EECS 4404/5327 Project Report (Part 5)

Group AA

Abstract

The application provides prediction of Close price of S&P 500 ETF funds. It uses past market data of over 30 years and performs regression tasks We used three models: RNN, LSTM, & linear regression to develop a predictor based on past market data. The linear regression has a MSE (Mean Squared Error) of 0.62. The highest accuracy observed for the RNN and LSTM approaching values as high as 0.99. In addition we also perform more comparative analysis of the models and error analysis of the LTSM model. These metrics do not account for the impact of sentiment analysis as we were unable to find historic data (analyst recommendations) that overlapped with the market data. The implication is that we may be able to provide an idea of market trends and price movements to its users, but at a lower cost or to people for whom such resources are not accessible. This application does not replace the expert analysts and/or fund managers, but provides a satisfactory starting point to stock traders interested in trading ETFs.

Introduction

1. What is your application?

Our application is a stock price predictor for the SPY (S&P 500 ETF), designed to assist investors in making informed decisions. It integrates historical stock market data and performs regression tasks to predict the Close value stocks.

2. What are the assumptions/scope of your project?

Our project scope is limited to the SPY index. It does not extend to other securities or global markets.

The project assumes a certain level of predictability in stock prices based on historical data. The effectiveness of our model is contingent on the stability and standard behavior of the stock market. Extreme events or highly volatile markets might not be accurately predicted.

We assume that features like Open, Volume, High, Low are relevant features for predicting the Close price of a stock.

The application can be extended to incorporate sentiment analysis as a feature that affects the closing price of a stock, but the scope had to be limited during implementation and the model no longer includes sentiment analysis. The historic market data spans 30 years, starting in 1993. We were unable to gather the relevant news articles spanning this time period. If we used current articles, there would be no correlation between historic closing prices and analyst sentiments. Similarly, if we used a smaller, recent, dataset with a smaller time span, it would not be sufficient data to train a meaningful model.

3. Justify why is your application important?

A stock price predictor can be a useful tool to assist retail (non-expert) investors in making informed decisions without having to pay for or rely on guidance from financial experts. They can also use the predictions generated by this model to assess their knowledge of market trends. Currently, large banks (capital markets division) and hedge funds are the only institutions with access to this technology. This is only available to large commercial clients (large corporations).

By providing insights based on market trends, our tool could potentially help investors mitigate risks associated with stock market investments. This is particularly valuable in volatile market conditions.

This project is an important learning experience for us, giving us the chance to delve into and experiment with machine learning techniques in a hands-on setting.

4. Similar applications

There are many papers that conducted sentiment analysis but they use data from twitter. Also they do not combine it with past market data. Similarly, Kalyani et al. use news data for sentiment analysis but they employed SVM and Naive Bayes. We hoped to use a combination of news data and past market data but use LSTM and RNN models to explore this challenge.

E-Mini S&P 500 Index - Forecasts the moving average for the S&P 500

Alpaca - Offers chart analysis and forecasting using ML

Each of these similar applications focus on one model and evaluates its performance. In our work, we compare advanced models like SimpleRNN and LSTM with

comparatively simple baseline model like Linear regression to show by how much their performances differ. In addition to that, the similar application focuses on a dataset containing all types of stocks and our models work with a fund tracking s&p 500. Such funds have lower risks and by analysing the trends on the fund, we make the models suitable to use for new traders/ beginners trying to start trading with low risk.

5. Adjustments to part 1

Initially, we wanted to have signalling as a part of our project that would tell the user whether to buy or sell a stock. The dataset did not contain these labels - so it would be challenging to supervised learning. We could attach these labels by using a trading strategy like attach a buy label to a stock whose Close > Open in the previous day, but there are several trading strategies like that and using one particular strategy would limit the usefulness of our application to traders and would increase ambiguity

The architecture we had initially envisioned would also use reinforcement learning to improve the predictions of the NN trained on historic market data. Due to time constraints and issues with data collection, we decided to reduce our scope from part 1 to not include reinforcement learning to improve the model's accuracy.

