Binary Classification for Algorithmic Trading

Sichao Bi, Grace Venerus, Xiangyu Xu, Samuel Rector, Quoc Tran
Boston College

May 1, 2025

Abstract

This project explores the use of machine learning and sentiment analysis to predict short-term movements in the S&P 500 index. By combining historical price data, technical indicators, macroeconomic variables, and news sentiment, we aim to forecast binary outcomes—either significant market shifts, defined by a daily movement in the index by greater than 1% or less than 1% change, or simple up/down movements. Our dataset spans from 2015 to early 2025, incorporating features such as moving averages, RSI, MACD, volatility measures, inflation rates, and sentiment scores derived from finance-related news articles using NLTK's Sentiment Intensity Analyzer. We train and compare several classification models, including Gaussian Naive Bayes, Support Vector Machines (SVM), Multi-layer Perceptron (MLP), and Long Short-Term Memory (LSTM) networks. Each model's performance is evaluated using balanced accuracy, confusion matrices, ROC curves, and cross-validation. Results show that while no model universally dominates, each has strengths aligned with specific investor goals—for instance, the neural network offers higher positive predictive power, whereas SVM provides more consistent classification of stable market days. Our findings underscore the potential of hybrid data sources and diverse modeling techniques in improving market prediction, while also highlighting the limits of predictability in volatile financial environments.

1 Introduction

1.1 Overview and Goal

The stock market reflects collective investor sentiment about the future earnings and growth potential of publicly traded companies. Its performance is influenced by a range of factors, including company performance, investor confidence, economic conditions, and global news events. Due to the complex and interconnected nature of these variables, investors cannot rely solely on price patterns or intuition when making decisions on whether to buy, hold, or sell their stocks. Numerous machine learning application models have been implemented for more holistic analysis, but accurately predicting stock prices remains a significant challenge, even for experienced researchers and data scientists.

This project aims to forecast stock price movement using data from Yahoo! Finance's API. By applying various classification techniques, we aim to develop and assess multiple models designed to predict binary stock movements. To enhance our models, we draw upon already existing algorithmic trading models to improve our dataset with features like moving averages and technical indicators. Our primary goal is to evaluate and compare the effectiveness of the selected models in forecasting stock movements. To do this, we will focus on two prediction targets: (1) identifying whether the S&P 500 stock moves significantly the next day - defined as a gain of more than 1% or a loss of more than 1%, and (2) classifying S&P 500 stock price direction as either upward or downward the next day.

The following sections of this paper detail the methods employed in our research, including the Python libraries used, relevant economic data sources, overall methodology, and evaluation metrics. We compare the performance and efficiency of models such as the Naive Bayes Classifier, Support Vector Machine (SVM), Neural Networks, Long Short-Term Memory (LSTM) Networks, and models incorporating Sentiment Analysis. Each model's results are evaluated using metrics such as k-fold cross-validation, confusion matrices, Receiver Operating Characteristic (ROC) curves, and, specifically for the SVM and LSTM models, a learning capacity curve analysis.

1.2 Related Work

Chhajer et al. leverage Support Vector Machines (SVM) to achieve highly accurate classification of new stock data by constructing an optimal hyperplane that separates n-dimensional spaces into distinct groups. J. Levendovszky et al. implement a Feedforward Neural Network as a fast trading algorithm, aiming to identify the highest-return trading portfolio based on estimated forward conditional probability distributions that closely follow stock price movements. Junming Yang et al. explore the use of Long Short-Term Memory (LSTM) networks to predict stock returns, utilizing cross-entropy cost as the lost function to enhance forecasting accuracy. Additionally, our sentiment analysis is informed by articles and public statements made by Jerome Powell, the current Chairman of the Federal Reserve, whose words can have a direct and significant influence on stock market behavior.

2 Methodology

2.1 Dataset and sources

The particular stock we use in the study is The Standard and Poor's 500, or the S&P 500 (Ticker: ^GSPC), is a stock market index tracking the stock performance of 500 leading companies listed on stock exchanges in the United States and is one of the most reliable and most commonly used financial indicators of the market. The stock data ranges from March 1, 2015 to February 27, 2025.

Yahoo Finance API

We extracted the following base features from the Yahoo Finance API: Open, High, Low, and Close prices. From these, we engineered additional features to better capture stock volatility and price behavior relative to short- and long-term trends (short-term: 5-50 days, long-term: 100-200 days):

- daily_return: daily closing stock price change in percent
- ma_5: 5-day simple moving average of the closing price
- ma_200: 200-day simple moving average of the closing price
- close_ma_5_ratio: ratio of the daily closing price to the 5-day moving average
- close_ma_200_ratio: ratio of the daily closing price to the 5-day moving average
- volatility_5: rolling standard deviation of the daily return for the past 5 days
- volatility_200: rolling standard deviation of the daily return for the past 200 days
- range_high_low: daily price range, calculated as the difference between the daily high and low prices, measuring daily price volatility
- close_to_high: ratio of the daily closing price to the daily high
- close_to_low: ratio of the daily closing price to the daily low

Moving Average

We incorporate technical analysis features and sentiment analysis based on the moving averages technique. A moving average (MA) is a widely used statistical technique that smooths out short-term fluctuations and highlights longer-term trends in time-series data. For the stock price, we implement a simple moving average over fixed-size window given by

$$\mathrm{SMA}_t = \frac{1}{k} \sum_{i=0}^{k-1} x_{t-i} \tag{1}$$

where

- k is the window size
- x_{t-i} is the value of the time series at time (t-i)

Moving Volatility

We incorporate volatility-based technical indicators to capture the stock's recent price stability or instability. Moving volatility is calculated as the rolling standard deviation of daily returns over a window. This technique allows us to quantify how much the asset's price fluctuates around its mean over different time frames.

$$\sigma_t = \sqrt{\frac{1}{w-1} \sum_{i=t-w+1}^{t} (r_i - \bar{r}_t)^2} \tag{2}$$

where

- w is the window size
- r_i is the daily return at time i
- $\bar{r_i}$ is the mean of returns in the window ending at t
- t is the current time index

Sentiment Analysis

We utilize a sentiment analysis model focused on articles and speeches by the Chairman of the U.S. Federal Reserve, Jerome Powell. Powell's statements can significantly impact the stock market, commodities, and bond yields on a daily basis, as investors closely interpret his words for signals on potential interest rate hikes, cuts, or risks of inflation and recession. In essence, Powell's language shapes the direction of the markets and the broader economy, making it crucial for public and private investors to carefully analyze every word of his speech to maximize portfolio returns. Sentiment analysis aims to identify the sentiment expressed in a text, marking the percentages of features that is positive, neutral, or negative. For our analysis, we use the Sentiment Intensity Analyzer from the Python package Natural Language Toolkit (NLTK), whose function polarity_scores also provides a compound sentiment score ranging from -1 to 1. To gather the data, we use Perigon API to extract news articles from the Internet. Each query is a news article in the finance field from the top 25 finance news organizations in English from March 1, 2015 to Feburary 27, 2025. Note that because of news organizations' copyright policies, the API is unable to extract full articles. However, the API provides a summary of the article, which we use as the text to be analyzed. Two features, published date and summary, are extracted from the API request and used in the sentiment analysis. The results of the sentiment analysis is thereafter merged with the main dataset from the Yahoo Finance API. We implement a exponentially weighted moving average where more recent sentiments are given higher weights than older ones, allowing the model to reflect the evolution and forgetfulness of the market over time.

$$\text{WMA}_t = \frac{\sum_{i=0}^{k-1} w_i x_{t-i}}{\sum_{i=0}^{k-1} w_i} \tag{3}$$

where

- k is the window size
- x_{t-i} is the value of the variable at time t - i
- w_i is the weight assigned to the i-th value

Technical Analysis Library

We use the Technical Analysis (TA) library to extract momentum and trend-based indicators, enabling a deeper analysis of stock movements through different moving averages:

- rsi_14: The Relative Strength Index compares the magnitude of recent gains and losses over 14 days to measure speed and change of stock price movements.

$$\text{RSI}_{14} = 100 - \left(\frac{100}{1 + RS}\right) \tag{4}$$

where RS is The Relative Strength, defined as the ratio of average gains to average losses over the past 14 days.

- macd: Moving Average Convergence Divergence is a trend-following momentum indicator that shows the relationship between two moving averages of prices

$$\text{MACD}_t = \text{EMA}_{12}(P_t) - \text{EMA}_{26}(P_t) \tag{5}$$

where

- P_t: the price (typically closing price) at time t
- $\text{EMA}_{12}(P_t)$: 12-period Exponential Moving Average of the price
- $\text{EMA}_{26}(P_t)$: 26-period Exponential Moving Average of the price
- MACD_t: the Moving Average Convergence Divergence value at time t

Pandas datareader and Federal Statistics

We also incorporate key macroeconomic indicators using the Pandas DataReader module, sourcing data from federal statistics agencies to account for broader economic conditions affecting stock performance:

- cpi: Consumer Price Index (CPI) - Measures the average change over time in the prices paid by urban consumers for a basket of goods and services, reported by the U.S. Bureau of Labor Statistics (BLS)
- fed_fund: Federal Funds Rate - The interest rate at which depository institutions lend reserve balances to other depository institutions overnight, targeted by the U.S. Federal Reserve to influence monetary policy
- unemployment: U.S. Unemployment Rate - Represents the percentage of the labor force that is jobless and actively seeking employment
- us_infla: U.S. Inflation Rate - The rate at which the general level of prices for goods and services rises, indicating a decline in purchasing power over time

Division of Training Set and Test Set

We performed a train-test split with $\frac{2}{3}$ of the dates as the training data and the remaining $\frac{1}{3}$ of the dates as the test data to prevent data leakage. This was an initial problem with our data that caused overfitting, so we made this change.

