MA 374 - Financial Engineering Extra Assignment

Sahil Kumar Gupta

200123081

Title: Prediction of Stock Price using various Techniques

In this project, we explore the use of three different methods to predict stock prices: Long Short-Term Memory (LSTM) neural networks, Markov Chain Monte Carlo (MCMC) simulations, and exponential smoothing. Each method has its own strengths and limitations, and by combining them we aim to obtain a more robust and accurate prediction. We then take a weighted average of the individual predictions to generate a final forecast. We apply these methods to predict the stock prices of a company using data from Yahoo Finance, and evaluate the performance of our approach using metrics such as mean squared error (MSE) and root mean squared error (RMSE).

At the end, we try to find a weighted sum of all the three methods by solving an optimization problem. This gives us the final result, which is much better than the individual results.

Exponential Smoothing

In time series analysis, we are interested in modeling and forecasting data that is collected over time. The data can be collected at regular intervals (e.g., daily, weekly, monthly) and can be used to make predictions about future values of the series. There are many techniques available for time series forecasting, and one popular method is exponential smoothing.

Exponential smoothing is a popular method for time series forecasting that is based on the idea of smoothing the data by giving more weight to recent observations and less weight to older observations. The basic idea is to calculate a weighted average of the past observations, where the weights decay exponentially as the observations get older.

There are several variations of exponential smoothing, including simple exponential smoothing, double exponential smoothing, and triple exponential smoothing (also known as Holt-Winters forecasting). Here we have focused on simple exponential smoothing.

Simple exponential smoothing is a method for forecasting a time series that is based on a weighted average of the past observations. The method assumes that the future values of the series are a function of the past values, and that the errors in the past observations are normally distributed with a mean of zero.

The formula for simple exponential smoothing is as follows:

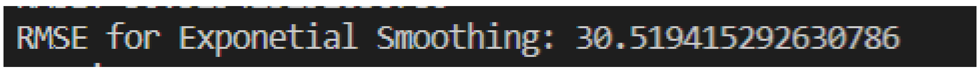
St = alpha*Yt + (1 – alpha) *St-1

where St is the smoothed value at time t, Yt is the actual value at time t, and α is the smoothing parameter, which is a value between 0 and 1 that determines the weight given to the current observation relative to the past observations. The larger the value of α, the more weight is given to the current observation.

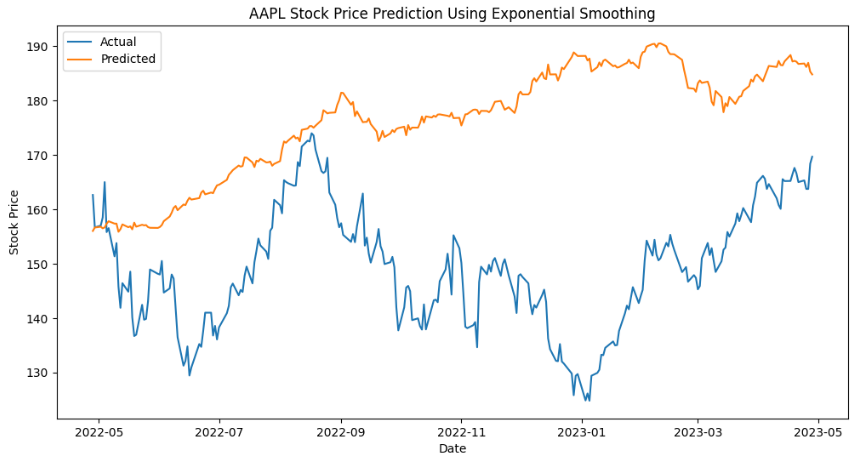
The initial value of the smoothed series, S1, is set equal to the first observation in the series, Yt. For subsequent time periods, the formula above is used to calculate the smoothed value.

The forecast for the next period, Yt+1, is simply equal to the smoothed value at time t, St. That is, Yt+1 = St.