We are also limiting the range of articles to perform sentiment analysis on market news on Yahoo Finance. For a time, we attempted to collect articles and label them ourselves for sentiment analysis. We realized that we were adding biases based on our assumptions. This, along with being unable to find articles spanning far enough back in time, we had to forgo the inclusion of sentiment analysis in the project.

Methodology

Design/pipeline

Our project follows a pipeline include 5 main stages:

Dataset Acquisition → Preprocessing Data → Model Implementation (Linear Regression, SimpleRNN, LSTM) → Prediction → Evaluation and Results

This approach leads to well-performing predictive modeling.

Dataset:

The dataset we used in the project is freely available in Kaggle [4]. It contains the last 30 years of S&P 500 ETFs daily data.

Preprocessing:

The dataset was thoroughly preprocessed before feeding it into out models:

1. Normalisation: We used a MinMaxScaler to normalise the key features like Open, High, Low, Close, Volume. By doing so, we scale all features between [0,1]
2. Feature Selection: We selected the relevant features that might affect the closing price of a stock. The selected features are: Open, High, Low, Close, Volume
3. Train-Test-Split: The train test ratio for our dataset is 80/20 - so that we can evaluate our model on unseen data.

Model implementation and training: Linear Regression

Technique: At the beginning of our analysis, we intuitively opted for training a Linear Regression model because it acts as a baseline model for our more complex predictive models doing the regression task.

Input: The input to the linear regression model is the normalised set of features Open, High, Low, Volume. We normalised it using MinMaxScaler.

Output: The output of the linear regression model predicts the Close price of the stock. Close price is the continuous dependent variable in the dataset.

Changes and decisions during model training: There weren't any architectural decisions or iterative approach/ change was necessary for this model. We evaluate the performance of this model using MSE and R-squared metrics.

Model Implementation and training: SimpleRNN and LSTM

Technique: We trained two types of neural network models for our project: A SimpleRNN and a LSTM (Long Short - Term Memory). The reason we chose these models is because of the nature of our dataset. Our dataset is a time series with daily stock data. These models are known for their ability to maintain a memory of previous input and tend to work well from sequential data.

Input: The normalised features of Open, High, Low, Volume input for both the models are inputs to both the models. As mentioned earlier, a MinMaxScaler ensures that all features proportionally contribute to the prediction. The data was reshaped into a 3D array before feeding them into the RNN-models.

Output: The output layers for both of these models have a single node. This is because the models predict a continuous float value - the Close price of a stock. For regression tasks like this, single node output seemed like the correct choice.

Changes and decisions during model training:

Throughout the training process, we made some changes to the NN structure:

- Initial number of nodes: We stated the first layer of both the models with 50 nodes. 50 nodes intuitively seemed to have good balance between capturing the trends in the dataset while not being too computationally expensive.
- Hidden Layers Functions: In both the SimpleRNN and LSTM models, hidden layers act as the model's memory, keeping track of temporal dependencies. Even Though we attempted to do hypertuning at the beginning with 30, 64 and 128 units. Later, we decided to use 50 layers in the hidden layers to prevent overfitting. We tried [50, 100, 150] for epoch values and then proceeded with 100 because it captured trends with realistic computation time. We use mean squared error as the loss function for the epochs. We used a batch size of 32 for the model.
- Model compilation: We used an adam optimizer to compile the model and minimize loss function. We did so because time series stock data can be noisy and Adam optimizer is a good choice for handling sparse gradient and noisy data [10]. We used Mean Squared Error because we are handling a regression problem that predicts continuous price value. MSE also helps us evaluate the performance of these models against the Linear Regression Model

Prediction

Linear Regression

When there is a new data input, the linear regression model predicts future prices by applying the linear relationship it learned between the features and the target Close price value. For a new data point, the model multiplies each feature by the weight that it learned and sums up these values to generate a prediction. The predicted output is a single continuous value depicting the stock price.