- Training Set – 12/14/15 - 2/14/22
- Testing Set – 2/15/22 - 1/31/25

2.2 Preprocessing steps

We apply StandardScaler to normalize all stock prices and derived indicators so that they have a mean of 0 and a standard deviation of 1 before any analysis or modeling. Since our features vary widely in scale (for example, U.S. inflation rates are typically single-digit values, while the close_to_high ratio ranges between 0 and 1), standardization ensures that no single feature disproportionately influences the model. This preprocessing step helps all features contribute more equally, leading to improved model performance. We also included the categorical variable day_of_week, which we One-Hot Encoded. We use sklearn Pipeline and FeatureUnion functions as well.

2.3 Models

In this study, we focused on four machine learning approaches to predict the movement of financial market indicator: Naive Bayes classifier, Support Vector Machine (SVM), Feedforward Neural Network (MLP), and Long Short-Term Memory (LSTM) network. Due to the significant randomness and volatility of the financial market, we tried different models to capture the patterns in the data.

Naive Bayes

The Naive Bayes Classifier is a probabilistic model based on Bayes Theorem, based on the assumption that predictors are conditionally independence. (Correlation graph here) Given features X and class y, Bayes classifier estimate

$$P(y \mid X) \propto P(y) \prod_{i=1}^{n} P(x_i \mid y) \tag{6}$$

where

- $P(y \mid X)$ is the posterior probability of class y given features X
- $P(y)$ is the prior probability of class y
- $P(x_i \mid y)$ is the likelihood of feature x_i given class y
- n is the total number of features

It often performs well in high-dimensional data due to low variance. We implement the Gaussian Naive Bayes Variant which assumes each conditional likelihood $P(x_i|y)$ follows a normal distribution. We use this model as our baseline model in order to compare the performance of it with more complex classification algorithms.

Support Vector Machine (SVM)

SVM is a supervised learning algorithm that aims to find the optimal hyperplane that could separate the classes with the max margin. Given a set of training data $\{(\mathbf{x}_i, y_i)\}_{i=1}^{n}$, where $\mathbf{x}_i \in \mathbb{R}^d$ and $y_i \in \{-1, 1\}$, the optimization problem for a hard-margin SVM can be formulated as:

$$\min_{w,b} \frac{1}{2}\|w\|^2 \quad \text{subject to} \quad y_i(w \cdot x_i + b) \geq 1 \tag{7}$$

To capture the patterns, we utilize a polynomial kernel of degree=1.

Neural Network

The Feedforward Neural Network (FNN) is a deep learning model composed of multiple layers, where each layer performs a computation of the form:

$$h_i = \phi\left(\sum_j w_{ij}x_j + b_i\right) \tag{8}$$

Where ϕ is a non-linear activation function. We used ReLU as the activation function, which is simply output the greater number from output and zero. w_{ij} are the weights, and b_i are the biases. The whole network always move forward to next layer, without any cycle or discard any information. The model is trained via backpropagation by minimizing the cross-entropy loss function using stochastic gradient decent. Our MLP model has two hidden layers, with 32 and 2 neurons, respectively. Such a design aims at capturing the non-linear relationship among technical indicators, sentimental indicators and macroeconomic features.

Long Short-Term Memory (LSTM)

LSTM networks are a specialized form of Recurrent Neural Network (RNN) capable of learning long-term dependencies in sequential data. Such a structure is suitable for our problem since it specializes in time-dependent data.

Unlike RNN, LSTMs are designed to avoid the vanishing and exploding gradient problem. Instead of using the same feedback loop to make a prediction, LSTMs use two separate paths: one for long-term memories and one for short-term memories. The structure of an LSTM model is made up of three gates: the input gate, the forget gate, and the output gate.

The cell state tracks the long-term memory, where the omission of weights prevents the gradient exploding problem. The hidden state tracks the short-term memories, and has weights like the ones used in RNN. The input gate is the fed the data example, the forget gate updates the long term memory with the percent that should be remembered, the output gate captures the new short-term memory.

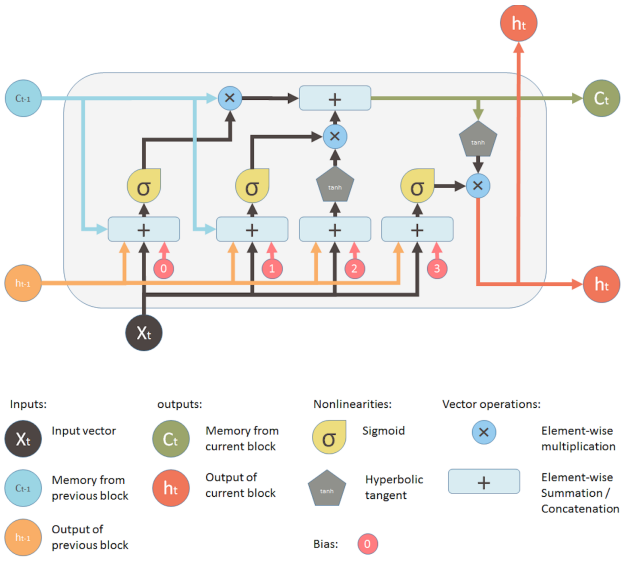

Figure 1: LSTM Architecture, Source: ML Review

We created a custom LSTM model with the following parameters: input size, hidden size, and number of layers. The input to the model is the standardized financial features outlined in the data description.

```
class SimpleLSTM(nn.Module):
    def __init__(self, input_size, hidden_size=64):
        super(SimpleLSTM, self).__init__()
        self.lstm = nn.LSTM(input_size, hidden_size, batch_first=True)
        self.dropout = nn.Dropout(p=0.4)  # 40% dropout
        self.fc = nn.Linear(hidden_size, 1)
        self.sigmoid = nn.Sigmoid()


    def forward(self, x):
        out, _ = self.lstm(x)
        out = self.dropout(out)
        out = out[:, -1, :]
        out = torch.tanh(out)
        out = self.fc(out)
        return torch.sigmoid(out).squeeze()
```

Our LSTM architecture is made up of a single layer with 64 hidden nodes. The architecture consists of an output layer that maps the hidden state to a scalar output. This output is fed to the sigmoid activation function, which produces a probability score between 0 and 1, representing the market moving up or down.

$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{9}$$

Because we are performing a classification task, we define binary cross-entropy loss (BCELoss) as our loss function, with inputs being the predicted label and the true label. This is given by:

$$\mathcal{L}_{\text{BCE}} = -\frac{1}{N} \sum_{i=1}^{N} \left(y_i \log(\hat{y}_i) + (1 - y_i) \log(1 - \hat{y}_i) \right) \tag{10}$$

where

- N is the number of samples in the dataset
- y_i is the true label for the i-th sample (either 0 or 1)
- $\hat{y}_i$ is the predicted probability for class 1 from the model
- $\mathcal{L}_{\text{BCE}}$ is the binary cross-entropy loss (averaged over all samples)

Regularization is performed by applying the ridge penalty (L_2 Norm) given by:

$$\|\mathbf{x}\|_2 = \left(\sum_{i=1}^{n} x_i^2 \right)^{1/2} \tag{11}$$

where

- $\mathbf{x}$ is the parameter vector (e.g., model weights)
- x_i is the i-th component of the vector $\mathbf{x}$
- n is the number of components (i.e., number of features or weights)
- $\|\mathbf{x}\|_2$ is the L_2 norm of $\mathbf{x}$, used as a regularization term

Additionally, we define a dropout layer of 0.4. This reflects the fraction of neurons that are set to zero in an effort to avoid overfitting. Optimization is performed using the Adam Optimizer, with a learning rate of 0.0001, with the following update step:

$$\theta_t = \theta_{t-1} - \alpha \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon} \tag{12}$$

where

- m_t is the Exponentially decaying average of past gradients (first moment estimate).
- α is the Learning rate (step size).
- ϵ is the Small constant for numerical stability.
- $\hat{v}_t$ is the Bias-corrected second moment estimate.
- θ_t is the Model parameters at time step t.
- θ_{t-1} is the Model parameters at time step $t-1$.

2.4 Evaluation metrics

In order to test the performance of the models, we utilize several evaluation methods to measure the out-of-sample classification accuracy. Specifically, we use k-fold cross-validation, confusion matrices, Receiver Operating Characteristic (ROC) curves, and a Learning Curve for the SVM model.

K-Fold Cross Validation

K-fold CV is a robust model validation technique that could estimate the out-of-sample error performance, where we split the dataset into 8 equal-sized segments, and in each iteration, we use one of the segments for validation and the other 7 for training. Thus, making sure the model will not showing a high accuracy but is actually over fitting the data set. Having the accuracy scores for 8 iterations, we calculated the mean and standard deviation so that we have a comprehensive measure on the performance and also the variance.

Confusion Matrix

A confusion matrix is a breakdown of model predictions compared to true classes. It splits the predictions into four categories.

Target: Significant Movement Metrics
Positive cases in our context represent the market (S&P 500 index) has a volatility more than one percent. Negative cases mean that the market is relatively stable.

- True Positives (TP): The model correctly predicts high volatility
- True Negatives (TN): The model correctly predicts market stability
- False Positives (FP): Incorrectly predicts high volatility when the market is actually stable
- False Negatives (FN): Incorrectly predicts stability when the market actually experiences high volatility

Target: Up/Down

- True Positives (TP): Correctly predicts when the market goes up
- True Negatives (TN): Correctly predicts when the market goes down
- False Positives (FP): Incorrectly predicts that the market goes up, when it goes down

- False Negatives (FN): Incorrectly predicts that the market goes down, when it goes up

$$\begin{aligned}
\text{True Positive Rate (TPR)} &= \frac{TP}{TP+FN} \\
\text{False Positive Rate (FPR)} &= \frac{FP}{FP+TN} \\
\text{False Negative Rate (FNR)} &= \frac{FN}{FN+TP} \\
\text{True Negative Rate (TNR)} &= \frac{TN}{TN+FP} \\
\text{Positive Predictive Value (PPV)} &= \frac{TP}{TP+FP} \\
\text{Negative Predictive Value (NPV)} &= \frac{TN}{TN+FN}
\end{aligned}$$

ROC

The Receiver Operating Characteristic (ROC) curve illustrates the diagnostic ability of a binary classifier by plotting the True Positive Rate (TPR) against the False Positive Rate (FPR) at various decision thresholds. By calculating the area under the curve (AUC), we could provide a direct measure of how the model's prediction ability behaves compared to complete random guessing, where its AUC score is 0.5.

