Using Classification Models to Predict S&P 500 Monthly Returns

Raphael Ravinet, Jeffrey Giantonio, Kai Penfold

December 2023

Contents

1 Introduction **2**

2 Variables **2**

3 Data **4**
3.1 Data Collection 4
3.2 Cleaning & Feature Engineering 4

4 Methodology **6**
4.1 Model Selection 6
4.2 Cross Validation 7
4.3 Hyperparameter Optimization 8
4.4 Within and Out of Sample Model Evaluation 9

5 Conclusions and Considerations for Further Research **11**

6 Appendix **12**

7 References **15**

1 Introduction

The accurate prediction of stock returns has long been a goal in finance, driven by its critical role in shaping investment strategies. Investors seek to forecast market trends to optimise their investment strategies, mitigate risk, and ultimately maximise gains. This predictive problem gains urgency in a world where financial markets are increasingly complex and volatile. In recent years, with vast amounts of readily-available data this problem has evolved, seeing research methodologies shift away from traditional statistical approaches like linear regression, to more advanced and sophisticated machine learning models. These models have proven to be adept at analysing large amounts of data and uncovering complex, non-linear patterns, placing machine learning at the forefront of financial forecasting problems.

In this report, we explore an alternative approach to simplify the problem of predicting stock market returns. We employ binary classification algorithms to predict the direction of the S&P 500 based on historical data. This approach stems from the recognition that predicting exact returns is challenging, thus we focus on classifying outcomes into distinct ranges in an attempt to gauge overall market trends more effectively.

Figure 1: SPY Monthly Returns Over Time (1995-2023)

2 Variables

Target variables: To capture our target variable, we used the SPDR S&P 500 ETF (ticker name: SPY), measuring monthly returns as the log difference in closing price at the end of each month. In taking a classification approach, we binned the outcomes of these differences into two classes: **High returns:** greater than +3% month-over-month returns. **Not high returns:** less than or equal to +3% returns (includes both negative and low positive returns).

We decided to set the threshold higher than just positive or negative for two reasons. The first and main reason was driven by model considerations: Most monthly returns are slightly positive or slightly negative, and the difference between these months around zero was likely to be muddled and therefore difficult for our simple model to discriminate between. By setting the bar higher, we hoped to better differentiate the classes. The second consideration was that in practice, investment decisions usually involve both transaction costs and opportunity costs, so by setting the bar higher we build a model which could help inform trades actually worth making. As illustrated in Figure 2 below, this introduces some class imbalance, but not too much to demand using class weighting techniques.

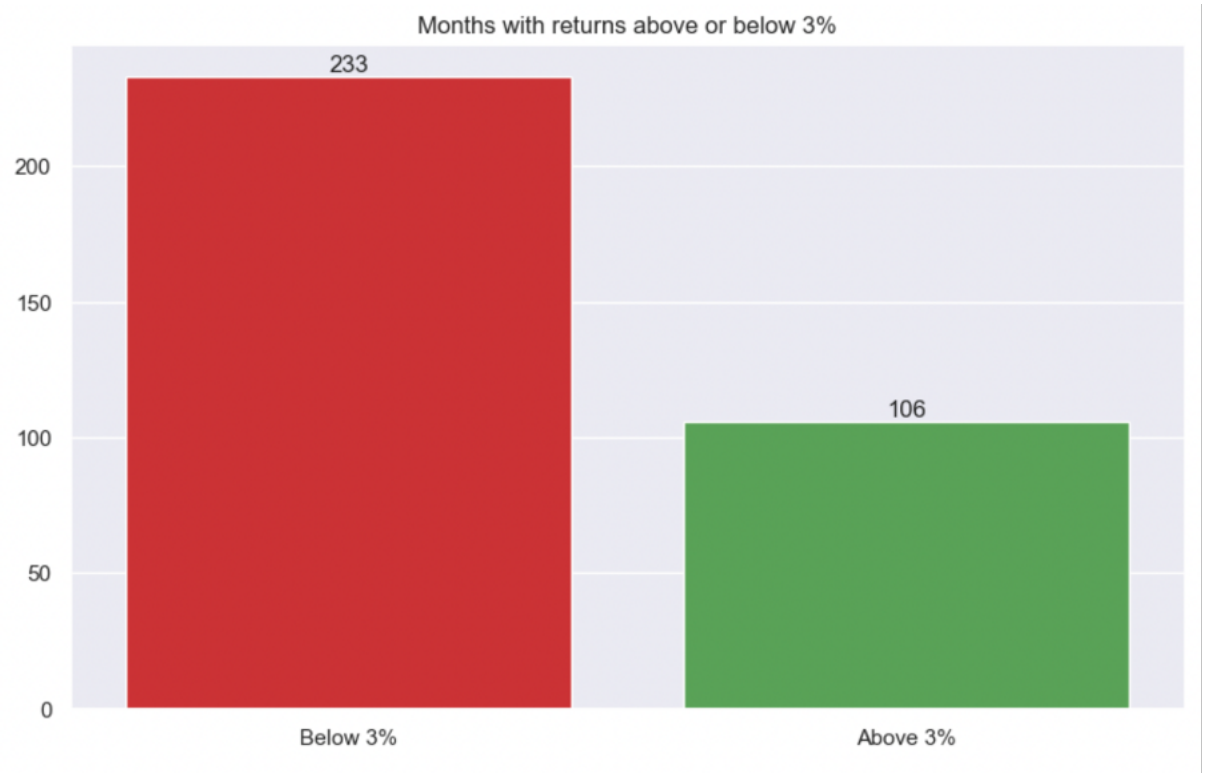


Figure 2: Number of months with returns above or below 3%

Predictors: To predict SPY, we collected macroeconomic and financial indicators over the period from January 1995 to April 2023. Because the performance of the companies listed in the S&P 500 depends on the economy in which they operate, it is plausible that changes in indicators of economic conditions might anticipate changes in the performance of those companies or changes in investor confidence in the market more broadly. Financial indicators may also reflect investor attitudes and credit conditions which affect investment decisions down the line. The full list of variables can be found in the appendix. A quick look at the graphs in Figure 3 suggest that the distribution of predictors in our training set isn't dramatically different between each class, but there are small visible differences in Industrial Production and the Bloomberg Commodity Index which our model may be able to exploit.