Prediction: SimpleRNN

When there is input data, the SimpleRNN opts for a sequential processing. The model takes the Close prices in order from the historic price data. It learns a pattern and then uses this learned pattern to predict the next Close price. Overall, the model can output a sequence of predicted Close price - one for each timestamp in the temporal series. This temporal sequential processing is practical and relevant for real-world financial data.

Prediction: LSTM

The LSTM model has a similar predicting mechanism like SimpleRNN but it handles long term dependency among the data better. When we input a sequence of past Close prices of the stock, the model uses its internal memory to infer the long term trends and patterns and predicts unseen Close price. The model can output a sequence of predicted Close prices but it has the ability to predict with higher accuracy. This kind of prediction is relevant in the financial area where long-term market and price trends can significantly affect future Close price of stocks.

Evaluation

Evaluation metrics: The main metrics for evaluating the performance of the SimpleRNN and and LSTM models is Root Mean Squared Error (RMSE) for both training and testing phases. RSME is a standard metric for evaluating performance of regression tasks. We also computed the Mean Squared Error (MSE) and R-squared (R2) values. MSE gives a straightforward way to interpret how big the magnitude of error is.

Baseline for comparison: When predicting Close value of stock prices, achieving low PMSE and high MSE is ideal. They represent the accuracy of prediction of the model. On the other hand we aim for a high R2 value (close to 1) - this indicates that that model is able to apprehend the variance in the Close prices of the stocks. We use these metrics as our baseline for evaluating the performance of our models.

Results

SimpleRNN

- Training RMSE: 0.0024
- Testing RMSE: 0.0018
- MSE: 4.6422e-06
- R2: 0.9999

The SimpleRNN model shows high accuracy with very low RMSE and MSE. It also has a almost perfect R2 value

LSTM

- Training RMSE: 0.0026
- Testing RMSE: 0.0021
- MSE: 8.7673e-06
- R2: 0.9998

The LSTM also shows high accuracy but it has slightly higher RMSE and MSE compared to SimpleRNN (which was not expected)

Linear Regression

- MSE: 0.6186

The linear regression shows relatively higher MSE showing that this may not be as accurate as the other 2 models.

Discussions

Implications

- The high accuracy and R-squared value for the SimpleRNN and LSTM models suggest that the model has learnt the underlying trends and patterns but it does raise concern about the high level of accuracy may be related to overfitting - maybe the model has learned the noise and fluctuation as well, and will not be able to generalise training data in the future.

- LSTM and its ability to learn long term temporal dependency was demonstrated through its high accuracy. LSTM is able to do so because of its internal memory cells that handle sequences with longer time intervals.

- We chose 50 units and 100 epochs as our hyperparameters. The hyperparameter tuning suggested the following best hyperparameters for SimpleRNN and LSTM:

  Best Hyperparameters for RNN: {'batch_size': 32, 'dropout_rate': 0.6, 'epochs': 100, 'units': 32}

  Best Hyperparameters for LSTM: {'batch_size': 128, 'epochs': 50, 'units': 128}

  However, there is a trade-off between model performance and how computationally expensive a model is, so we opted for a balance and performed training with the chosen hyperparameters. More exhaustive hyperparameter tuning would help improve the reliability of the current design

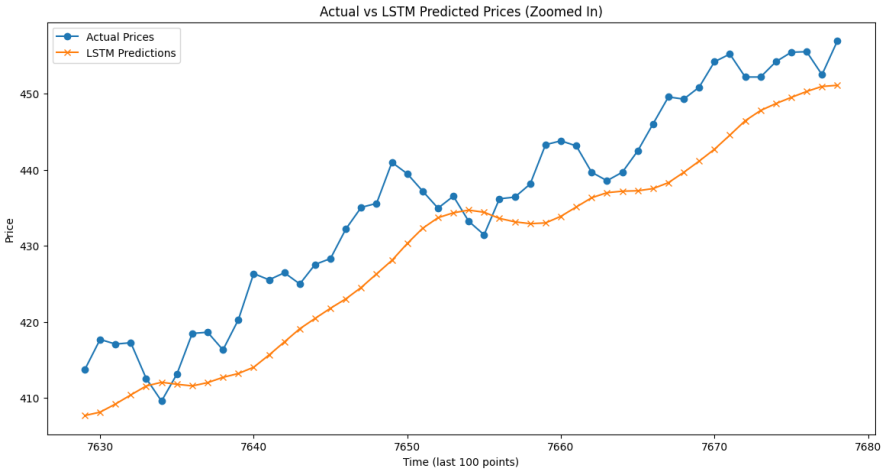
- Even Though a MinMaxScalar normalises a data, we need to investigate further if there was data leakage from the test set during training - which may have led to very high accuracy