Balanced Accuracy

Balanced accuracy is used for classification tasks when dealing with target data with a class imbalance. This metric was crucially implemented in our "Significant Movement" target prediction, as only 27.5% of the days in the data set experienced a significant shift. It is the average of the true positive rate and the true negative rate, given by:

$$\text{Balanced Accuracy} = \frac{1}{2}\left(\frac{TP}{TP+FN} + \frac{TN}{TN+FP}\right)$$

Learning Capacity Curve

For the SVM model, we also employ a learning cacpacity curve measure to find the best degree. This curve plot the model's training and validation performance as a function of degree. Therefore, we could find the optimized degree by diagnosing degrees that lead to underfitting and overfitting.

For the LSTM model, a learning capacity curve is also created to determine the most optimal hidden size. The curve plots the model's in-sample and out-of-sample error rates against the hidden layer size. Using this graph, we can determine the number of hidden neurons that balances model complexity and generalization capability.

Loss and Accuracy Curves

For the LSTM model, we use loss and accuracy curves graphed against epochs. These curves are important for representing model training and can help monitor learning patterns such as overfitting or underfitting. Loss curves against epochs are expected to be negative, indicating that the model is effectively minimizing the loss function. Accuracy curves are expected to be positive, but taken in the context of stochastic, noisy financial data, we expect that this curve this will oscillate.

3 Results and Discussion

3.1 Exploratory Data Analysis

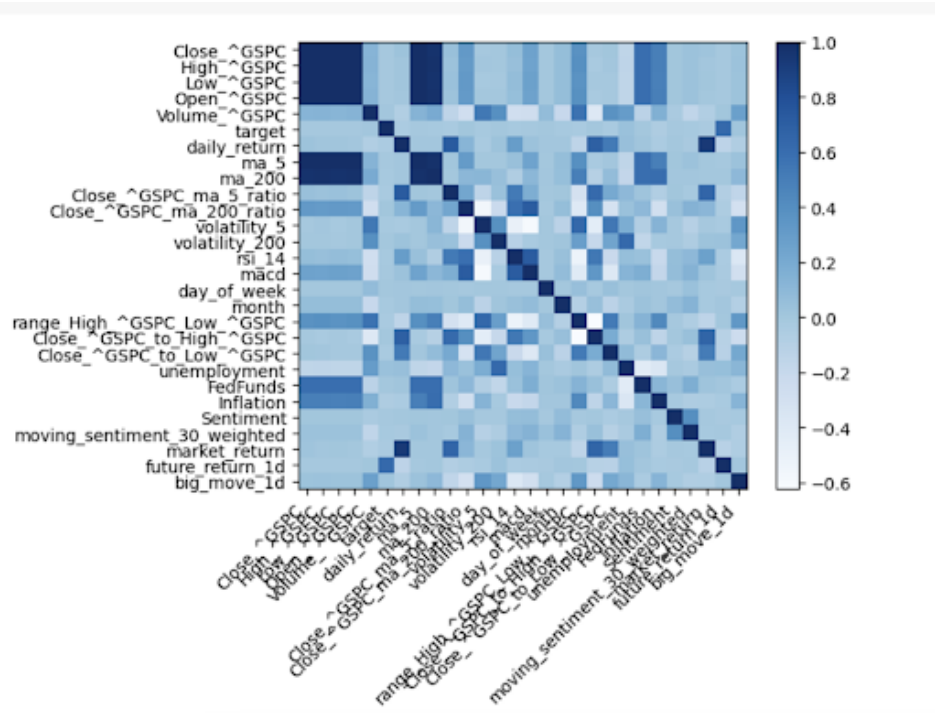

Figure 2: Correlation Matrix of Numeric Variables in the Dataset

This heatmap displays pairwise correlations between all features used in the model. Darker squares indicate stronger correlations—either positive or negative—while lighter ones show little to no relationship. Notably, technical indicators like moving averages and volatility show moderate correlation with market returns and sentiment features.

Figure 3: Time-Series Trend of Closing Prices

This plot visualizes the S&P 500's closing price trajectory from 2015 to 2025. The market shows a strong upward trend with notable drops, such as the COVID-19 crash in 2020 and other corrections in 2022. The price surge post-2023 reflects a strong recovery and possible tech-driven growth.

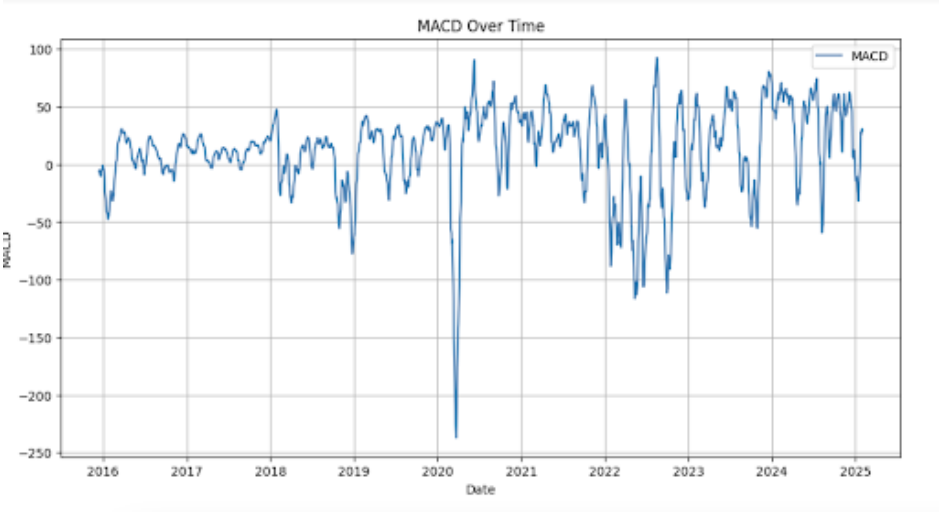

Figure 4: Time-Series Trend of Moving Average Convergence Divergence

This graph shows the Moving Average Convergence Divergence (MACD) of the S&P 500 from 2015 to 2025. Periods of strong momentum and trend reversals are visible through the oscillations and extreme spikes, especially the sharp dip in early 2020. High volatility in recent years suggests increased market instability or reaction to external events.

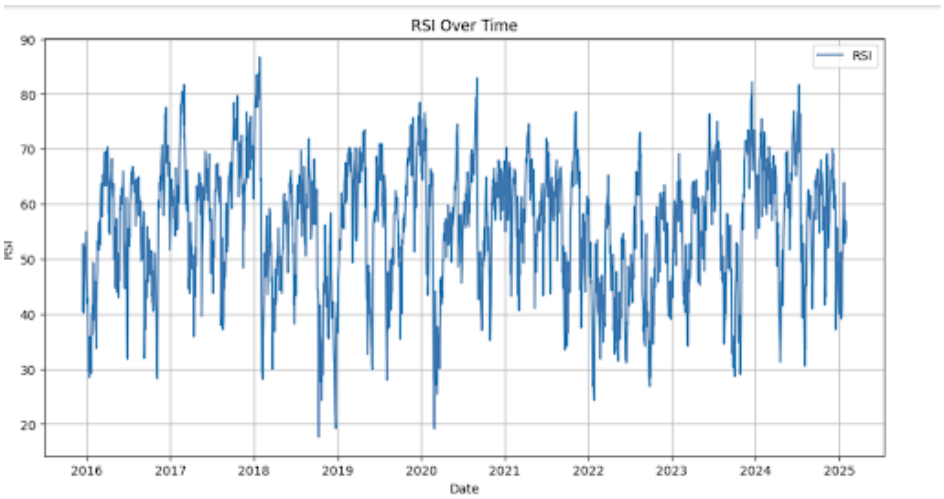

Figure 5: Time-Series Trend of Relative Strength Index

This chart tracks the 14-day Relative Strength Index (RSI), which fluctuates between overbought (great than 70) and oversold (lower than 30) levels. The consistent oscillations reflect the short-term momentum of the S&P 500, with frequent crossings of key thresholds. Peaks and dips in RSI often precede turning points in price.

3.2 Target: Significant Movement

As stated previously, our first goal was to identify future significant market (S&P 500) shifts, which we defined as either price increases of greater than 1% or price decreases of less than -1% that occur the following day. So this binary target has the value 1 for significant S&P shifts and 0 for non-significant S&P shifts. We used a variety of aforementioned machine-learning models to attempt to recognize these movements. Balanced accuracy was the primary evaluation metric in this instance. Balanced accuracy serves a primary role here because of its ability to handle the imbalanced nature of this binary classification. Approximately 27.5% of days in the dataset feature a future significant shift, so balanced accuracy accounts for this. In terms of true positive rate and true negative rate, we elected not to institute any penalties associated with either because each metric can possess meaning in certain contexts (which we will discuss later). For each classification model, we used a train/test split as well as cross-validation. Confusion matrices allow for convenient evaluation. Other visuals accompany each result.

Naive-Bayes Classifier

Note: We removed highly collinear features (such as Close and High) to preserve conditional independence. We also adjusted for the prior probability difference described by the class imbalance.

Balanced Accuracies

Train	0.70
Test	0.59

Our Bayesian model does fairly well in terms of balanced accuracy score, featuring a modest generalization gap. Although we attempted to reduce collinearity, the model may still pick up on unwanted patterns. Regardless, this phenomenon is not overly concerning.

8-Fold Cross Validation

Mean Balanced Accuracy	0.65
Standard Deviation	0.12

The cross-validation performance here reinforces the train/test split results. The mean balanced accuracy is still more than adequate, but this estimate varies substantially during the 8 folds. This indicates the inconsistency with which the model identifies market patterns.