By using the code provided in this project, we were able to apply simple exponential smoothing to real-world stock data and generate forecasts for future stock prices. The RMSE value:



```
RMSE for Exponetial Smoothing: 30.519415292630786
```



Long Short-Term Memory (LSTM) algorithm

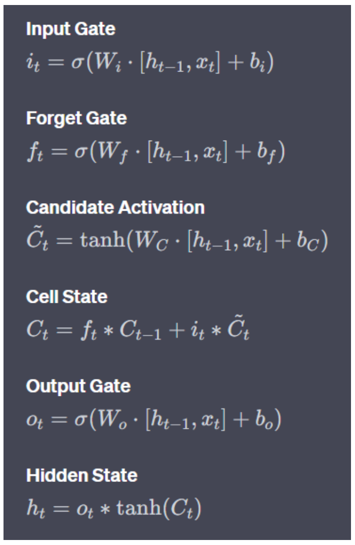
The LSTM algorithm is based on a memory cell that can store information over a long period of time. It uses three gates - the input gate, forget gate, and output gate - to control the flow of information into and out of the cell. The input gate decides which information to let into the cell, the forget gate decides which information to discard from the cell, and the output gate decides which information to use as the output of the cell.

Mathematically, the LSTM algorithm can be expressed as follows:

- **The input gate:** **i_t = sigmoid (W_i * [h_{t-1}, x_t] + b_i)**
- **The forget gate:** **f_t = sigmoid (W_f * [h_{t-1}, x_t] + b_f)**
- **The output gate:** **o_t = sigmoid(W_o * [h_{t-1}, x_t] + b_o)**

- **The memory cell:** **c_t = f_t * c_{t-1} + i_t * tanh(W_c * [h_{t-1}, x_t] + b_c)**
- **The hidden state:** **h_t = o_t * tanh(c_t)**

Here, x_t represents the input at time step t, h_{t-1} represents the hidden state at time step t-1, and i_t, f_t, o_t, c_t, and h_t represents the input gate, forget gate, output gate, memory cell, and hidden state at time step t, respectively. W_i, W_f, W_o, and W_c represent the weight matrices, and b_i, b_f, b_o, and b_c represent the bias vectors.



Input Gate

$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i)$

Forget Gate

$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f)$

Candidate Activation

$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] + b_C)$

Cell State

$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$

Output Gate

$o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o)$

Hidden State

$h_t = o_t * \tanh(C_t)$

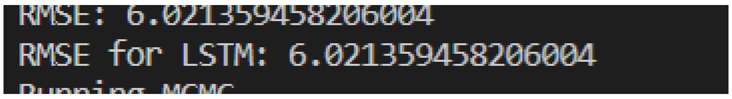
Same set of equations but with mathematical notations

To predict stock prices using LSTM, we first prepare the data by dividing it into training and testing sets, and normalizing it to have zero mean and unit variance. During training, the LSTM parameters are updated using backpropagation through time (BPTT). The loss function used for regression tasks is typically mean squared error (MSE).