- We should further experiment and investigate the relevant features for training. Our dataset contained other features that we did not include in training such as Day (of the month), Weekday, Week (of the year), Month, Year. These features could help to infer other trends in market data - we can do feature engineering to see if these features are relevant.

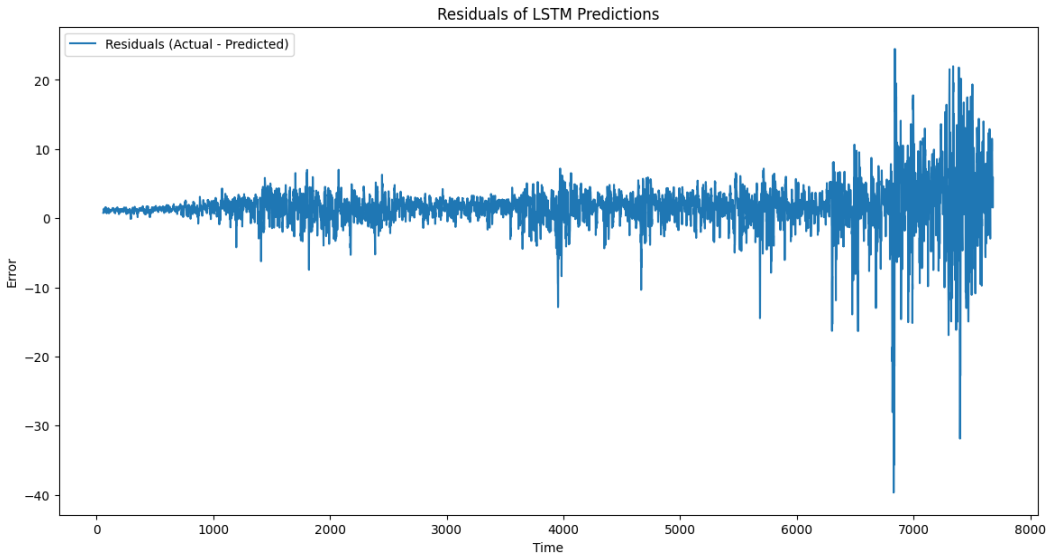
- To capture the real world essense of the stock market and its volatility, we can use an appropriate dataset for sentiment analysis. Exploring more advanced methods such as SARIMA, Prophet or techniques such as seasonal decomposition may be beneficial. Also considerations for noise may be beneficial.

Strengths

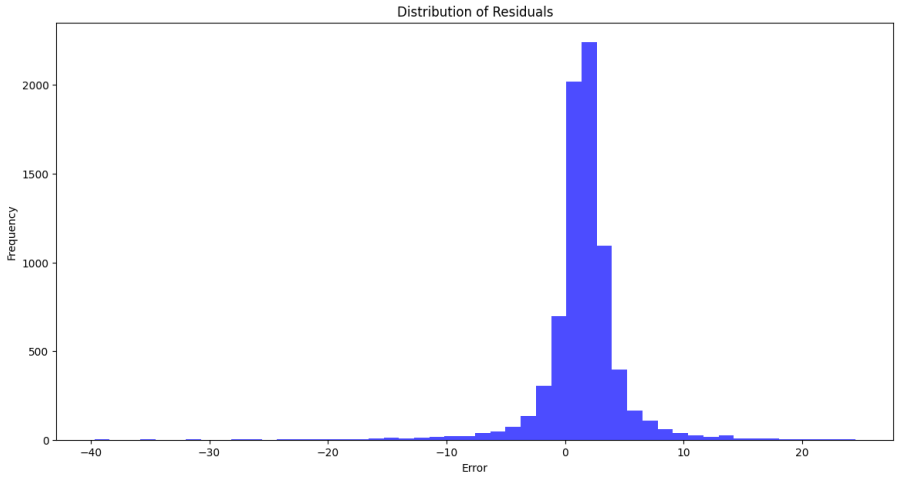
- The LSTM model displays significant strength in capturing strength and models ability to predict underlying patterns. So the project has potential to forecast future Close values given the historical data. This can give traders a good estimation of how stock prices are moving.