Classification Report

- **Positive** = Significant Market Movement
- **Negative** = Non-Significant Market Movement

True Positive Rate	0.40
True Negative Rate	0.79
Positive Predictive Value	0.43
Negative Predictive Value	0.77

Confusion Matrix

1 = Significant Market Movement
0 = Non-Significant Market Movement

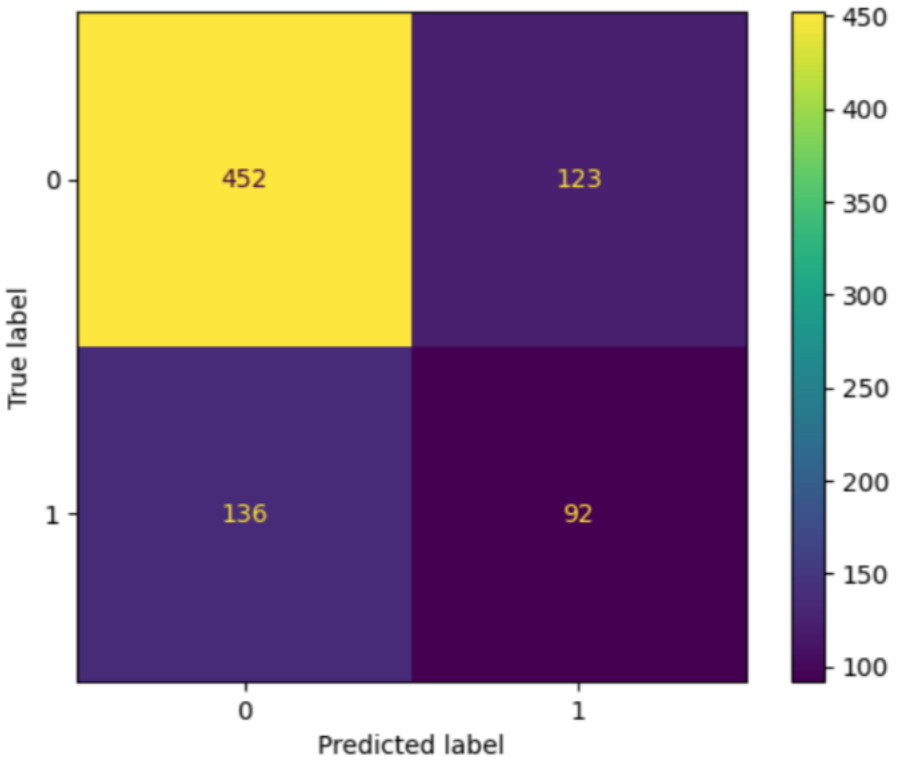

Figure 6: NB Confusion Matrix

Considering figure 6, as expected, it seems that our models will have a harder time identifying true positives (the minority class). This model identifies future non-significant market shifts with proficiency, but fails to identify significant market shifts reliably.

ROC Curve

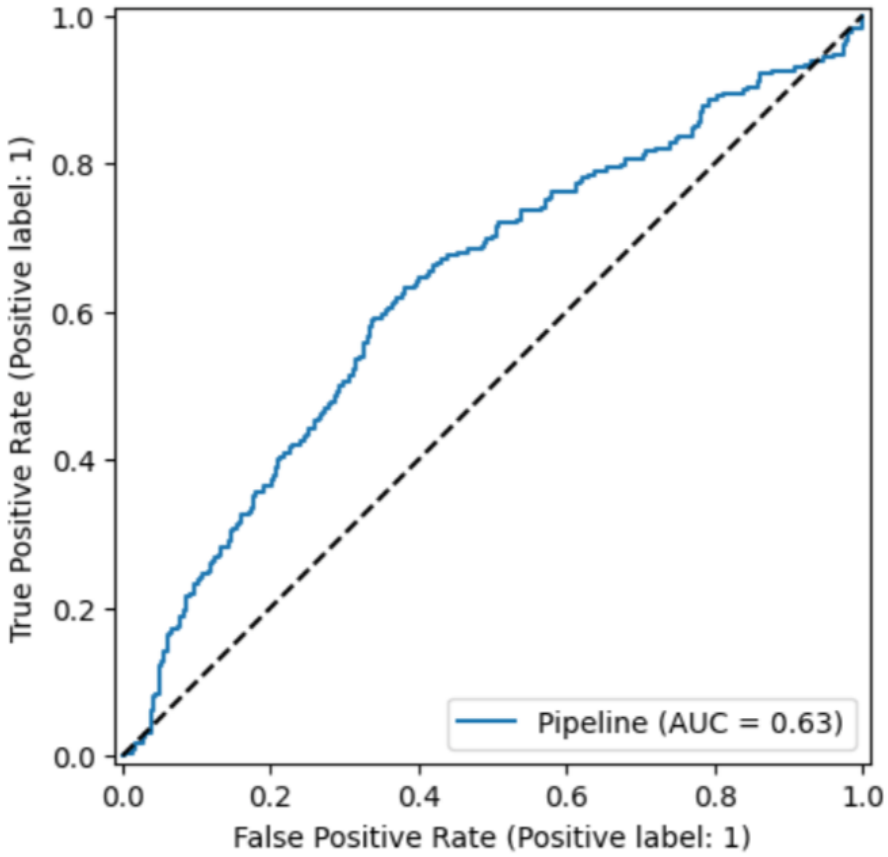

Figure 7: NB ROC Curve

Figure 7 seems to suggest that this model outperforms random guessing. All in all, this Naive-Bayes classifier serves as a solid baseline model, but it leaves plenty to be desired. It shows the potential that we can build a model that identifies real patterns, but its generalization gap and highly variable cross-validation estimates certainly should raise eyebrows.

Support Vector Classifier

For our SVC, we used a polynomial kernel with a degree of 1. This makes for a simple model that seems to generalize relatively well to unseen data. The C-value is high (1,000), but this does not seem to be at the expense of generalization (at least at 1 degree).

Learning Capacity Curve

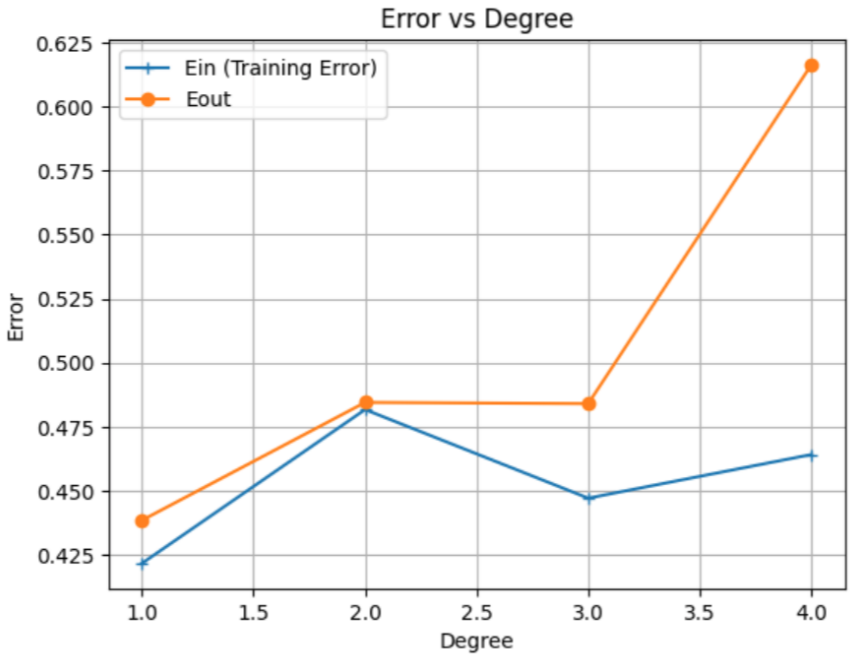

Figure 8: SVC Learning Curve

Figure 8 illustrates the generalization gap that is created by raising the degree of the polynomial. Using only one degree seems to minimize both in-sample error and out-of-sample error. As, the degree increases, the model overfits on the training data.

Balanced Accuracies

Train	0.62
Test	0.58

This SVC has a solid balanced accuracy and a comparatively small generalization gap. As we stated before, this could be a byproduct of the simplicity of the model.

8-Fold Cross Validation

Mean Balanced Accuracy	0.57
Standard Deviation	0.09

These cross-validation figures reinforce the previous sentiment. While mean balanced accuracy is lower than that of the Naive-Bayes classifier, the standard deviation of estimates is a bit lower, suggesting more simple, consistent results.

Classification Report

True Positive Rate	0.32
True Negative Rate	0.84
Positive Predictive Value	0.45
Negative Predictive Value	0.76

Confusion Matrix

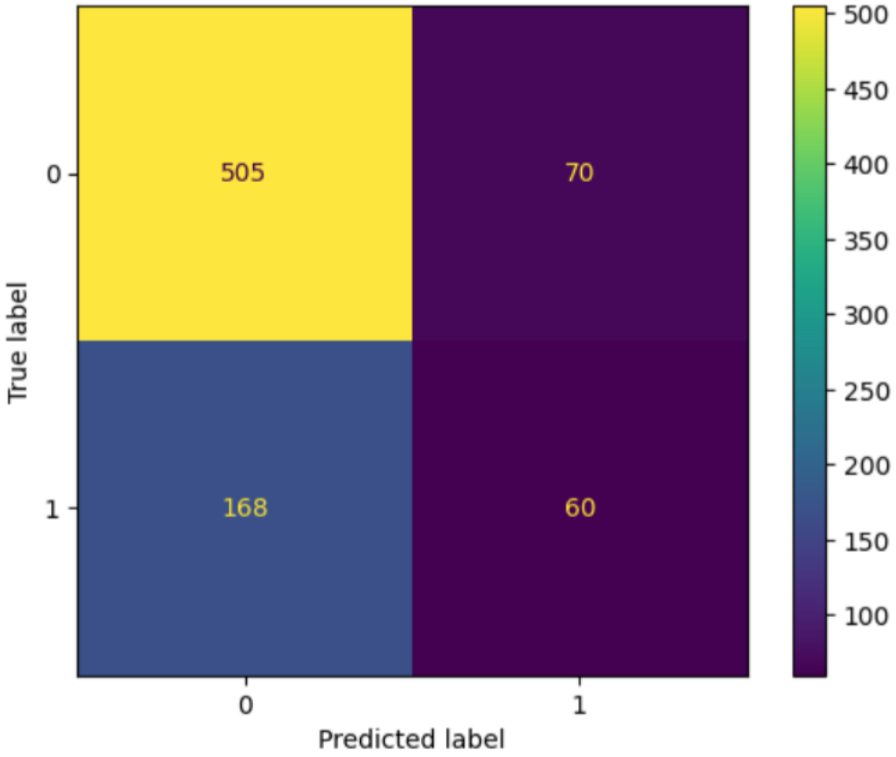

Figure 9: SVC Confusion Matrix

Figure 9 shows that this model does even worse than the Bayesian model when attempting to catch significant market shifts, but better in terms of recognizing non-significant market shifts. However, if it does identify a significant market shift, it is slightly more likely to be correct (higher PPV).