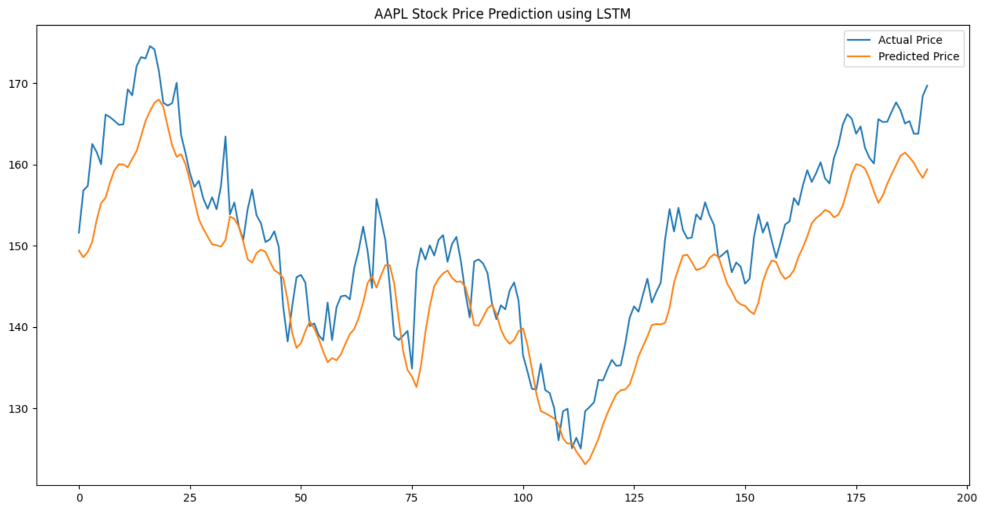
To use an LSTM for stock price prediction, we can feed in a sequence of historical prices as input and train the LSTM to predict the next price in the sequence. We can then generate a sequence of predicted prices by feeding in the predicted prices as input and iteratively predicting the next price.

To evaluate the performance of the LSTM model, we use the Root Mean Squared Error (RMSE) metric, which measures the difference between the predicted and actual stock prices. The lower the RMSE, the better the model's performance.

In this project, we implemented the LSTM model in Python using the Keras library, and trained it using five years of historical stock price data obtained from Yahoo Finance. We then used the trained model to predict the stock prices for the next year, and evaluated its performance using RMSE.



```
RMSE: 6.021359458206004
RMSE for LSTM: 6.021359458206004
Running MCMC
```



Markov Chain Monte Carlo (MCMC) method

The MCMC method is a statistical technique used to generate random samples from a probability distribution. In our case, we used MCMC to generate random samples of stock prices based on a probability distribution function.

The MCMC algorithm can be represented mathematically as follows:

1. Define the target distribution p(x).
2. Choose a proposal distribution q(x).
3. Initialize the Markov Chain with a starting value x_0.
4. For i = 1 to N, generate a new sample x_n from the proposal distribution q(x_n|x_i-1).
5. Calculate the acceptance probability a = min(1, p(x_n)q(x_i-1|x_n) / (p(x_i-1)q(x_n|x_i-1)))
6. Accept the new sample with probability a and reject it otherwise.
7. Return the resulting samples as a set of Markov Chain samples.

The MCMC algorithm can be used for a wide range of applications, including Bayesian inference, optimization, and simulation.

To begin with, we defined a normal distribution function `q(x)` with mean `mu` and standard deviation `sig`. We then used this distribution function to generate random samples using MCMC. We set the number of samples to be generated as `N`.

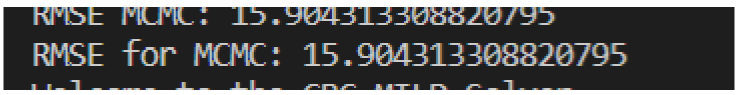
Next, we implemented the Metropolis-Hastings algorithm to generate the samples. The Metropolis-Hastings algorithm is a Markov Chain Monte Carlo method that generates a sequence of samples from a probability distribution. It does so by accepting or rejecting new proposed samples based on a ratio of the posterior probabilities of the new and the current samples.

We initialized our algorithm with a starting point `r` and calculated the probability of this point using the normal distribution function `q(x)`. We then generated a new proposed point `rn` by adding a random number to the current point `r`. We then calculated the probability of this new point `rn` using the same normal distribution function `q(x)`.

