization. pages 51–60. In Schaffer, 1989.

[14] Yahoo Finance, 2006. Historical data for the S&P500 and few Leading Indicators.

[15] Edward Gately. *Neural Networks for Financial Forecasting*. John Wiley & Sons, 1995.

[16] Investors' Intelligence Inc. New Rochelle NY.

[17] John G. Taylor Jimmy Shadbolt. *Neural Networks and the Financial Markets: Predicting, Combining and Portfolio Optimisation (Perspectives in Neural Computing).* Springer, 2006.

[18] W.D. & Smith D.G.C. Laing. A comparison of time series forecasting methods for predicting the cegb demand. Proceedings of the Ninth Power Systems Computation Conference, Cascais, Portugal., 1987.

[19] Burton G. Malkiel. *A Random Walk Down Wall Street.* W. W. Norton & Company, 2003.

[20] Winkler RL Murphy AH. Diagnostic verification of probability forecasts. *International Journal of Forecasting*, 7:435–455, 1992.

[21] Humphrey Neill. *The Art of Contrary Thinking.* Caxton Press; 5th Enlrg edition, 1997.

[22] Edgar E. Peters. *Chaos and Order in the Capital Markets: A New View of Cycles, Prices, and Market Volatility.* John Wiley & Sons, 1996.

[23] Christoffersen PF. Evaluating interval forecasts. *International Economic Review*, 39:841–862, 1997.

[24] Min Qi. Predicting us recessions with leading indicators via neural network models. *International Journal of Forecasting*, 17:383 401, 2001.

[25] TIAA-CREF Brokerage Services. Online Glossary.

[26] Paul P. Wang Shu-Heng Chen. *Computational Intelligence in Economics and Finance (Advanced Information Processing S.).* Springer-Verlag Berlin and Heidelberg GmbH & Co. K, 2003.

[27] Jeremy J. Siegel. *Stocks for the Long Run.* McGraw-Hill, 2002.

[28] Vijay Singal. *Beyond the Random Walk: A Guide to Stock Market Anomalies and Low-Risk Investing.* Oxford University Press, Inc., 2003.

[29] David Stamp. *Market Prophets: Can Forecasters Predict the Financial Future?* Pearson Education, 2002.

[30] Peter Norvig Stuart Russel. *Artificial Intelligence, A Modern Approach.* Prentice Hall, 2003.

[31] J.W Taylor. Short-term electricity demand forecasting using double seasonal exponential smoothing. *Journal of Operational Research Society*, 54:799–805, 2003.

[32] Eric Troseth. Can you forecast the market? (october 20th edition). *Work & Money*, 2003.

[33] Andreas S. Weigend and Shanming Shi. Predicting daily probability distributions of s&p500 returns. *Journal of Forecasting*, 19:375–392, 2000.

[34] Frank Williams, 1930.

[35] Iraj Zandi, 2004. interview on Neural Networks and length of data.

[36] G. Peter Zhang. *Neural Networks in Business Forecasting.* Information Science Publishing, 2003.