ROC Curve

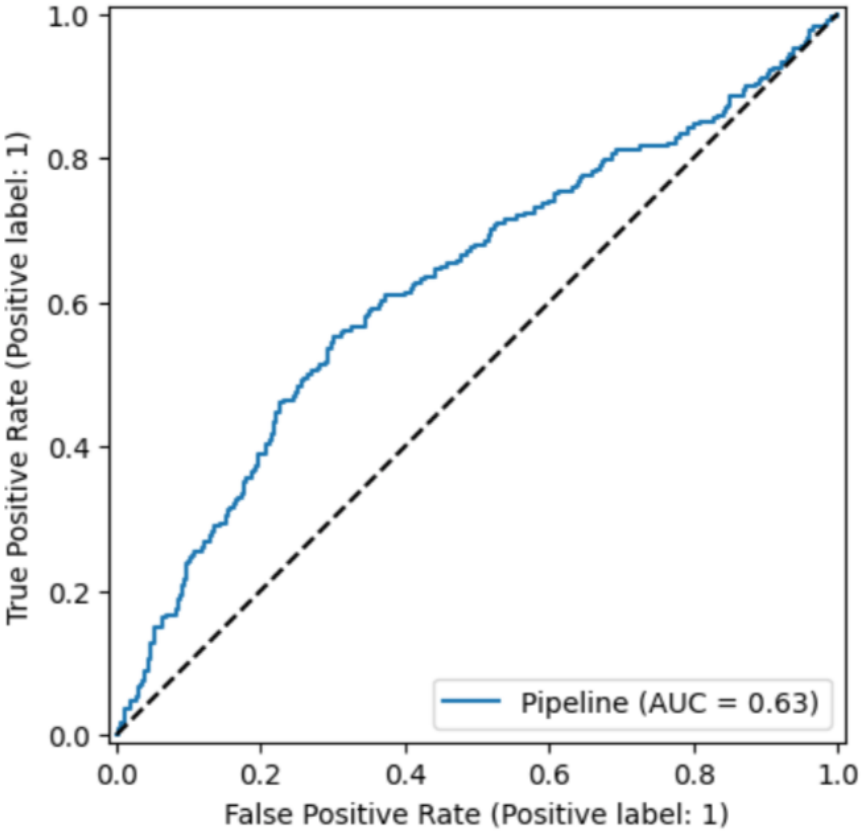

Figure 10: SVC ROC Curve

Figure 10 shows a similar AUC to the Naive-Bayes approach. The steep initial slope and curve shape indicate the recognition of some pattern, although neither of these curves really show significant pattern recognition above random guessing.

Overall, this SVC provides new value. It does relatively well in terms of generalization gap, standard deviation of estimators, and true negative rate. Its overall linear simplicity is intriguing, as it combats the notion that an overly powerful model is necessary to identify patterns in the market.

Neural Network

For this neural network, we used two hidden layers of sizes 32 and 2, respectively. The two-neuron second layer allows for active interpretation of the inner workings of this hidden section.

Balanced Accuracies

Train	0.71
Test	0.58

A relatively substantial generalization gap can be seen with this model, and its balanced accuracy on the test sample is on par with our other work.

8-Fold Cross Validation

Mean Balanced Accuracy	0.57
Standard Deviation	0.08

Despite the generalization gap seen in the train/test split, cross-validation yields the smallest standard deviation in balanced accuracy estimates.

Classification Report

True Positive Rate	0.24
True Negative Rate	0.92
Positive Predictive Value	0.53
Negative Predictive Value	0.75

Confusion Matrix

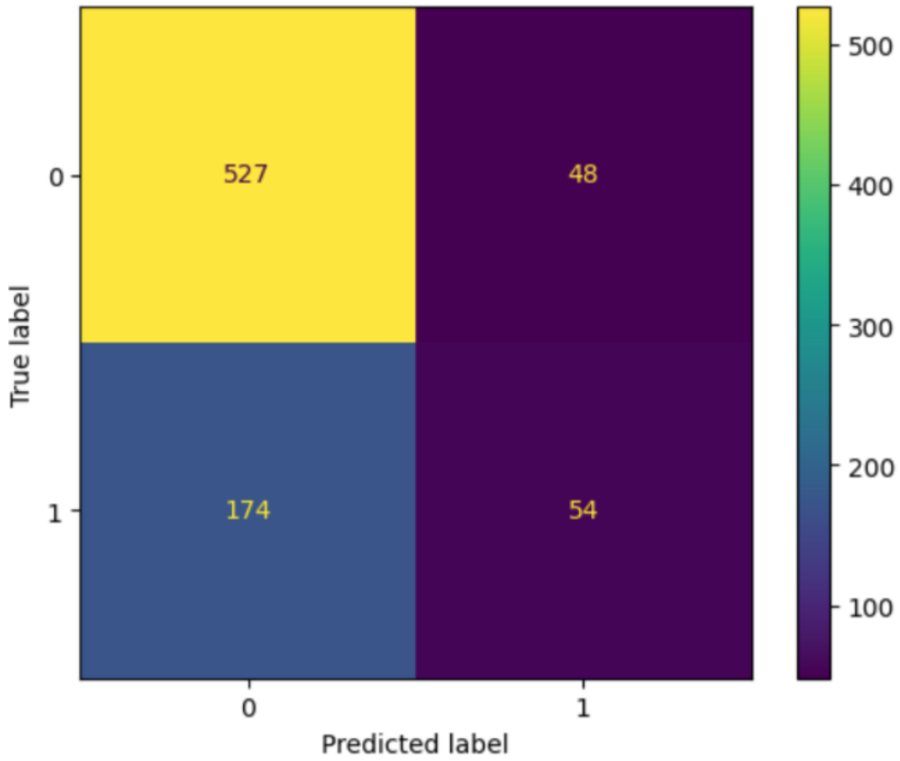

Figure 11: NN Confusion Matrix

While figure 11 shows that this model yields a low true positive rate, it is certainly worth noting that its positive predictive value exceeds 0.5. This means that if the model said that there would be a significant market movement the next day, then it actually occurred 53% of the time. In terms of true negative rate, the model does an excellent job not classifying non-significant movements as significant.

ROC Curve

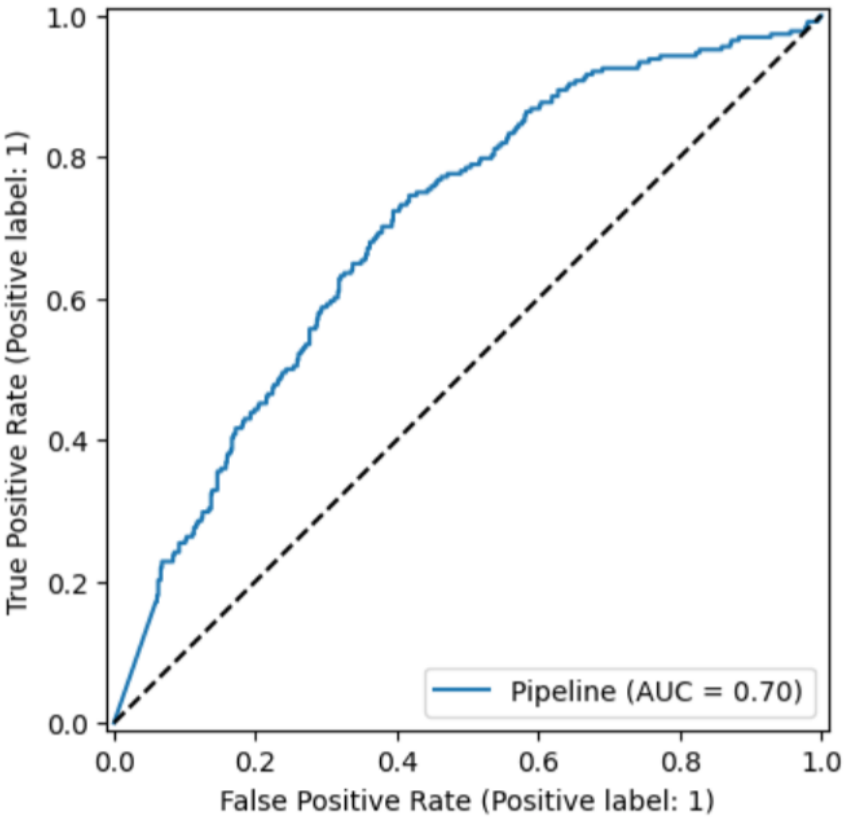

Figure 12: NN ROC Curve

Figure 12 shows that this neural network has the strongest AUC value yet, indicating that it is recognizing some sort of signal, potentially better than the other models.

