Predicting The Stock Market Using Machine Learning

Alden Park
2023A CPE 695-WN
Stevens Institute of Technology
Hoboken, USA
apark4@stevens.edu

JiaHui Lian
2023A CPE 695-WN
Stevens Institute of Technology
Hoboken, USA
jlian@stevens.edu

Johnny Guamanquispe
2023A CPE 695-WN
Stevens Institute of Technology
Hoboken, USA
jguaman2@stevens.edu

***Abstract*—Predicting the stock market has always been a challenging feat. Due to the fluctuating and volatile nature of stock markets. Stock market prices are constantly changing. They are easily influenced by world events and other external factors, which makes them almost unpredictable. This project is about predicting the stock market using machine learning. By using a Random Forest classifier model with backtesting to solve this problem. Our major contribution is building our own backtesting function to accurately measure the predictions of our model and allow fine-tuning parameters for greater accuracy, which gives us an advantage over existing solutions. This report will further discuss our machine-learning model and approach for predicting the stock market.**

I. Introduction

This project aims to predict the stock market using machine learning. The stock market is a highly volatile and unpredictable environment, making it difficult to predict. The project uses a Random Forest classifier model with backtesting to solve this problem. The major contribution of this project is building our own backtesting function to accurately measure the predictions of our model and allow fine-tuning parameters for greater accuracy. This gives us an advantage over existing solutions.

There are several traditional methods that are widely used today. One such method is to determine the accurate product or service value based on the company's financial report. This provides insights into the strength and economic conditions of the country, which can make it appealing for investors. Ultimately, if the intrinsic value is higher than the market value, then it is a good investment; otherwise, it is not. Our model and approach predict the future stock market based on the historical stock market dataset. The model we would be using is a Random Forest Classifier with backtesting. Backtesting means we are going to train the model with data from a certain time period and test its performance on older data.

II. Related Work

Stock market prediction has been a topic of interest in both the finance industry and academia for a long time. Existing solutions typically fall into two main categories: traditional statistical methods and modern machine-learning techniques. Traditional statistical methods have been extensively used for time-series prediction tasks, including stock price prediction. Autoregressive Integrated Moving Average (ARIMA) models are commonly utilized due to their simplicity and interpretability [1]. However, the assumption of linearity in ARIMA models may not capture the complex, nonlinear nature of stock markets. Another statistical approach is the Generalized Autoregressive Conditional Heteroskedasticity (GARCH) model [2]. It is efficient at modeling volatility in financial markets but may struggle with the noisy and non-stationary nature of stock price data.

With the advent of artificial intelligence and machine learning, a multitude of advanced techniques have been applied to predict stock prices. Artificial Neural Networks (ANN) have been successfully used in various prediction tasks [3]. They are capable of modeling complex, non-linear relationships in data but suffer from problems such as overfitting and a lack of interpretability. Recurrent Neural Networks (RNN) and their variant, Long Short-Term Memory (LSTM) networks, are particularly powerful for sequential data, making them a popular choice for time-series prediction [4]. However, training these networks can be computationally intensive and requires large amounts of data.

More recent approaches focus on the fusion of machine learning models and statistical methods, or the combination of different machine learning models, to improve prediction performance [5, 6]. The idea is to leverage the strengths of different models to overcome their individual weaknesses. However, the complexity of these hybrid models could make them difficult to implement and interpret.

The existing solutions demonstrate a trade-off between performance and complexity. This project aims to explore this trade-off and adopt the most suitable model for stock price prediction. Our goal is to balance prediction accuracy with computational efficiency and interpretability.

III. Our Solution

A. Description of Dataset

The dataset utilized in this study is directly sourced from Yahoo Finance, a well-known financial website that provides historical stock price data for various companies. The dataset consists of daily stock price information for a specific company over a defined period. Each record (row) in the dataset represents a trading day and contains essential features, including:

- Date: The date of the trading day.

- Open: The opening price of the stock on that trading day.
- High: The highest price the stock reached during the trading day.
- Low: The lowest price the stock reached during the trading day.
- Close: The closing price of the stock on that trading day.
- Volume: The trading volume, representing the total number of shares traded on that day.

To gain initial insights into the data, a simple line plot can be used, depicting the stock's closing prices over time and revealing the general trend in the data.