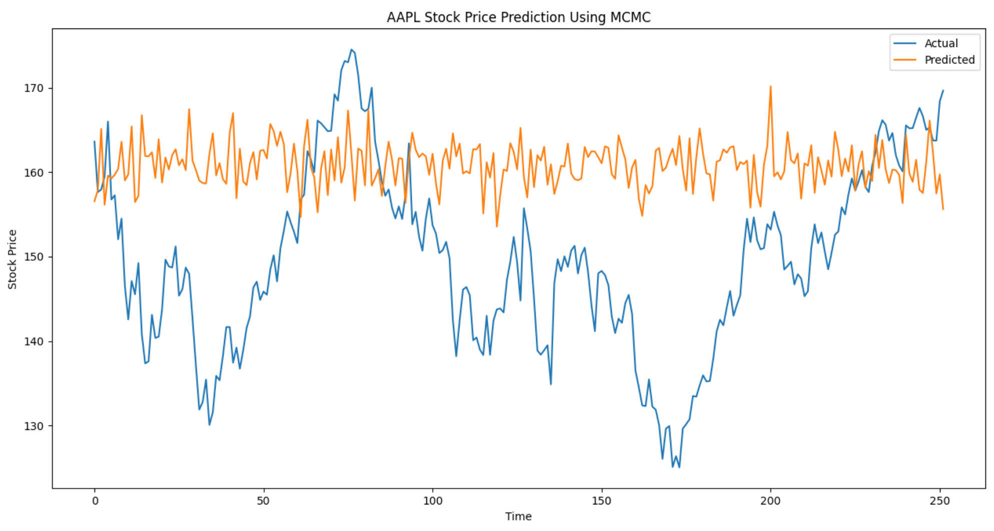
If the probability of the new proposed point `rn` is greater than or equal to the probability of the current point `r`, we accept the new point and set it as the current point. If the probability of the new point is less than the probability of the current point, we calculate a probability ratio `u` and accept the new point with probability `u`. We repeat this process for `N` iterations and obtain a sequence of samples.

To predict stock prices, we used the generated sequence of samples to model the future stock prices. We took the last known stock price as our starting point and used the MCMC method to generate a sequence of future stock prices. We then took the mean of this sequence as our predicted stock price.

Overall, the MCMC method is a powerful statistical technique for generating random samples from a probability distribution. In our case, we used it to predict future stock prices based on a normal distribution function. However, it is important to note that MCMC can be computationally intensive and may not be suitable for large datasets. It is important to evaluate the performance of the model using appropriate metrics and to compare it with other predictive techniques.



```
RMSE MCMC: 15.904313308820795
RMSE for MCMC: 15.904313308820795
```



Weighted Average

All the three methods have different performance level which is visible from the RMSE values. Now to make our predictions better, we used optimization techniques on RMSE values to get optimal weights and generate a weighted prediction for the test data and hence predicted the stock prices. The entire process of this can be described as:

We started by defining three RMSE values for the three prediction techniques as decision variables in Pulp. We also defined the weights as decision variables, which can range between 0 and 0.7 and must sum to 1. Our objective was to minimize the weighted RMSE of the three techniques. We modeled the objective function as a series of constraints and solved the problem using the default solver.

Once we obtained the optimal weights using Pulp, we used them to generate a weighted prediction for a set of test data. We also calculated the RMSE of this weighted prediction, which gives us an indication of how well the three techniques are performing when their predictions are combined in this way.

The mathematical expression used to calculate the weighted RMSE can be written as follows:

weighted_RMSE = sqrt((w1*rmse_1)^2 + (w2*rmse_2)^2 + (w3*rmse_3)^2)

Here, w1, w2, and w3 are the weights assigned to each prediction technique, and rmse_1, rmse_2, and rmse_3 are the RMSE values for each technique. The objective is to minimize the weighted RMSE.