Neurons of Hidden Layer

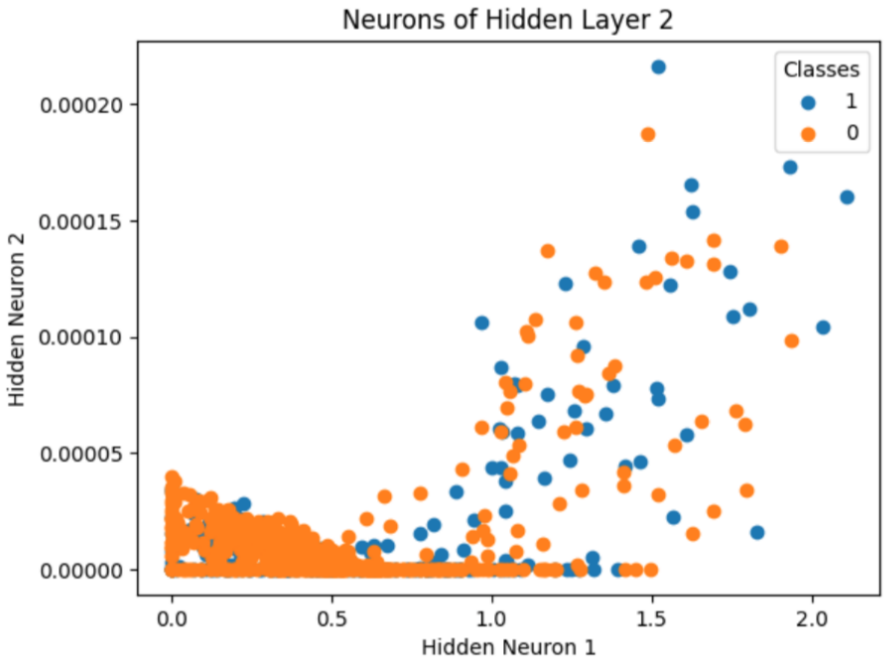

Figure 13: NN Hidden Layer

While figure 13 looks underwhelming in terms of class separation, we think it is significant that amongst the points that are not clustered around the origin, the classes are pretty balanced. This is noticeable because the classes are not inherently balanced. The fact that this cloud has a relatively even class distribution could indicate that the neural network has identified a signal, even if it looks weak. Even a weak signal is relatively strong in this scenario.

Overall Model Comparison

Each model provides a unique approach for predicting our target involving the magnitude of future S&P 500 movements. While there is uncertainty associated with the insights they provide and their respective accuracies, it seems that they are able to recognize moderate trends in our data. Given the randomness associated with the stock market, this is certainly notable.

Table 1: Model Comparison for Significant Market Movement Prediction

Model	**Train BA**	**Test BA**	**TPR**	**TNR**	**PPV**	**NPV**
Naive Bayes	0.70	0.59	0.40	0.79	0.43	0.77
Support Vector Classifier	0.62	0.58	0.32	0.84	0.45	0.76
Neural Network	0.71	0.58	0.24	0.92	0.53	0.75

The interpretability and usefulness of these models is totally subjective depending on use case. In other words, this is not a one-size-fits-all sort of solution where one model is clearly better than the field. For example, options traders who bet on big market swings would prefer to see a high true positive rate, so they may be inclined to utilize the Bayesian model. Below, we have assembled a table that outlines what type of trader would favor each metric, and as a result which model they would benefit most from.

Metric	**Trader Type**	**Model Preference**	**Why?**
True Positive Rate	Options traders	Naive-Bayes Classifier	They benefit from volatility and would want to recognize any instance of it.
True Negative Rate	Market makers	Neural Network	They benefit from stable markets where they can act as middleman.
Positive Predictive Value	Hedge funds	Neural Network	They can benefit from swings, but tend to do so in a calculated fashion.
Negative Predictive Value	Asset managers	Support Vector Classifier	They tend to be more conservative than hedge funds.

Note: Naive-Bayes Classifier generated a higher NPV than the SVC, but the uncertainty surrounding the NB would push me to recommend the SVC to more conservative firms/individuals.

3.3 Target: Up/Down

Our second objective is a binary classification task: predicting whether the market will go up or down the next day. Unlike the "Significant Movement" target, this "Up/Down" target has a more balanced class distribution. This task reflects real-world investing decisions and can inform tactical portfolio management. Due to the noisier and more marginal nature of this prediction problem, we sought out alternative approaches. Our goal is to determine whether an LSTM can learn these subtleties in financial indicators to produce reliable directional predictions.

Classical Models

While more classic models could identify certain patterns (to a degree) in terms of market movements, this was not the case when attempting to predict day-to-day direction of the S&P. Figure 14 shows the confusion matrices produced by an SVC and Figure 15 shows the neuron scatterplot of a neural network, tuned similarly to those of the previous target:

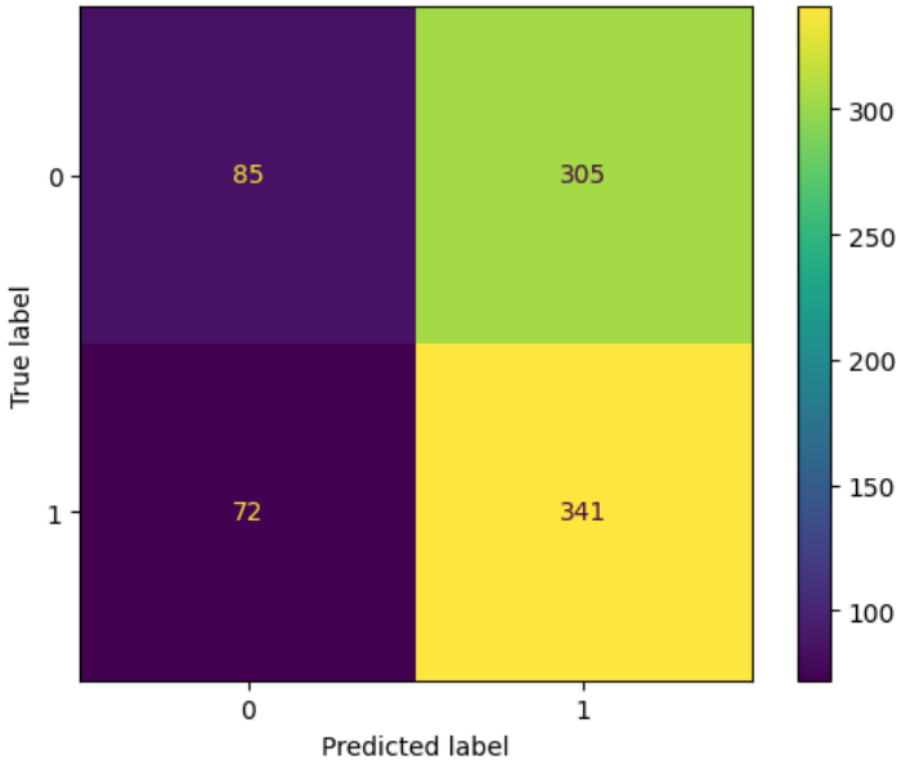

Figure 14: Up/Down Target SVC Confusion Matrix

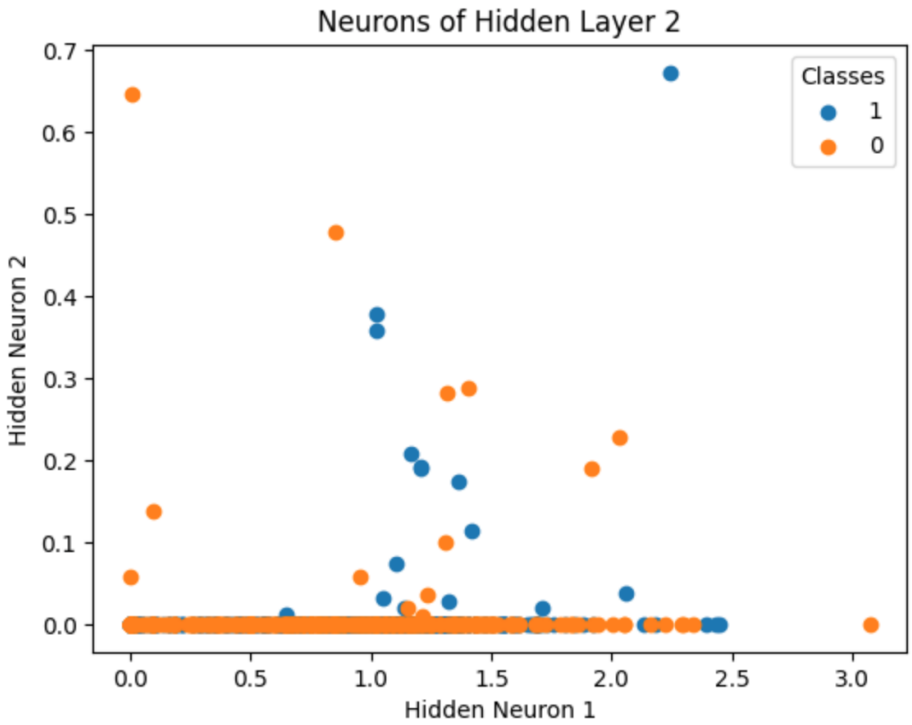

Figure 15: Up/Down Target Neural Network

Evidently, a higher-capacity modeling technique was necessary, and LSTM was the method we selected.

LSTM

LSTM training is carried out in batches of sequences, small groups of training data examples, to update the model. Each time a batch is processed, the model makes predictions on these examples and calculates the loss and the gradients. With this information, it updates the weights of the model. These steps are repeated until all batches have been processed. Smaller batches tend to have more randomness but better generalization, whereas larger batches are faster but tend to overfit. Here we used a batch size of 32.

Each time the model has gone through all of the batches of the training data is one epoch. After each epoch, the model tends to improve, so we continue to train over many epochs. At the start of each, the data is reshuffled and the batch process begins again. In this model, we used 50 epochs.

The model itself is made up of one hidden layer of size 64. To balance the trade off between in-sample and out-of-sample error, a drop out rate and regularization were used. A drop out rate of 0.4 ensures that the network is not too reliant on a given neuron. Additionally, the Ridge Penalty (L2 Norm) prevents overfitting and improves generalization.