- Residual plot shows that the model is stable across majority of the data and residual is dense around the 0 axis value - which means that for the majority of data the model stays close to the actual values:



- The normal distribution also shows that the error is distributed around zero, which can make the model reliable for trader/ market experts to integrate into their trading strategy:



- The models (RNN & LSTM) chosen are well suited for time-series forecasting. While there is significant room for improvement in design of the architecture, the fundamental tools are appropriate for the task. Comparing it to other works, it is sufficiently different from anything else attempted. LSTM also helps to deal with the vanishing gradient issue that can occur in RNNs. In literature, hybrid models perform better than simple RNN models by themselves.

Limitations

- The plot of residuals for the LSTM model shows that even though the residuals appear stable until the 6000th date point, the fluctuation increases after that. This suggests that the models predictability may deteriorate over time

- The previous limitation may also suggest a problem with overfitting, where the model does not perform well on new market trends. Together with the first limitation, this can make the LTSM model unreliable for a lot of traders who want to rely on the latest market trends for trading stocks. As a result, the model might be deemed as less useful.

- The error distribution shows that there are outliers with high error, even though most of the distribution is around zero. Having these errors prevents the design from making accurate predictions.

- Besides, there are limitations of the models on their own, leading to the limitation of the whole predictive process. RNN has vanishing gradient problem, lack of homogenous seasonal patterns and possibly lack of long-term dependencies in the data. LSTM is simple. Using more advanced architectures such as transformer may be better suited

- Lack of changing characteristics of the market that cannot be extracted from price data. It would require further analysis of sentiment in addition to technical analysis. Since sentiment is a powerful driver in the stock market data, not accounting for that in our predictive design may drive away traders from using this.

Future directions

Instead of predicting the closing price, our focus could shift to informing trading decisions. This adds complexity to the application as trading decisions are based on many factors (such as investment and liquidation periods, market outlook, etc) other than current price. We can incorporate interesting datasets that allows us to perform sentiment analysis because sentiments are a substantial driving force in the stock market.

Providing more options in this regard would give the user targeted guidance. Additionally, advanced techniques to develop a more robust architecture should be explored.

Additional Questions

1. What is the feedback that you found useful from the peer evaluation?
   - Include more numeric results and metrics
   - Similar applications and our contribution
   - Improve discussion about strengths and limitations.

2. What changes did you make based on the feedback from peer evaluation?

   - Included more details in the discussion section to explain our current model, compare it to literature and where it stands in relation to other advanced architectures.
   - Added additional notes comparing our project to other similar applications and primary research done that is similar.

References

1. https://www.sciencedirect.com/science/article/pii/S0957417415005126
2. https://ieeexplore.ieee.org/abstract/document/6690034
3. https://arxiv.org/abs/1607.01958
4. https://www.kaggle.com/code/gkitchen/s-p500-spy-prediction/notebook
5. https://towardsdatascience.com/temporal-loops-intro-to-recurrent-neural-networks-for-time-series-forecasting-in-python-b0398963dc1f
6. https://encord.com/blog/time-series-predictions-with-recurrent-neural-networks/
7. https://machinelearningmastery.com/time-series-prediction-lstm-recurrent-neural-networks-python-keras/
8. https://github.com/cjhutto/vaderSentiment
9. https://www.tandfonline.com/doi/full/10.1080/00051144.2023.2217602
10. https://technology.gov.capital/how-does-the-adam-optimizer-handle-noisy-or-inconsistent-gradients