In the pre-processing phase, we will perform the following steps: Removing Irrelevant Columns: The dataset contains additional columns, such as dividends and stock splits, which are not relevant for predicting the stock price. Therefore, we will remove these two columns from the data. Adding Prediction Price Column: To facilitate prediction tasks, we will insert a new column representing the target variable, i.e., the predicted stock price. Creating a Target Column: We will include a new "target" column that indicates whether "tomorrow's" price is greater or less than the previous day's price. This column will be helpful for classification tasks. Limiting Data Range: To avoid outliers caused by significant shifts in the stock market, like the dotcom bubble, we will limit the data to records from after 1990.

Once the dataset is cleaned and pre-processed, we will divide it into two sets:

1) Training Set: This set will be used to train the machine learning model, enabling it to learn patterns and relationships within the data.
2) Test Set: The test set will serve to evaluate the model's performance on unseen data, providing insights into its predictive capabilities.

By following these steps, the dataset will be appropriately prepared for training and testing the machine learning model.

B. Machine Learning Algorithms

There a selection of algorithms will focus on those that have proven effective for time-series data, such as RandomForestClassifer, Recurrent Neural Networks (RNNs), Long Short-Term Memory (LSTM), and Auto Regressive Integrated Moving Average (ARIMA). Each model's performance will be evaluated and compared using various metrics, including the mean squared error (MSE), mean absolute error (MAE), and R-squared.

1) Algorithm 1: RandomForestClassifer: The first algorithm our focus will be on employing a RandomForestClassifier as our machine learning model. The choice of RandomForestClassifier offers several advantages in our context. This classifier utilizes randomized decision trees with randomized parameters, and the results from multiple trees are averaged, effectively combating overfitting issues in the model.

One of the key reasons we opted for RandomForestClassifier is its ability to capture non-linear patterns in the data. In financial markets, such as our stock market dataset, the relationships between features and the target variable are often non-linear. For instance, the open price of a stock may not have a straightforward linear correlation with the target variable; a higher open price does not necessarily guarantee a higher target value. With RandomForestClassifier, we can effectively handle these non-linear relationships, making it a suitable choice for our dataset.

2) Algorithm 2: Long Short-Term Memory (LSTM): Long Short-Term Memory (LSTM) is a specialized type of recurrent neural network (RNN) designed to recognize patterns in sequential data. Unlike traditional RNNs that struggle with long-term dependencies due to the vanishing and exploding gradient problems, LSTMs are equipped with input, forget, and output gates along with a cell state. These components enable LSTMs to effectively store or discard information across sequences, allowing them to learn and remember over extended periods.

Stock prices represent time-series data, where the sequence and order of data are vital. LSTMs, with their design tailored for sequence recognition, are therefore well-suited for stock price predictions. They can handle the volatility of financial markets by capturing long-term dependencies and reducing noise in the data. Moreover, their capability to integrate multiple features, such as trading volume or technical indicators, enhances their prediction potential.

3) Algorithm 3: Multilayer perceptron (MLP): A Multilayer Perceptron (MLP) is a feed-forward artificial neural network that generates a set of outputs from a set of inputs. The model consist of multiple layers of nodes where each layer is connected to the next layer. Each node in the network (apart from the input nodes) has a nonlinear activation function.

MLPs are particularly suited for stock price prediction due to their ability to model complex non-linear relationships, a characteristic intrinsic to financial markets. The market's multifactorial influences, which are often intertwined and non-linear, can be captured by the MLP's architecture. Additionally, MLP's adaptability ensures that as new market data emerges, the network can be updated to accommodate evolving patterns, ensuring that predictions remain relevant.

C. Implementation Details

1) Algorithm 1: RandomForestClassifer: To implement the algorithm, we will utilize the features "close," "volume," "open," "high," and "low" to predict our target variable. The model will be trained on the training data, and we will evaluate its performance by comparing the predicted values to the actual "target" values.

During the configuration of the RandomForestClassifier, there are several essential parameters that warrant consideration. Notably, the parameter 'n_estimators' emerges as a pivotal factor. It dictates the number of these constituent decision trees that will collaboratively form the final ensemble model. The magnitude of this value can significantly influence the model's performance and its propensity to generalize well on unseen data. The initial parameter was set at 100.

Additionally, the parameter 'min_samples_split' assumes a crucial role in the endeavor to strike a balance between model accuracy and overfitting mitigation. This parameter pertains to the minimum number of samples required to further partition a node during the tree-building process. By increasing this value, the model becomes more conservative in its decision-making, prioritizing generalization capabilities over the potential for capturing noise in the training data. This was also set at 100.