Learning Capacity Curve

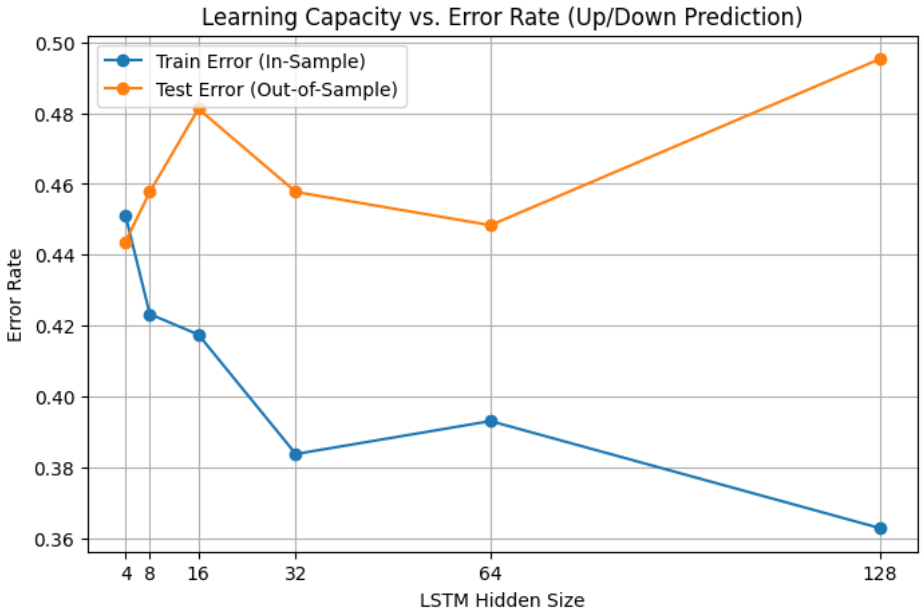

Figure 16: LSTM Learning Capacity

Figure 16 plots the in-sample error rate and out-of-sample error rate against learning capacity. Learning capacity here is defined by the size of the hidden layer in the LSTM model, 4 nodes being the most trivial model, and 128 nodes being the most complex model. As expected, the generalization gap begins to decrease as learning capacity increases. Yet, as reflected in the figure, the generalization gap increases again as learning capacity becomes too high and the model begins to overfit to the training data. Using this analysis, it can be determined that the smallest generalization gap occurs while still having a meaningfully deep model when the hidden size is 64, which supports the hyperparameter choice we made for our final model.

Accuracies

Train	0.81
Test	0.55

With regularization from the L2 Norm and a dropout rate of 0.4, the LSTM model achieved a training balanced accuracy of **81.30%** and a test balanced accuracy of **55.06%**. Despite efforts to reduce learning capacity in hopes to minimize the gap, the results suggest strong in-sample learning but limited generalization to unseen data. The confusion matrix and classification metrics on the test set are shown below.

Confusion Matrix

- True Negatives (class 0 correctly predicted): 86
- False Positives (class 0 predicted as 1): 105
- False Negatives (class 1 predicted as 0): 82
- True Positives (class 1 correctly predicted): 153

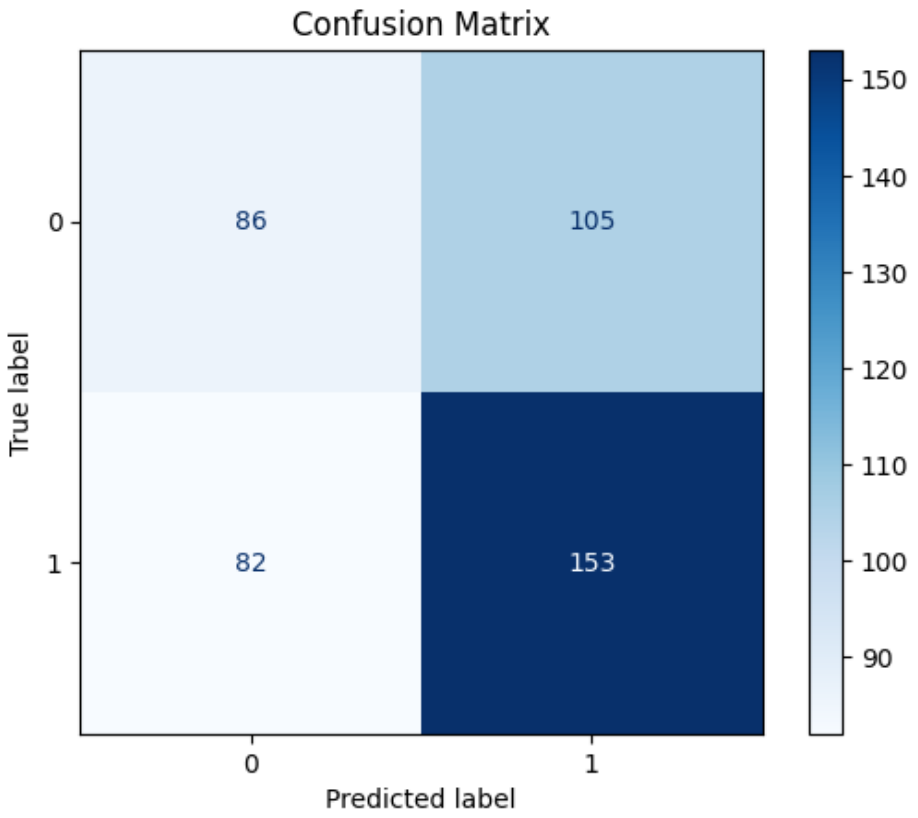

Figure 17: LSTM Confusion Matrix with L2 Regularization

Classification Report

Class	Precision	Recall	F1-Score	Support
0.0	0.51	0.45	0.48	191
1.0	0.59	0.65	0.62	235
Accuracy		0.56		426
Macro Avg	0.55	0.55	0.55	426
Weighted Avg	0.56	0.56	0.56	426

Figure 17 and the Classification Report above reflect the volatile nature of time-series financial data. Despite an analysis of in-sample error rate and out-of-sample error rate as well as regularization efforts, LSTM is only able to predict the market movement marginally better than random guessing.

Training Loss and Test Accuracy Over Epochs

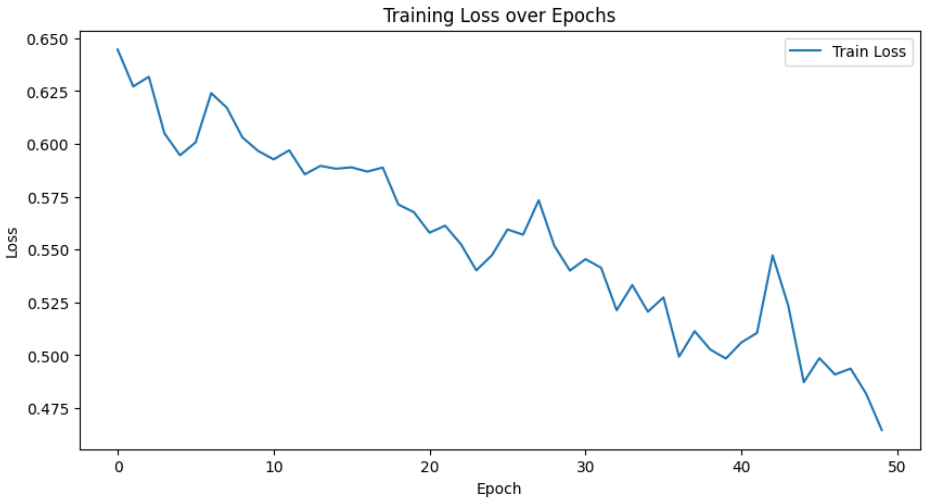

Figure 18: LSTM Training Loss

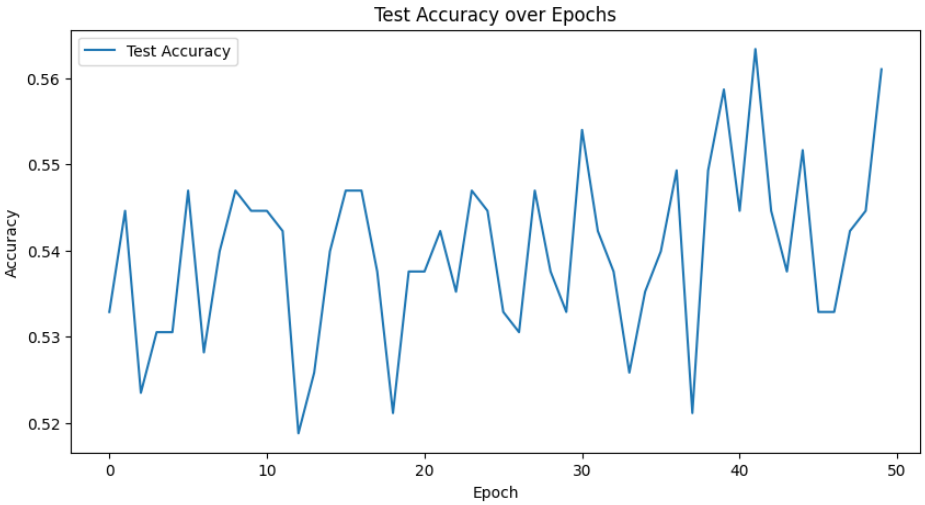

Figure 19: LSTM Test Accuracy

Figure 18 shows the training loss over 50 epochs for the LSTM model with batch size 32. The loss decreases as the epochs complete, indicating that the model is learning from the training data and supporting our choice of loss function. However, as shown in Figure 19, the test accuracy fluctuates between approximately 52% and 56%, suggesting unstable performance, given the volatile nature of financial time-series data. We can observe a subtle positive relationship, which indicates that model is learning at a marginal level.