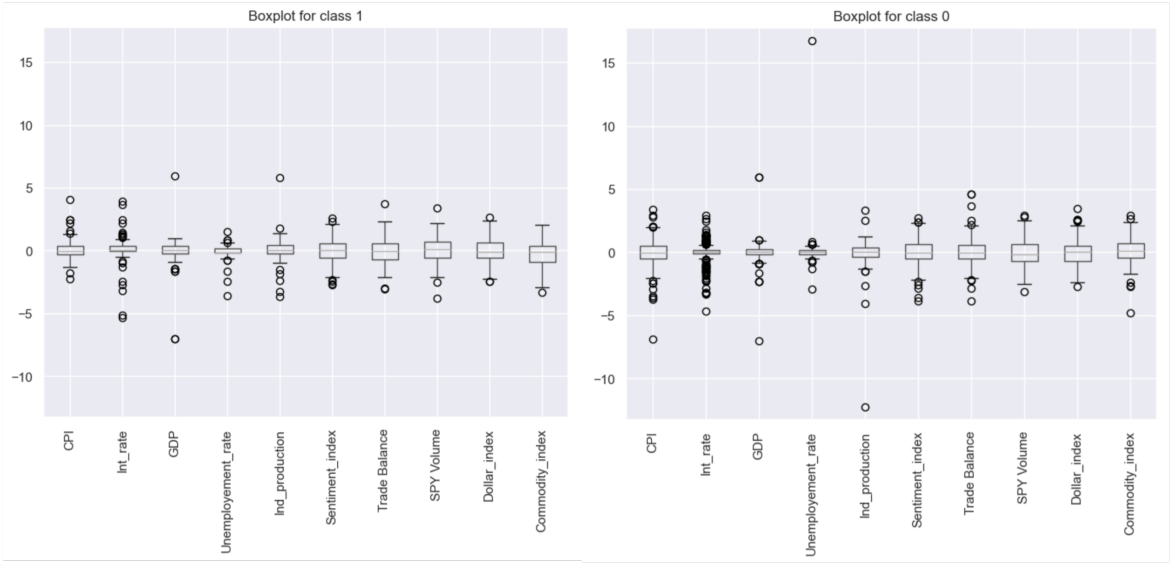


Figure 3: Distribution of predictor values by target class (training sample)

3 Data

3.1 Data Collection

Our full dataset consists of 339 monthly observations. We chose this period to maximise the sample size within two constraints: 1) the relevance of the historical period for learning about the current period and 2) the availability of historical indicators, some of which began in the 1990s. It is worth noting here that this period includes some potential outliers (e.g., dot-com bubble, 2008 financial crisis, COVID-19 pandemic), which we decided not to remove due to our small sample size. After preprocessing our data, we set aside the last 40 months (December 2019 - April 2023) for testing, leaving 299 months (February 1995 - November 2019) for training.

We extracted our macroeconomic variables from FRED-MD, the Federal Reserve's monthly database of macroeconomic variables, using an API key. Most of our financial variables were similarly extracted from Yahoo Finance, while the remainder were extracted manually from Guru Focus.

3.2 Cleaning & Feature Engineering

Data format: For ease of processing, the imported data was organised into 3 different dictionaries, based on their sources (FRED, Yahoo Finance and GuruFocus). As these data sets shared the same format and structure, we were able to apply the following procedures to each of them before combining them: (1) removing text headers, (2) converting the 'Dates' variables to ISO 8601 format (YYYY-MM-DD), (3) renaming column labels for better readability.

Granularity: The data we extract from Guru Focus was daily financial data. As we are dealing with monthly data, we used the resample method from pandas, to aggregate

these daily data points into monthly data based on its mean.

Data structure: After this initial cleaning process we combined all datasets into a unique dataframe, using the 'concat' method from the pandas library. This allowed us to join our data frames' columns side by side, aligning on their dates. In the combined dataframe, we had a total of 48 columns. Most of them were not relevant to our objective (e.g 'realtime_start'), so we dropped these columns and we were left with 12 columns.

Lags: Because this is a prediction problem, we want to use information available this month to predict outcomes next month. Thus, we needed to shift all of our predictor variables back one period to structurally align this month's predictors with next month's outcomes. For our financial data this was sufficient as this data is available immediately or almost immediately for the periods they describe. For our macroeconomic variables though, we had to shift each observation back one additional period to account for the several-week delays in government reporting and thus to avoid look-ahead bias.