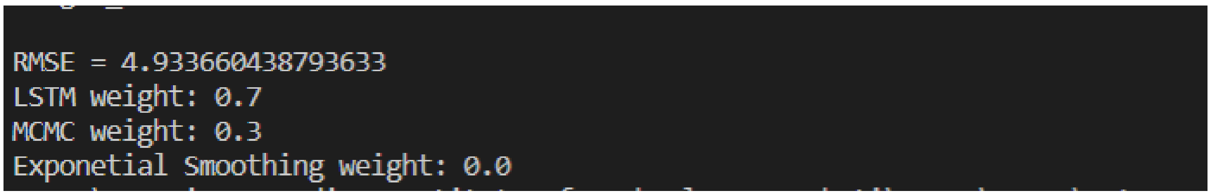
We can also calculate the weighted prediction using the optimal weights as follows:

y_weighted_pred = (w1_value * y_pred_1) + (w2_value * y_pred_2) + (w3_value * y_pred_3)

Here, y_pred_1, y_pred_2, and y_pred_3 are the predictions generated by each technique, and w1_value, w2_value, and w3_value are the optimal weights obtained using Pulp. The RMSE of this weighted prediction can be calculated using the standard formula:

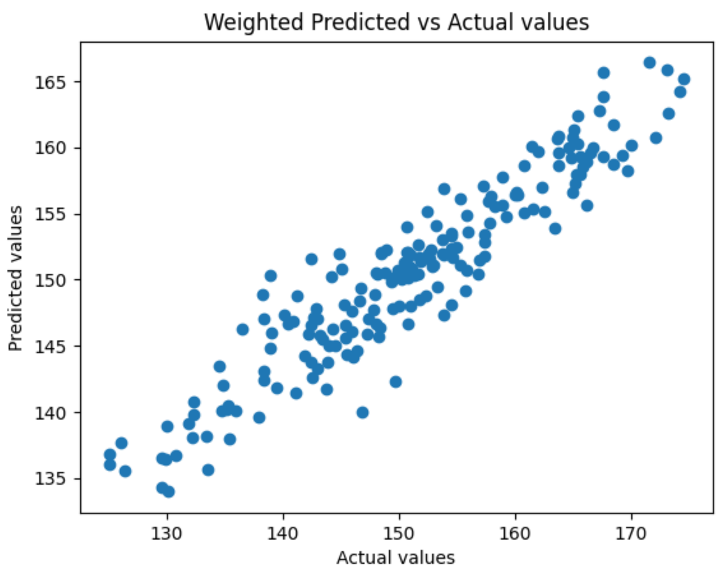
RMSE_weighted_pred = sqrt(mean((y_actual - y_weighted_pred)^2))

The final results are as:

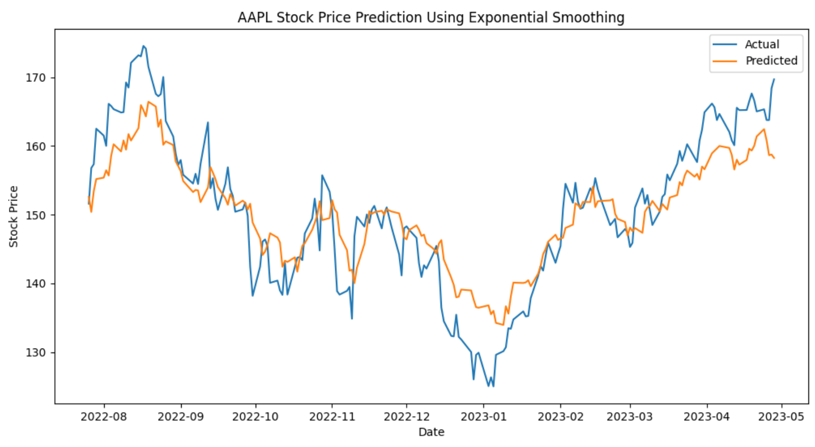


```
RMSE = 4.933660438793633
LSTM weight: 0.7
MCMC weight: 0.3
Exponetial Smoothing weight: 0.0
```

We can see a reduced RMSE value than all three methods.



We can see almost a linear relationship between the predicted values and the actual values with a slope m =1, which shows how good our prediction is.


AAPL Stock Price Prediction Using Exponential Smoothing
Actual
Predicted
Stock Price
Date