To fortify the reliability of the outcomes and ensure reproducibility, the 'random_state' parameter comes into play. By seeding the random number generator with a specific value, you can ensure that the model's behavior remains consistent across different runs. This is particularly valuable when seeking to compare different model configurations or when sharing the codebase with collaborators. This was set at 1.

To quantify the model's performance, we have chosen the sklearn precision-score as our error metric. The precision-score measures the precision of the model's predictions, i.e., the ratio of correctly predicted positive instances to all instances predicted as positive. In our base model evaluation, the precision-score value yielded an accuracy of about 54%.

Considering that our dataset is a "time-series" dataset, we are cautious about using traditional cross-validation techniques during model training. Cross-validation might inadvertently introduce data leakage, as it involves using "future" data to predict the past. Consequently, the model could appear to have an impressive prediction rate during training but fail drastically when applied to new or real-world data. To prevent this issue, we have decided to implement backtesting.

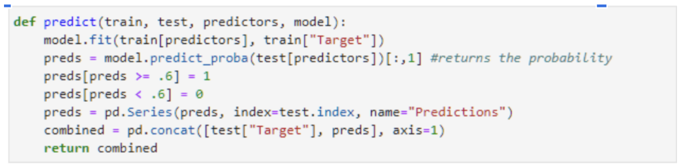
Backtesting is an appropriate approach for our time-series data as it simulates how the model would have performed in the past based on historical data. By withholding certain periods of the data for testing purposes, we can train the model on earlier data and evaluate its performance on unseen, more recent data. This approach allows the model to genuinely learn how to predict future data without being influenced by future information during the training phase.

In the process of backtesting, it is imperative to secure a sufficient volume of data for training the model effectively. Given that a standard trading year encompasses 250 trading days, a strategic approach involves initiating the backtest with a substantial historical dataset, spanning a decade's worth of trading years, thus encompassing a total of 2500 days.

This methodology entails a gradual advancement, beginning with training the model over a single trading year, which corresponds to 250 days. The backtest's initial phase involves training the model on the initial ten years of data. Subsequently, the process unfolds by iteratively training on the data from one trading year and employing this trained model to forecast outcomes for the subsequent year. This iterative technique functions as follows: The model is initially nurtured on the data from the first ten years, enabling it to grasp patterns and relationships. With this foundation established, the model is then deployed to predict the outcomes of the eleventh year, utilizing the wealth of information from the preceding ten. The cycle continues, progressively feeding an additional year's worth of data into the training process while harnessing the predictive capabilities honed over the prior years.In essence, this approach fosters a dynamic evolution of the model's predictive abilities as it assimilates new data. This unfolding sequence of training and forecasting mirrors the market's temporal progression, allowing for a nuanced assessment of the model's adaptability and efficacy over time. With the implementation of backtesting, we yielded in 53% accuracy. Although our precision did not change by much, this allows us to validate our model.

In an attempt to refine our precision, we introduced a probability-based adjustment into our initial prediction function. By incorporating a novel element, "predict_proba," into our function, we introduce the capacity to assess the probability of stock price fluctuations in line with our dataset. Subsequently, a crucial step entails establishing a threshold that governs the inclusion of these data points within our model. This strategic filtering mechanism augments our model's robustness and engenders heightened confidence in its predictive capabilities.

This refinement holds particular significance as it leads to a reduction in the overall frequency of trading days, a trade-off justified by the amplified precision in discerning instances of price elevation. This deliberate approach resonates deeply with our overarching strategy: to predict price shifts with a focus on accuracy rather than pursuing trades incessantly, a practice that could inevitably lead to financial losses. In essence, this approach acknowledges the nuanced nature of financial markets, where not every day presents an optimal opportunity for trading. By leveraging the probabilistic insights of "predict_proba" and employing a judicious threshold, we strike a harmonious balance between precision and trading frequency, ultimately enhancing our model's predictive prowess and ensuring a more strategic and informed trading decision-making process.