Missing values: We had 263 NaN values, 244 belonging to the GDP variable because this data is reported quarterly instead of monthly. The remaining missing values refer to the first or last rows, corresponding to prediction lags. As we will filter our dates later on to the exact period in our analysis, this won't be a problem. To deal with NaN values in the GDP column, we forward-filled from the most recent available GDP data point. In economic analysis, these interim values are often imputed using methods like cubic spline, but for our purposes we are less interested in the true relationship between true variables and more interested in what we can predict using information as it becomes available. More practically, the cubic spline values would not be valid as a way to estimate the latest interim GDP figures in real time.

```
Number of NaN Values per Column:
cpi                                                    1
int_rate                                               1
gdp                                                  244
unemployement                                          1
indpro                                                 1
consumer_sentiment                                     1
trade                                                  1
spx                                                    1
Volume                                                 1
Dividends                                              1
vix                                                    3
dollar_index                                           1
S&P 500 Earnings Yield with Forward Estimate           1
Shiller PE Ratio for the S&P 500                       2
S&P 500 PE Ratio with Forward Estimate                 1
S&P 500 Dividend Yield                                 1
BofA Merrill Lynch U.S. High Yield Total Return Index  1
```

Figure 4: Number of Missing NaN Values per Variable

Additional datasets: We drew from two more datasets with very different structures. We followed the same steps as above (removed text headers, converted dates column, renamed column labels) before joining them into the combined dataset, but we also had to deal with specific cases in each of these datasets: ACWI - Data were indexed by the last or second to last day of each month. In order to match our other datasets (which

were indexed by the first day of each month), we "pushed" our data points one or two days to the future so they would be recorded as the first day of the next month. By doing so, we avoid data leakage. Commodity Index - The date was in a different format (DD/MM/YYYY), so we created a function to convert it to ISO 8601 format.

Created variables: We also created a few new variables from our existing variables. First we forward-lagged each of our macroeconomic variables 1-5 periods so that our model could capture delayed effects up to five months in the future. Second, we created variables to capture longer term trends (1 year or more) in our variables (e.g. annual CPI), including our target variable, measuring the exponential moving average of SPY for the previous 12 or 24 months.

Transformations: Before scaling all of our variables using StandardScaler, we took the first difference of all percentage variables and the percentage change of all persistent non-percentage variables so that all variables would measure change, rather than absolute level. For our target variable SPY, we took the log difference, which is standard practice in financial analysis and allows differences to be interpreted similarly to percentages.

4 Methodology

4.1 Model Selection

As our project aimed at binary classification of stock market returns, we opted for Logistic Regression as our baseline model due to its simplicity and high interpretability. It serves as an essential benchmark, offering a clear comparison point for the performance of more complex models that we plan to employ later.

Despite its advantages, Logistic Regression has certain limitations, particularly when dealing with data that exhibits complex and non-linear patterns. This limitation stems from its inherent linearity assumption. Additionally, Logistic Regression's sensitivity to outliers poses another challenge for our analysis. In our dataset, significant market events like the global financial crisis and the COVID-19 pandemic are likely to introduce outliers which have the potential to skew the Logistic Regression's analysis, leading to less reliable predictions.

To address these limitations we turned to more advanced machine learning models: Random Forest and Gradient Boosting. Random Forest offers a more robust framework and it is known for its ability to capture non-linear relationships in the data. By building multiple decision trees and averaging their outcomes, Random Forest is able to provide a view of the data while minimising the risk of overfitting that is often associated with single decision tree models.

Gradient Boosting, on the other hand, builds models in a sequential manner where new models attempt to correct the errors made by the previous ones. It is often used when dealing with complex and non-linear datasets, and if fine-tuned, it can lead to effective predictions.

4.2 Cross Validation

In order to estimate and compare our model performance in unseen data we applied cross validation methodology. Standard methods such as K-Fold cross-validation and Leave-one-out cross validation involve splitting the dataset into several different folds, using some of these to train and some to test, then averaging performance scores. Whilst these methods are popular and relatively straightforward to implement, they are not suitable for time series analysis as they fail to respect temporal dependence among data points. Instead, these methods assume independent and identical distribution among observations, which is seldom the case with financial time series data. Therefore, implementing something like K-Fold cross-validation in our context may cause data leakage by unintentionally using data from future training periods that have not yet been seen in the current fold's test set. This can have significant impacts on our models' performance, so navigating this was crucial.

Instead, we used the TimeSeries Split function from the scikit-learn library. This method allowed us to partition our data into five sequential folds, thus preserving the temporal integrity of our observations. Here our training sets remain sequential and increase in size with each fold, gradually including all previous data up to that point, so our models are always trained on available historical data. Using TimeSeriesSplit also ensures that our test sets are non-overlapping, nullifying any concerns for data leakage and ensuring that the model's predictive ability is tested only on data it hasn't seen during training.

This is illustrated in Figure 5 below, whereby our 299 training observations are split across 5 different folds, each beginning where the previous fold ended, ensuring no overlap and thus preserving temporal integrity.