ROC

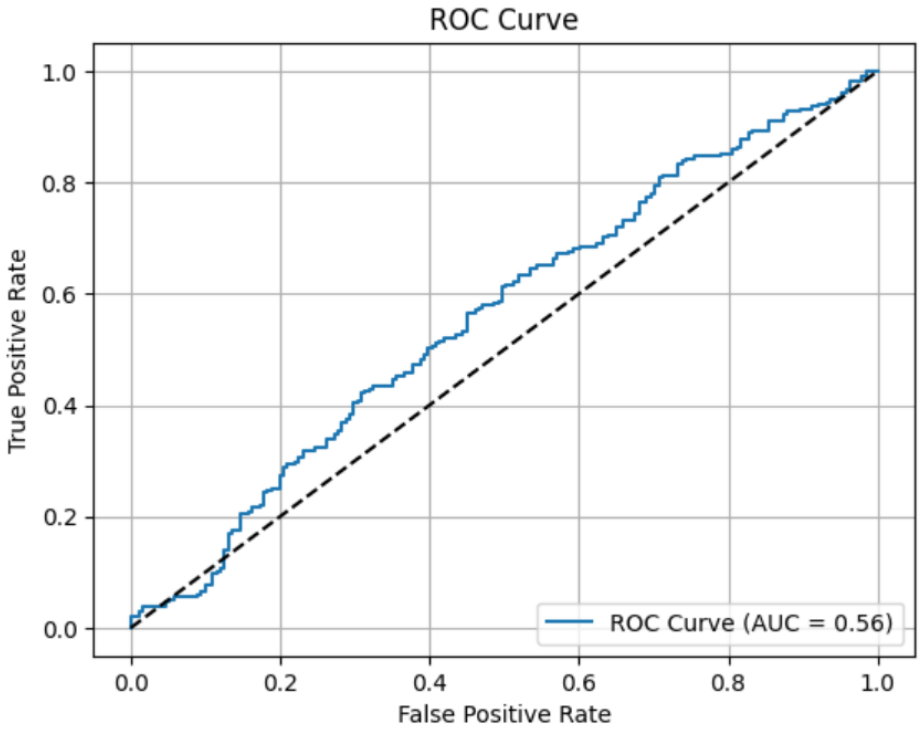

Figure 20: LSTM ROC

Figure 20 plots the ROC curve associated with the LSTM. The curve is slightly above the diagonal line, indicating that the LSTM model performs marginally better than random predictions. Despite the constraints of the stochastic nature of financial time series data, this curve indicates that the LSTM model has limited but non-trivial predictive power.

LSTM Conclusions

The use of LSTM in the Up/Down prediction task demonstrates the limited nature of direction forecasting, even given deep architectures. While our LSTM was able to learn the patterns of the training data, even with regularization using a drop-out rate of 0.4 and the ridge penalty, we found difficulty generalizing – our sample accuracy fluctuating between 52% and 56%. This generalization issue is expected given the inherent low signal-to-noise ratio in time-series financial data. Although seemingly marginal, it is important to contextualize our findings. Our results align with other research in the field whose findings reflect deep learning methods that achieve similarly modest edges over random guessing, as outlined below.

Stock	Accuracy	Precision	Recall	F1
BOVA11	0.546	0.560	0.350	0.431
BBDC4	0.559	0.553	0.129	0.209
ITUB4	0.545	0.475	0.134	0.209
CIEL3	0.530	0.476	0.137	0.213
PETR4	0.533	0.563	0.231	0.327

Figure 21: Reported Accuracies for LSTM Model Predictions from "Deep learning with long short-term memory networks for financial market predictions", Fischer & Krauss

4 Conclusion

Our original goal was to identify algorithmic signals in S&P500 daily prices, and we accomplished this. . . to a degree. This is an inherently difficult task that we approached with the best of our abilities. We utilized time developing intuitive features and tuning model parameters, and while this effort produced models that generally exceeded the accuracy of random guessing, our results still carry substantial uncertainty. For instance, the models that predicted significant market movements achieved balanced accuracies that exceeded 50%, but cross-validation shed light on the variability of these estimates. One might suggest that these models could help a trader in the long run, but one cannot conclude this given the randomness associated with our outcomes. This was an incredibly important factor in our determination of results. If we simply reported balanced accuracies that surpassed the random guess threshold, that would be a misleading representation of our work.

Our approach had a heavy focus on feature engineering, as we added sentiment scores, deliberated the best technical indicators, and sourced macroeconomic data. We did this in the hopes that some signal would be derived from their presence. This did not seem to be the case. That time may have been better spent finding optimal model parameters, potentially by using extensive grid searches. Other models may have also been useful, including but not limited to Temporal Convolutional Networks. So, we would recommend a more model-centric approach to anyone interested in extending this, as opposed to the feature-centric route we took.

Despite the obstacles associated with this topic, its complexities allowed us to apply a breadth of methods from this course to an intriguing context. This is a much-discussed area of research that has seen progress with the advent of deep neural networks, and we predict that this progress will continue in the future.

References

Bird, S., Klein, E., & Loper, E. (n.d.). *NLTK SentimentIntensityAnalyzer API Documentation.* https://www.nltk.org/api/nltk.sentiment.SentimentIntensityAnalyzer.html?highlight=sentimentintensity

Chen, K.-Y., Zhou, Y., & Dai, F.-R. (2018). LSTM-based Deep Learning Model for Stock Prediction and Predictive Optimization Model. *Preprint.* https://www.researchgate.net/publication/318329563

Chhajer, P., Shah, M., & Kshirsagar, A. (2022). The applications of artificial neural networks, support vector machines, and long–short term memory for stock market prediction. *Decision Analytics Journal, 2*, 100015-. https://doi.org/10.1016/j.dajour.2021.100015

Fischer, T., & Krauss, C. (2018). Deep learning with long short-term memory networks for financial market predictions. *European Journal of Operational Research*, 270(2), 654–669. https://www.researchgate.net/publication/318329563

Federal Reserve Bank of St. Louis. (2025). *Effective Federal Funds Rate (DFF).* Retrieved from https://fred.stlouisfed.org/series/DFF

Investopedia. (2022, April 5). *When Jerome Powell speaks, markets shudder.* https://www.investopedia.com/when-jerome-powell-speaks-markets-shudder-7488957

Levendovszky, J., Reguly, I., Olah, A., & Ceffer, A. (2019). Low Complexity Algorithmic Trading by Feedforward Neural Networks. *Computational Economics, 54(1)*, 267–279. https://doi.org/10.1007/s10614-017-9720-6

Li, Y., Zhou, D., & Xie, J. (2024). Sentiment-aware financial time series prediction using transformer models. *Humanities and Social Sciences Communications*, 11, Article 108. https://www.nature.com/articles/s41599-024-02807-x#Sec10

Lynn, B. (2016). *Predicting the stock market using a LSTM neural network* (Master's thesis, Lund University). Retrieved from https://lup.lub.lu.se/luur/download?func=downloadFile&recordOId=9137039&fileOId=9137040

Sak, H. (2017). *Understanding LSTM and its diagrams.* ML Review. Retrieved from https://blog.mlreview.com/understanding-lstm-and-its-diagrams-37e2f46f1714

Stanford University. (n.d.). *CS224n Final Project Report: Predicting Stock Price Movement using News Sentiment and LSTM.* Retrieved from https://web.stanford.edu/class/archive/cs/cs224n/cs224n.1234/final-reports/final-report-170049613.pdf

TA-Lib. (2024). *Momentum indicators — Technical Analysis Library in Python.* https://technical-analysis-library-in-python.readthedocs.io/en/latest/ta.html#momentum-indicators

TA-Lib. (2024). *Trend indicators — Technical Analysis Library in Python.* https://technical-analysis-library-in-python.readthedocs.io/en/latest/ta.html#trend-indicators

U.S. Inflation Calculator. (2025). *Current Inflation Rates: 2000-2025.* Retrieved from https://www.usinflationcalculator.com/inflation/current-inflation-rates/

Yang, J., Li, Y., Chen, X., Cao, J., & Jiang, K. (2019). *Deep Learning for Stock Selection Based on High Frequency Price-Volume Data.* https://arxiv.org/pdf/1911.02502

Python Libraries and Functions

- **yfinance** — Access to Yahoo! Finance's API.
- **pandas**
  - `.shift()` — Access forward rows (used in target feature engineering).
  - `.rolling()` — Rolling window calculations for volatility and moving averages.
- **numpy** — Numerical operations and array manipulations.
- **matplotlib.pyplot** — Data visualization.
- **pandas_datareader** — Used to scrape unemployment data.
- **google.colab** — Used to access CSV files for macroeconomic data in Google Colab.
- **ta** — Technical analysis library.
  - `RSIIndicator` — RSI calculation.
  - `MACD` — MACD calculation.
- **nltk** — Natural language processing.
  - `SentimentIntensityAnalyzer`, `polarity_scores()` — Sentiment analysis tools.
- **time**
  - `sleep()` — Used to avoid API request rate limits.
- **requests**
  - `get()` — Used to make API requests.
  - `status_code` — Used to check the status of an API request.
  - `json()` — Retrieve API responses in JSON format.
- **csv**
  - `writer()` — Used to write API-retrieved data into a CSV file.
- **sklearn**
  - `metrics`: `balanced_accuracy_score`, `classification_report`, `confusion_matrix`, `ConfusionMatrixDisplay`, `roc_curve`, `auc`, `RocDisplay`
  - `model_selection`: `train_test_split`, `cross_val_score`
  - `preprocessing`: `StandardScaler`, `OneHotEncoder`
  - `compose`: `ColumnTransformer`
  - `neural_network`: `MLPClassifier`
  - `svm`: `SVC`
  - `naive_bayes`: `GaussianNB`
  - `pipeline`: `Pipeline`, `FeatureUnion`
- **PyTorch**
  - `torch.tensor` — Tensor object.
  - `torch.optim.Adam` — Optimizer.
  - `torch.no_grad` — Context manager to disable gradient calculation.
  - `torch.nn.Tanh`, `Sigmoid`, `Dropout`, `LSTM`, `BCELoss`, `Module`, `Linear (fc)` — NN components from `torch.nn`.