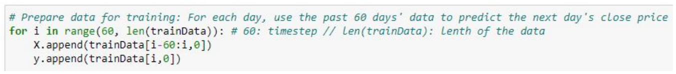
```
def predict(train, test, predictors, model):
    model.fit(train[predictors], train["Target"])
    preds = model.predict_proba(test[predictors])[:,1] #returns the probability
    preds[preds >= .6] = 1
    preds[preds < .6] = 0
    preds = pd.Series(preds, index=test.index, name="Predictions")
    combined = pd.concat([test["Target"], preds], axis=1)
    return combined
```

Fig. 1. Adjusted Predict Function

2) Algorithm 2: Long Short-Term Memory (LSTM): The second machine learning algorithms we used to solve the problem is Long Short-Term Memory (LSTM). The project leverages the power of Long Short-Term Memory (LSTM) networks, a type of recurrent neural network (RNN), to predict the stock price of the S&P 500 (S&P 500 can somehow present the whole market but specified stock can also be used for this algorithm). LSTMs are particularly suited for sequence prediction problems, given their ability to remember past information and detect long-term dependencies.

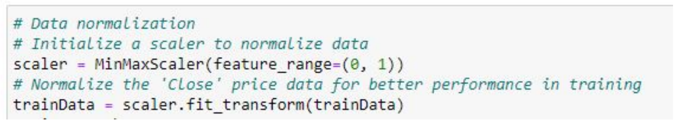
Data Acquisition and Pre-processing: The data is fetched using the yfinance library, which offers tools to easily extract financial data. The dataset comprises the historical data of the S&P 500 index, specifically focusing on the 'Close' price column. For the sake of this prediction, only data from 1990 onwards is considered. The columns "Dividends" and "Stock Splits" are discarded since they aren't essential for this analysis.

Normalization and Training Data Preparation: The data undergoes a normalization process using MinMaxScaler to make the LSTM model's training more effective. For the training dataset, for each day, the past 60 days' data is used to predict the subsequent day's closing price.



```
# Prepare data for training: For each day, use the past 60 days' data to predict the next day's close price
for i in range(60, len(trainData)): # 60: timestep // len(trainData): lenth of the data
    X.append(trainData[i-60:i,0])
    y.append(trainData[i,0])
```

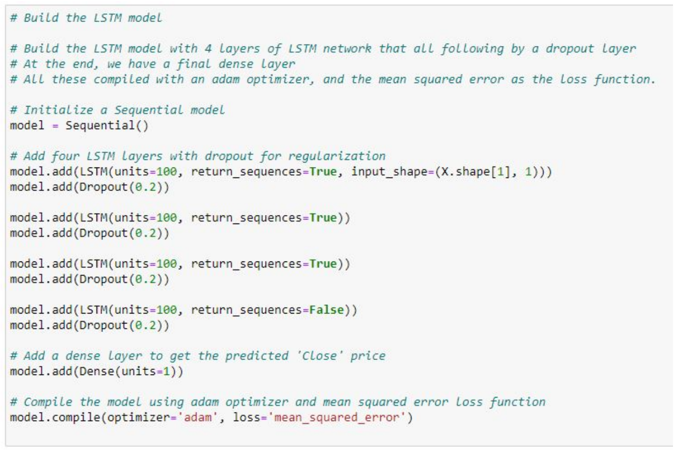
Fig. 2. Training Data Preparation



```
# Data normalization
# Initialize a scaler to normalize data
scaler = MinMaxScaler(feature_range=(0, 1))
# Normalize the 'Close' price data for better performance in training
trainData = scaler.fit_transform(trainData)
```

Fig. 3. Data Normalization

Model Architecture: The core of the project is the LSTM-based model. The model comprises four LSTM layers, each followed by a dropout layer to prevent overfitting. The network concludes with a dense layer that outputs the predicted 'Close' price for the next day.



```
# Build the LSTM model

# Build the LSTM model with 4 layers of LSTM network that all following by a dropout layer
# At the end, we have a final dense layer
# All these compiled with an adam optimizer, and the mean squared error as the loss function.

# Initialize a Sequential model
model = Sequential()

# Add four LSTM layers with dropout for regularization
model.add(LSTM(units=100, return_sequences=True, input_shape=(X.shape[1], 1)))
model.add(Dropout(0.2))

model.add(LSTM(units=100, return_sequences=True))
model.add(Dropout(0.2))

model.add(LSTM(units=100, return_sequences=True))
model.add(Dropout(0.2))

model.add(LSTM(units=100, return_sequences=False))
model.add(Dropout(0.2))

# Add a dense layer to get the predicted 'Close' price
model.add(Dense(units=1))

# Compile the model using adam optimizer and mean squared error loss function
model.compile(optimizer='adam', loss='mean_squared_error')
```