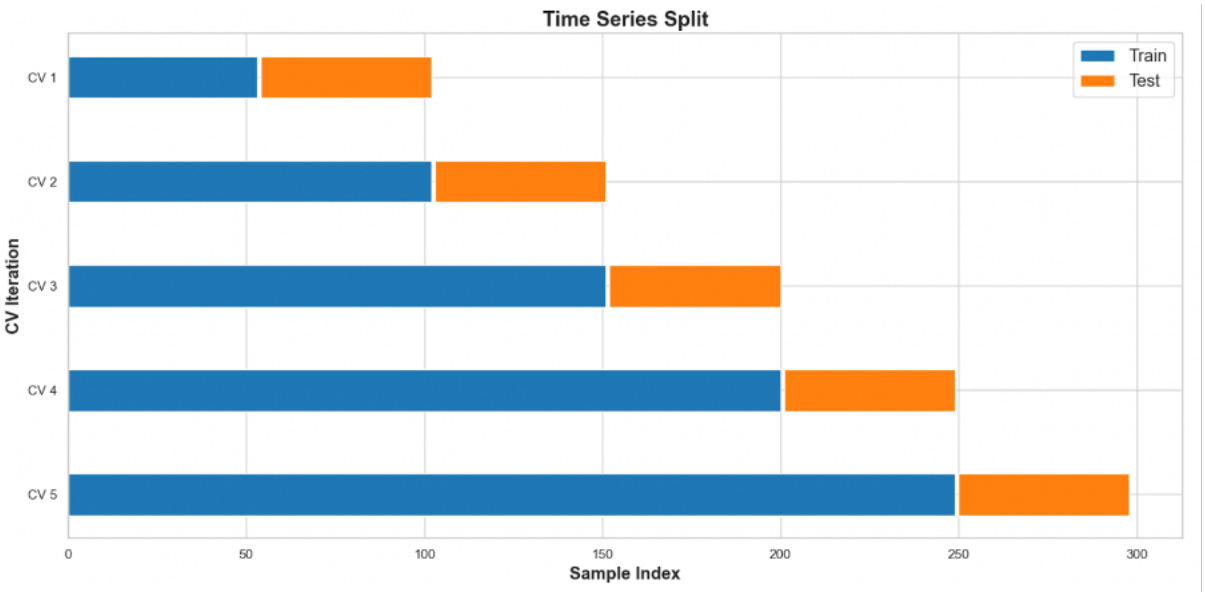


Figure 5: Time Series Split Across Data Set

4.3 Hyperparameter Optimization

Hyperparameter optimization plays a crucial role in the refinement of machine learning models, especially when dealing with the complexity of financial time series data. These parameters have a great influence in each model's performance by dictating how the algorithm learns from the data, and thus require careful calibration.

Models like Random Forest can often perform well using default settings and as such might require less hyperparameter tuning. This is due to the robustness of its ensemble learning technique and built-in cross-validation through bootstrap aggregating (bagging), both of which work to yield solid results despite less tuning. On the other hand, alternative models such as Gradient Boosting have a more extensive range of hyperparameters that can have big impacts on performance, and so tuning these correctly should be carefully considered. This sensitivity to hyperparameter settings comes inherently from the sequential learning process used in these models. Unlike Random Forest, each tree in Gradient Boosting learns to correct the mistakes made by its predecessor, meaning every tree directly influences the final result – therefore, parameters controlling both size and depth of trees need to be optimal. Another example of this is the learning rate used in Gradient Boosting models. This aims to control how strongly each tree corrects the mistakes of those before it, yet if set too high, the model may converge to suboptimal results. If set too low, the model may require an impractical number of trees to reach any meaningful outcome. This drives the motivation for us to consider a thorough yet efficient hyperparameter optimisation process.

We originally considered methods such as grid search and random search to approach our problem. These methods exhaustively evaluate all possible combinations of parameters, and while they excel when testing fewer hyperparameters in less complex models, they can suffer from two key limitations when working in a higher dimension of hyperparameters:

1. **Lack of scalability:** The exponential expansion of grid points to search when increasing the number of hyperparameters becomes computationally infeasible.

2. **Difficulty in locating global optima:** Equal importance is placed on each hyperparameter, meaning that the model does not learn from poor combinations and can potentially overlook more promising areas of the search space.

Based on these limitations and given our resource constraints for this project, we opted for an alternative and more feasible approach – Optuna. Optuna is an advanced hyperparameter optimization framework that employs a more informed search strategy. It applies Bayesian optimization with a tree-structured Parzen Estimator (TPE), which iteratively refines its search based on past trial outcomes, focusing on the most promising regions of the hyperparameter space. This method not only accelerates the search process but also enhances the likelihood of uncovering the global optimum within a vast and complex parameter space. This search process is highlighted in Figure 6 below.

For each of our tree-based models, we defined a set of hyperparameters that we were interested in optimising alongside a range of values. We then defined a function aimed at maximising each model's AUC score.

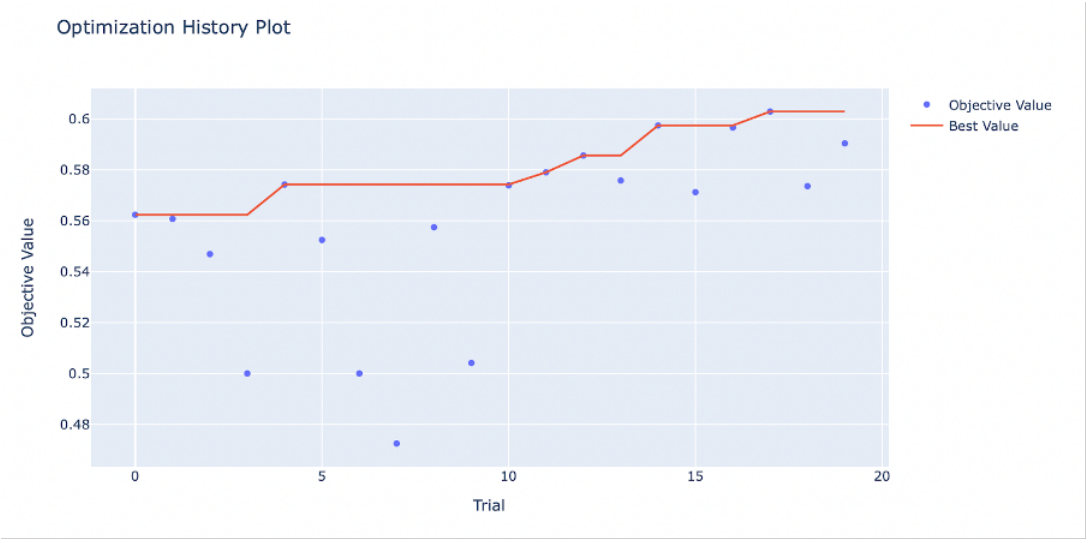


Figure 6: Optuna Optimization History by Each Trial

The hyperparameter tuning results can be seen in Figure 7 below for the Gradient Boosting model. Here, we identify three key hyperparameters: max_features, max_depth and min_samples_split. Together, they indicate that the number of features considered at each split, the depth of each tree, and the minimum number of data points required to split a node were integral in configuring the model. Intuitively, this makes sense – these settings play a key role in balancing the variance-bias trade off.

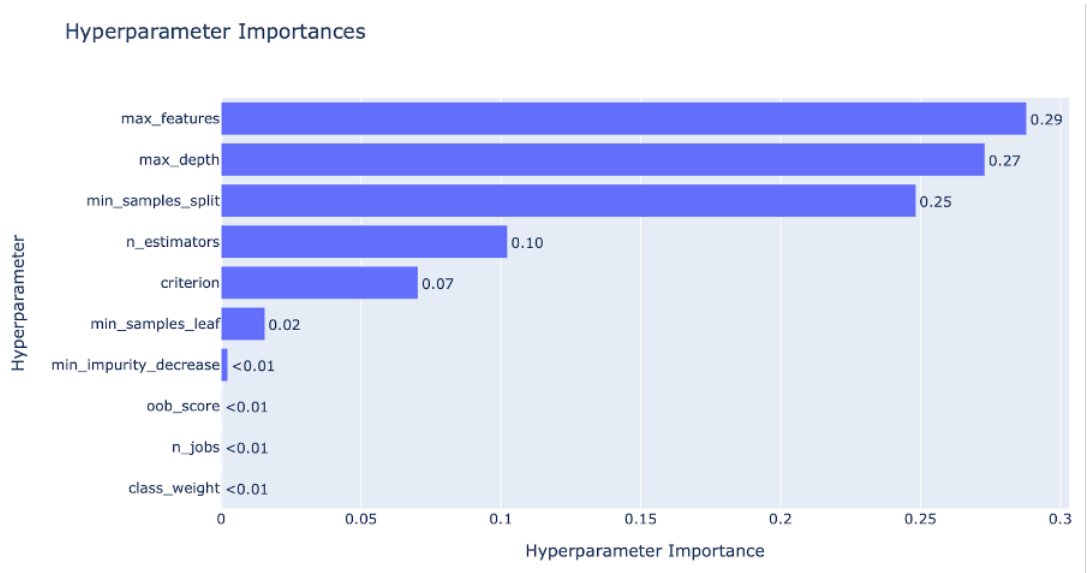


Figure 7: Hyperparameter Importance for Gradient Boosting Model

4.4 Within and Out of Sample Model Evaluation

Both within sample and out of sample, our models have weak AUC scores but are still better than a random guess, which in the context of finance is decent. Running our model on the 40 months we set aside for our test sample, we see that all three models performed slightly better with respect to AUC compared to in-sample and slightly worse

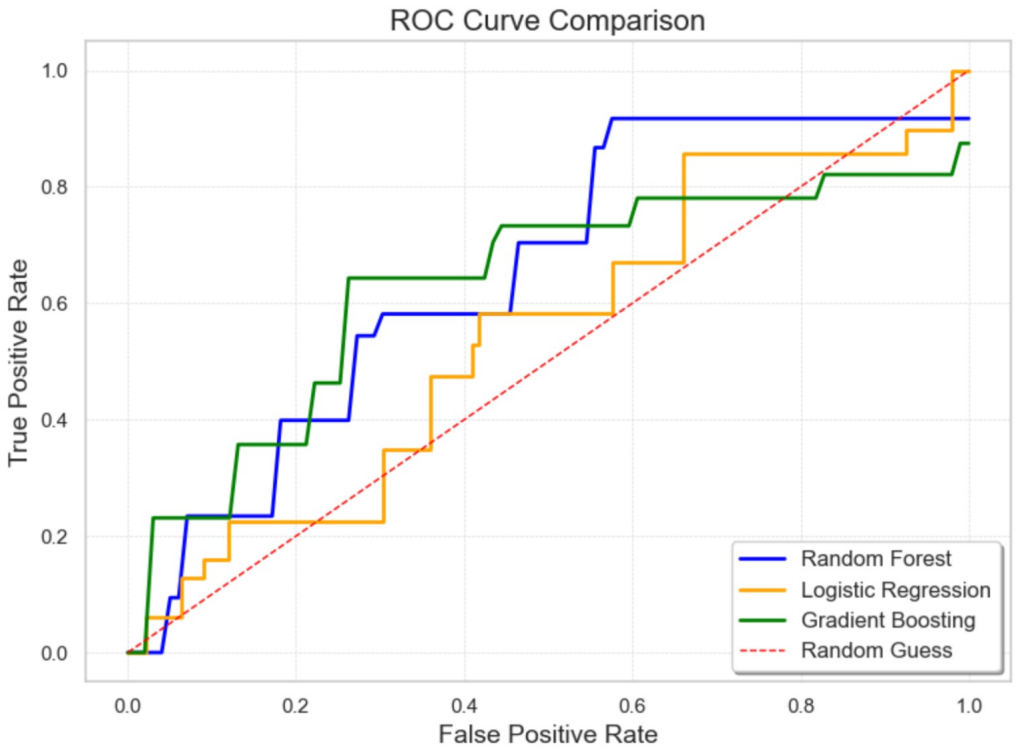


Figure 8: Out of Sample Model ROC Curves

with respect to accuracy. Out of sample, Gradient Boost now edges out both of the other models on both AUC and accuracy. Moreover, Gradient Boost edges out Logistic Regression in both precision and sensitivity. Random Forest, meanwhile, is entirely insensitive to positive cases and produces zero high-return months, either correctly or incorrectly.