Fig. 4. Model Architecture LSTM

Debugging and Performance Optimization: In any machine learning project, the initial results rarely match expectations. The same applied to this implementation, where a few challenges arose: 1. Overfitting: The initial model faced issues with overfitting, which means it performed exceedingly well on the training data but failed on the test data. The introduction of dropout layers after each LSTM layer is a technique employed to tackle this problem. By randomly setting a fraction of the input units to 0 during training, the dropout layers help prevent over-reliance on any single neuron, thus promoting model generalization. 2. Learning Rate Tuning: The 'adam' optimizer used for compiling the model has adaptive learning rates. A deeper dive into the parameters of this optimizer could help improve convergence speed and, potentially, the final model's performance. 3. Model Evaluation: To evaluate the model's real-world utility, the direction of stock movement (upward or downward) is predicted. Precision, a key metric in this analysis, gives insights into the reliability of the model's positive (upward movement) predictions. An increase in precision means fewer false positives, which is vital for decision-making in stock trading.

In conclusion, the LSTM-based model presents a promising technique for stock price prediction. The combination of data preprocessing, LSTM architecture, and performance tuning techniques provides a robust solution to predict stock market trends. However, it's crucial to acknowledge the inherent unpredictability of stock markets and consider external factors and broader market knowledge before making any trading decisions.

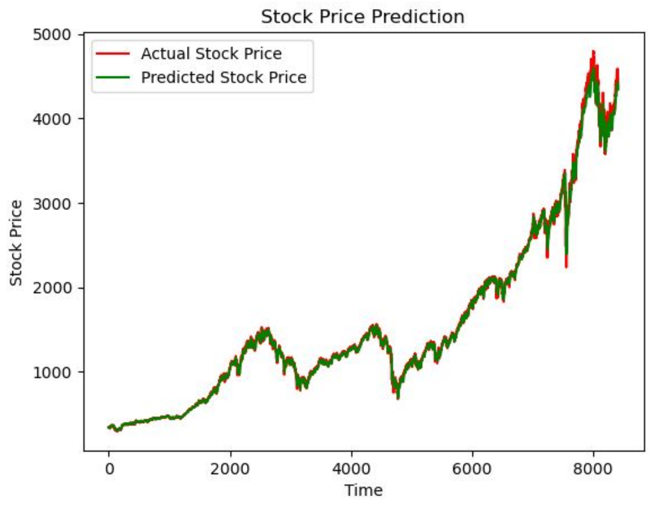
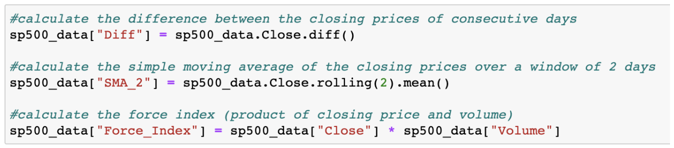


Fig. 5. LSTM Predicted Stock Price Vs. Actual Stock Price

3) Algorithm 3: Multilayer perceptron (MLP): A Multilayer Perceptron (MLP) is a feed-forward artificial neural network that generates a set of outputs from a set of inputs. We used an MLP model for predicting the stock market particularly for the S&P 500, we trained the model on historical stock prices and used it to predict future prices for the S&P 500. We fine tuned the MLP parameters to get the best accuracy results and prevent over fitting.

Data Acquisition and Pre-processing: The data was extracted using the yfinance library. The dataset contained the historical data of the S&P 500 index. The data obtained is from the last 10 years closing price of the S&P 500 index. Most of the columns were discarded as they aren't essential for this analysis. Our focus was on he closing price of the S&P 500 index.

Training Data Preparation: The data was cleaned by removing unnecessary columns and create a new set of columns to better help trained the model.