Within Sample: Threshold = 0.5			
Score	Logistic	Random Forest	Gradient Boost
AUC	0.519	0.592	0.560
Average Accuracy	57.60	71.02	61.22
Precision	0.281	0.320	0.268
Sensitivity	0.435	0.129	0.306

Out of Sample: Threshold = 0.5			
Score	Logistic	Random Forest	Gradient Boost
AUC	0.596	0.635	0.638
Average Accuracy	55.00	60.00	65.00
Precision	0.333	0	0.667
Sensitivity	0.125	0	0.250

Table 1: Within & Out of Sample Model Results

Despite the weak predictive power of all of our models, we take the consistency of these results with our in-sample results as a positive sign that we have successfully minimised the variance of our models, especially because this test period (December 2019 - April 2023) could be considered highly unusual, spanning the COVID-19 pandemic, a return to high inflation, and the war in Ukraine. Because we were interested in seeing whether Random Forest could be induced to make useful predictions at a lower level of confidence, we lowered the decision boundary for all three models from a 50% probability cutoff to 40%. As is evident in the confusion matrices below, lowering the probability cutoff made Random Forest the most sensitive model but also the most imprecise. Despite both also increasing in imprecision, Gradient Boost remained the most precise model, and

also remained more sensitive than Log Regression. Whether 40% (or even 50%) is an acceptable confidence level would ultimately depend on the specific risk preference of the investor using this model. However, Gradient Boost appears to be the best model at either threshold.

Out of Sample: Threshold = 0.4			
Score	Logistic	Random Forest	Gradient Boost
AUC	0.596	0.635	0.638
Average Accuracy	62.50	55.00	65.00
Precision	0.556	0.45	0.583
Sensitivity	0.313	0.563	0.438

Table 2: Out of Sample Model Results With Lower Probability Cutoff Threshold

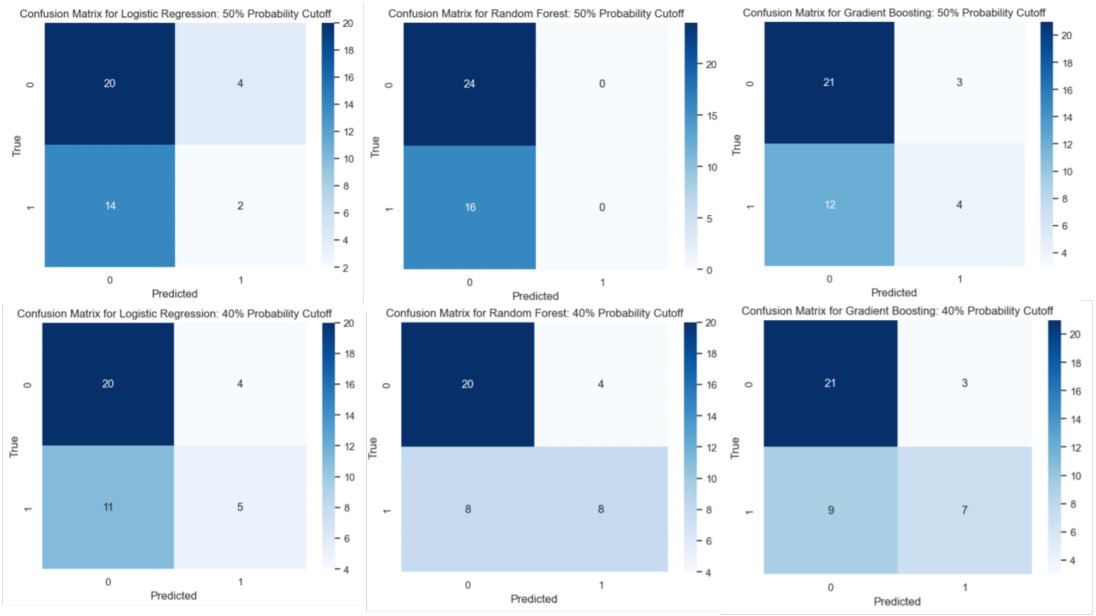


Figure 9: Out of Sample Confusion Matrices for 50% and 40% Probability Cutoffs

5 Conclusions and Considerations for Further Research

Anyone who could accurately and consistently predict the stock market would be very rich. Unfortunately, our classification models do not seem likely to deliver us an early retirement. However, we were able to make at least two models which appeared to perform better than a coin flip. Furthermore, our models exhibited a low degree of variance in a test period that can be considered quite unusual compared to our training period. Of the three models we tested, Gradient Boost marginally outperformed the others across scores, which we believe may reflect the non-linearity of stock market returns which Gradient Boost was able to capture better than the other two models. It is possible however that

Gradient Boost's advantage over Random Forest reflects an over-fitting to the data.

The primary limitation of our models is inherent to our approach: by simplifying the problem to making monthly predictions, instead of daily or by the minute, we necessarily limited ourselves to a small sample size. Thus, future research might attempt to replicate these models after more monthly data has accumulated. Another limitation of our models may have been the overbroad classification of outcomes in the training phase. By training our models on two outcomes (high returns vs anything else), we ended up putting a typical month, where returns are slightly positive or negative, into the same bucket as extreme high-loss months such as October 2008 during the global financial crisis. This might have made it more difficult for our model to learn to identify clear patterns of difference between each of these categorically different kinds of outcomes and outcome of interest. This could be addressed by introducing more classes in the training phase or perhaps with more sophisticated weighting techniques to help our models differentiate between these data points.