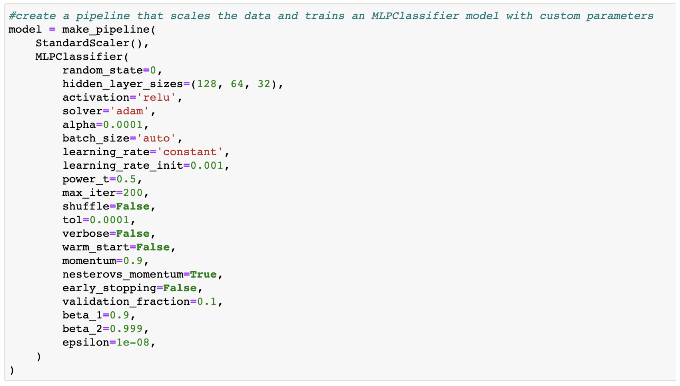
```
#calculate the difference between the closing prices of consecutive days
sp500_data["Diff"] = sp500_data.Close.diff()

#calculate the simple moving average of the closing prices over a window of 2 days
sp500_data["SMA_2"] = sp500_data.Close.rolling(2).mean()

#calculate the force index (product of closing price and volume)
sp500_data["Force_Index"] = sp500_data["Close"] * sp500_data["Volume"]
```

Fig. 6. MLP Training Columns

Model Architecture: The selected model is the MLP model. Which consists of multiple layers of nodes, each layer connected to the next layer in a directed graph.



```
#create a pipeline that scales the data and trains an MLPClassifier model with custom parameters
model = make_pipeline(
    StandardScaler(),
    MLPClassifier(
        random_state=0,
        hidden_layer_sizes=(128, 64, 32),
        activation='relu',
        solver='adam',
        alpha=0.0001,
        batch_size='auto',
        learning_rate='constant',
        learning_rate_init=0.001,
        power_t=0.5,
        max_iter=200,
        shuffle=False,
        tol=0.0001,
        verbose=False,
        warm_start=False,
        momentum=0.9,
        nesterovs_momentum=True,
        early_stopping=False,
        validation_fraction=0.1,
        beta_1=0.9,
        beta_2=0.999,
        epsilon=1e-08,
    )
)
```

Fig. 7. MLP Model

One of the observed performance was a lower accuracy. In order to improve the performance of the MLP, we tried different activation functions for the hidden layers, increased the number of hidden layers and neurons in each layer, and tune the hyper parameters as needed. The increase in accuracy was significant from initial lowest score of 32 into an much improved 48.3 score after the fine tuning was performed.

D. Comparison

In the realm of stock price prediction, comparing the Random Forest Classifier (RFC), Long Short-Term Memory (LSTM), and Multilayer Perceptron (MLP) provides a comprehensive view of how different algorithms perform in capturing market dynamics.

In the given scenario, the RFC achieves the highest precision of 0.57, followed by the LSTM at 0.52 and the MLP trailing at 0.47. However, interestingly, all three models exhibit the same accuracy of 0.48. This suggests that while RFC is better at correctly predicting the positive class (price rise, for instance), it doesn't necessarily mean it gets more overall predictions right, as indicated by the consistent accuracy across all three models.

RFC's strength in this task might be attributed to its ensemble nature. By creating numerous decision trees and aggregating their outputs, RFC can capture a wide array of features and their potential interactions, offering a more robust prediction in diverse market situations. Additionally, RFC inherently offers feature importance, allowing analysts to gain insights into which factors are most influential in stock price prediction.

On the other hand, LSTMs are specifically designed to handle sequence data, making them well-suited for time-series data like stock prices. Their ability to capture long-term dependencies might explain why their performance is competitive with RFC. However, they might require more intricate fine-tuning and a larger amount of data to showcase their full potential.

MLP's lower precision suggests it might struggle with the intricate and often non-linear relationships within stock market data. While MLPs are powerful tools, their architecture might be too simple to capture the multifaceted influences affecting stock prices without extensive optimization.

In summary, while RFC seems to be the best in terms of precision, it's essential to consider other factors like interpretability, training time, and data requirements. The equal accuracy across models highlights the challenging nature of stock price prediction, where even sophisticated algorithms achieve similar general accuracy levels.

IV. Future Directions

Our future plans moving forward with the project is to refine our parameters and metrics to increase the accuracy of the model prediction. We will look into the other algorithms that may possibly yield a better result than our current models.