6 Appendix

Figure 10: Feature Correlation Matrix

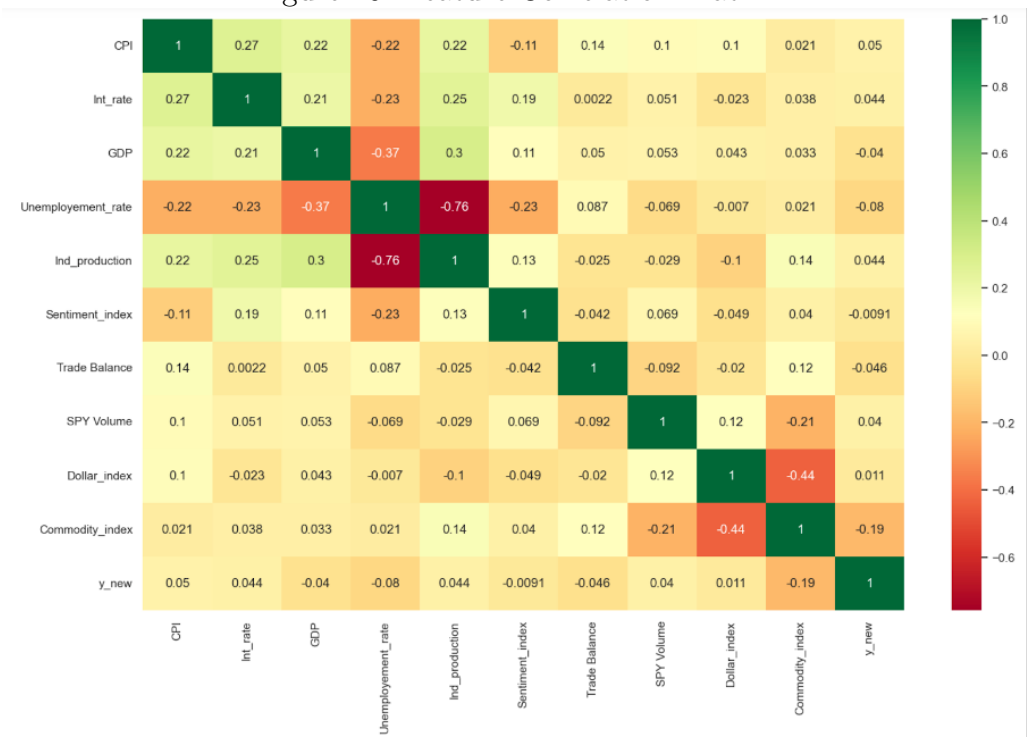


Table 3: Full List of Variables

Economic Variables	
CPI (Consumer Price Index)	Measures changes in the price level of a basket of consumer goods and services, indicating inflation or deflation.
Int_rate (Interest Rate)	The cost of borrowing or the return for lending, crucial for economic activity and monetary policy
GDP (Gross Domestic Product)	Represents the total monetary value of all goods and services produced, reflecting economic health
Unemployment	The percentage of the labor force that is jobless, indicating economic well-being and labor market conditions
Indpro (Industrial Production)	Measures the output of the industrial sector, including manufacturing, mining, and utilities
Consumer Sentiment	Indicates the confidence level consumers have in the economy's health, influencing their spending decisions
Trade	Represents the balance of exports and imports, affecting currency value and economic strength
SPX (S&P 500 Index)	A stock market index representing the stock performance of 500 large companies listed on stock exchanges in the USA
SPX_vol (S&P 500 Volume)	Measures the S&P 500 daily volume
VIX (Volatility Index)	Often referred to as the "fear gauge," it measures the market's expectation of volatility over the coming 30 days.
Dollar Index	Measures the value of the U.S. dollar against a basket of foreign currencies, indicating currency strength
CPI_Annual (Annual Consumer Price Index)	The year-over-year change in CPI, reflecting long-term inflation or deflation trends
Financial Variables	
ACWI (All Country World Index)	A stock index of global stocks for a broad benchmark of the global equity market
Earnings_yield	The estimated earnings of a stock divided by its price, offering a sense of value
Shiller PE Ratio	A price-earnings ratio based on average inflation-adjusted earnings from the previous 10 years, assessing market valuation
PE_ratio (Price-to-Earnings Ratio)	The ratio for valuing a company by comparing its current share price to its per-share earnings
S&P 500 Dividend Yield	The dividend yield of the S&P 500, showing the return investors get for holding the stocks
High_yield_return	The returns from high-yield bonds, indicating risk appetite and credit conditions
Comm_index (Commodities Index)	Tracks the overall price movement in commodity markets
Technical Analysis Variables	
RSI (Relative Strength Index)	Measures the magnitude of recent price changes to evaluate overbought or oversold conditions in the price of a stock
12-Period EMA	12-period Exponential moving average
24-Period EMA	24-period Exponential moving average
OBV (On-Balance Volume)	Uses volume flow to predict changes in stock price