1) Parameter Refinement: We will fine-tune the parameters of our models, to optimize their performance on the specific problem at hand. By systematically exploring the hyperparameter space using techniques like grid search or random search, we aim to discover the most effective combination of settings.
2) Metric Optimization: We understand the importance of selecting appropriate evaluation metrics. We will carefully choose and refine the metrics used to assess the model's performance, ensuring they align with the project's objectives and capture the desired outcomes accurately.
3) Algorithm Exploration: We recognize the need to explore other algorithms that might offer superior predictive capabilities. We will investigate a variety of algorithms, such as Gradient Boosting Machines, Support Vector Machines, other Neural Networks, and advanced models like Transformers, depending on the nature of our data and problem.
4) Feature Engineering: We will carefully analyze our feature set and explore opportunities for feature engineering. By extracting more relevant information or creating new informative features, we aim to improve the model's ability to capture patterns and make accurate predictions.
5) Error Analysis: Understanding the model's weaknesses is crucial. We will conduct a comprehensive analysis of the errors it makes and identify potential sources of mispredictions. This analysis will guide us in focusing on specific areas that need improvement.

V. Conclusion

In conclusion, this project represents an ambitious endeavor to predict the stock market using machine learning techniques, acknowledging the inherent challenges posed by the market's high volatility and unpredictability. The key focus of the project lies in leveraging a Random Forest classifier model with a novel backtesting function, which plays a crucial role in accurately measuring model predictions and fine-tuning parameters to achieve greater accuracy. This unique approach provides a notable advantage over existing solutions in the field.

Traditional methods, such as evaluating a company's financial reports and intrinsic value, remain widely used in today's stock market analysis. These methods offer valuable insights into a company's strength and the economic conditions of the country, guiding investment decisions based on potential discrepancies between intrinsic and market value. In contrast, the current project's approach involves predicting future stock market behavior by relying on historical stock market data and utilizing the Random Forest Classifier with backtesting.

It's important to acknowledge that predicting the stock market remains a complex and challenging task, and no method, including machine learning approaches, can guarantee absolute accuracy due to the market's dynamic nature. However, the incorporation of machine learning and the development of a proprietary backtesting function represent valuable contributions to the field, providing potential benefits in enhancing predictive capabilities.

Ultimately, the success of this project will depend on how effectively the Random Forest model and backtesting function can adapt to the unique characteristics of the stock market and generate reliable predictions. The approach's potential advantages, such as the ability to capture complex patterns and trends in historical data, make it an intriguing avenue for exploring and understanding market behavior.

As with any financial forecasting endeavor, it is crucial to exercise caution and recognize that all predictions come with inherent risks. This project's findings and insights can serve as valuable additions to the broader landscape of stock market prediction methodologies, but users should always approach financial decisions with careful analysis, diversification, and risk management strategies.

In the task of stock price prediction, a comparison of the Random Forest Classifier (RFC), Long Short-Term Memory (LSTM), and Multilayer Perceptron (MLP) reveals distinct capabilities of each model. RFC emerges as the leader in precision with 0.57, outpacing both LSTM's 0.52 and MLP's 0.47. Yet, all models deliver an identical accuracy of 0.48, indicating a shared challenge in overall prediction quality. The ensemble nature of RFC allows it to adeptly handle diverse market scenarios, benefiting from the combined strength of multiple decision trees. LSTM, tailored for sequence data, exhibits promise, especially when fine-tuned for time-series stock data. Meanwhile, MLP, despite its versatility, might require advanced fine-tuning to effectively handle stock market complexities.

Despite the variations in precision, the consistent accuracy across models underscores the intricate challenge of stock price prediction. While RFC offers a precision advantage, the choice of model should be informed by specific project needs, data availability, and desired interpretability.

References

[1] Box, G.E., Jenkins, G.M., Reinsel, G.C., and Ljung, G.M. "Time series analysis: forecasting and control." *John Wiley & Sons*, 2015.

[2] Bollerslev, T. "Generalized autoregressive conditional heteroskedasticity." *Journal of econometrics*, vol. 31, no. 3, pp. 307-327, 1986.

[3] Bishop, C.M. *Neural networks for pattern recognition*. Oxford university press, 1995.

[4] Hochreiter, S., and Schmidhuber, J. "Long short-term memory." *Neural computation*, vol. 9, no. 8, pp. 1735-1780, 1997.

[5] Zhang, G.P., and Qi, M. "Neural network forecasting for seasonal and trend time series." *European Journal of Operational Research*, vol. 160, no. 2, pp. 501-514, 2005.

[6] Tsai, C.F., and Hsu, Y.F. "Combining multiple feature selection methods for stock prediction: Union, intersection, and multi-intersection approaches." *Decision Support Systems*, vol. 50, no. 1, pp. 258-269, 2010.