Figure 11: Feature Importance: Logistic Model

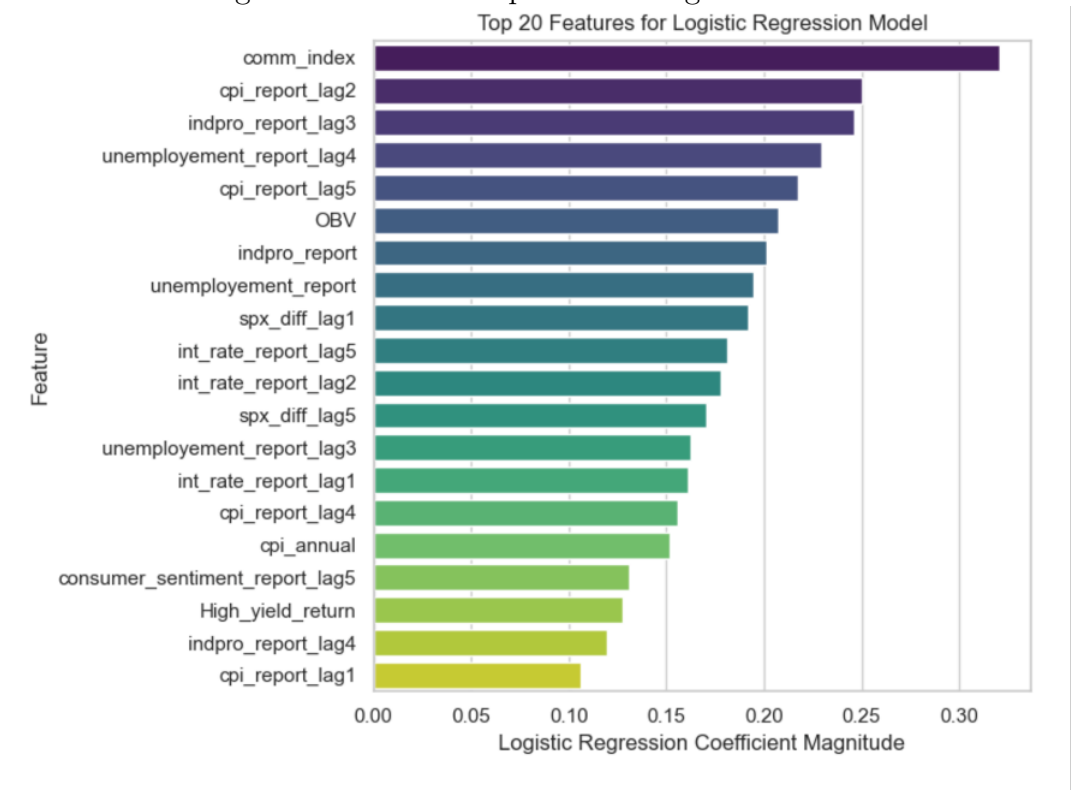
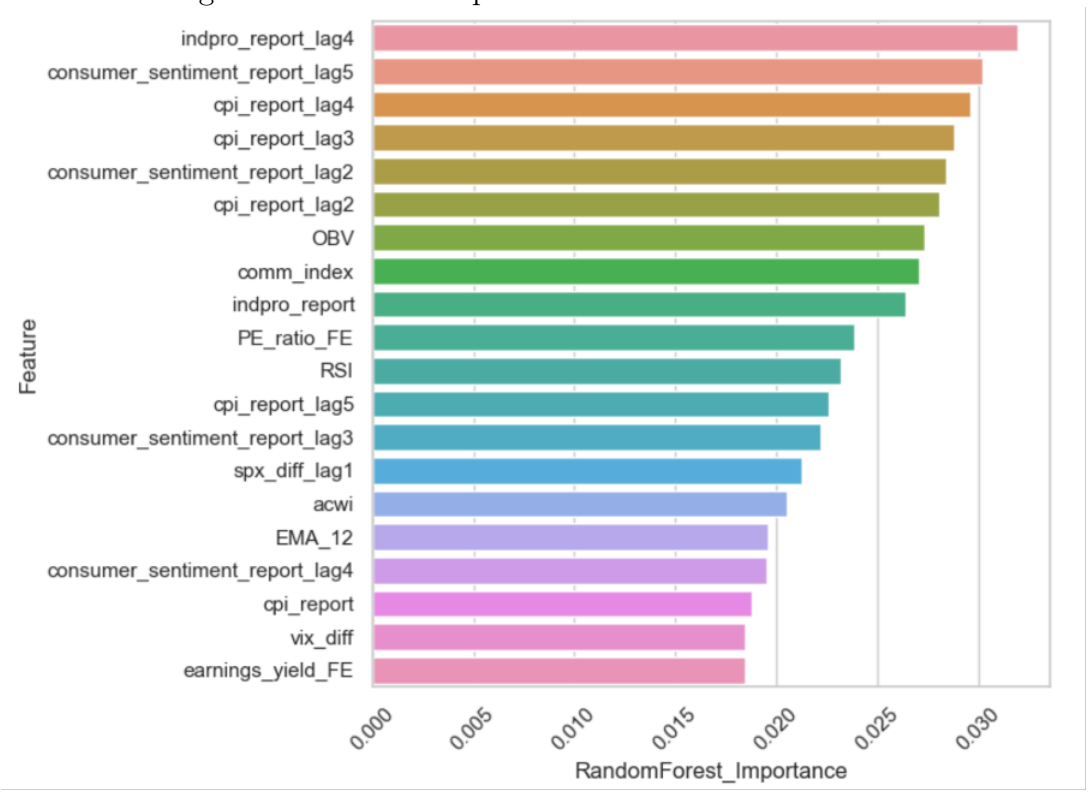


Figure 12: Feature Importance: Random Forest Model



7 References

Ang, Andrew, and Geert Bekaert. Stock Return Predictability: Is It There?, Feb. 2006, https://doi.org/10.3386/w8207.

Lim, Yenwee. "State-of-the-Art Machine Learning Hyperparameter Optimization with Optuna." Medium, Towards Data Science, 1 Apr. 2022, towardsdatascience.com/state-of-the-art-machine-learning-hyperparameter-optimization-with-optuna-a315d8564de1.

Lohrmann, Christoph, and Pasi Luukka. "Classification of intraday SP500 returns with a random forest." International Journal of Forecasting, vol. 35, no. 1, 2019, pp. 390–407, https://doi.org/10.1016/j.ijforecast.2018.08.004.

Sousa, Ricardo M., et al. "Predicting asset returns in the BRICS: The role of macroeconomic and fundamental predictors." International Review of Economics amp; Finance, vol. 41, 2016, pp. 122–143, https://doi.org/10.1016/j.iref.2015.09.001.