Can you create a Profitable Stock Market Trading Strategy from a Machine Learning Pipeline?

Henry Pasts
Brown University Data Science Initiative

Github Repository: https://github.com/henrypasts/data1030_project.git

Introduction

In recent years, an increasing amount of work in machine learning has centered around stock market prediction due to the monetary incentive of creating a successful prediction strategy. Stock market prediction is nothing new, and much of the prior research revolves around using technical analysis tools (CMT Association)[1]. Technical Analysis[2] tools use price and volume to distinguish the trend, momentum, overbought/oversold conditions, etc of a security in order to gauge the attractiveness of an entry point at a given time.

One criticism of technical analysis is that it does not *directly* make predictions of future price movements. Technical analysis has focused on formulating indicators of different measures of price such as the open price, close price, and volume that only track the past x periods of price to calculate the value of an indicator for a given day. There are varying degrees of complexity of indicators, but they all do not explicitly make statistical inferences into the future.

Target Variable

Initially, the percent change of the next day's price from the current close price was the target variable, but the modeling approach shifted to solving a classification problem where the target variable is 1 if the next day's Close is higher than the current Close and 0 if it is lower. When predicting percent change, most models predicted a constant value for the percent change of the next day's price, which minimized the root mean squared error but was not an ideal result.

Features used for Prediction

The original dataset pulled the "Date", "Open", "High", "Low", "Close" and "Volume" of the S&P 500 ETF SPY for the last 30 years using the Yahoo Finance API (yfinance).

[1] https://cmtassociation.org/
[2] https://www.investopedia.com/terms/t/technicalanalysis.asp

Since the focus was on combining Technical Analysis tools and Machine Learning, 30 Technical Analysis indicators were added to the feature matrix. The lookback window for these indicators was optimized using the training data and attempted to find lookback periods that correlated to the next day's price. The optimization can be found in 'src/indicator_optimization.ipynb'. A full description of the indicators is at 'src/indicator_description.ipynb'.

Previous Work

I see other research as a good way to set model expectations.

Kamalov, Smail, Gurrib, 2021,[a] used Deep Learning to predict the direction of the next day's price of the S&P 500 with a non-trivial accuracy of 56%.

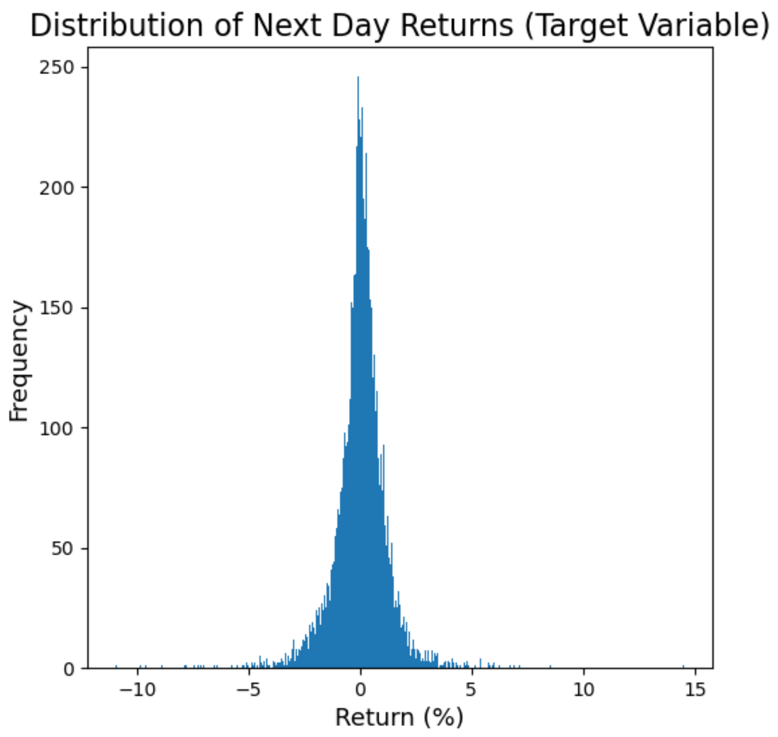
Fuster and Zou at Stanford University[b] used ARIMA, Logistic Regression, Gaussian Discriminant Analysis (GDA), Support Vector Machines, and Neural Networks to predict the spread of a stock pair using a co-integration with S&P500 data and were able to predict the direction of spreads between individual stocks in the S&P 500 with an accuracy and precision of 57-58%, as well as a recall of 90%+.

Other research not cited seems to indicate a 54-57% ability to predict the next day's price with mediocre precision and high recall. The high recall is not surprising, given most points have positive direction. My initial goal was to offer risk protections against big losses, however a new goal has been to selectively choose the days which have the highest probability of increasing the next day. With the data biased on the upside, it is difficult to offer much risk protection, thus creating a strategy that beats the 'Buy/Hold' with a similar level of risk metrics may offer good relative returns.

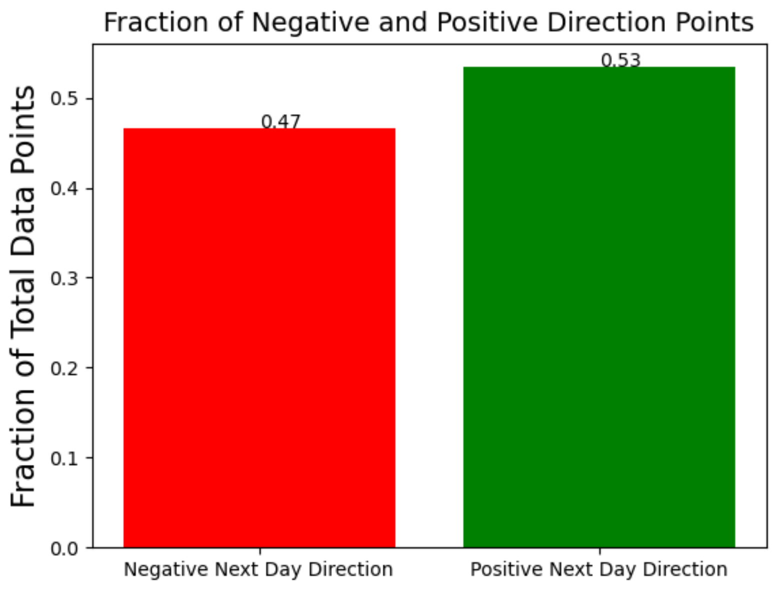
EDA

After optimizing indicators as a part of my feature engineering, there were 7269 data points and 39 features.

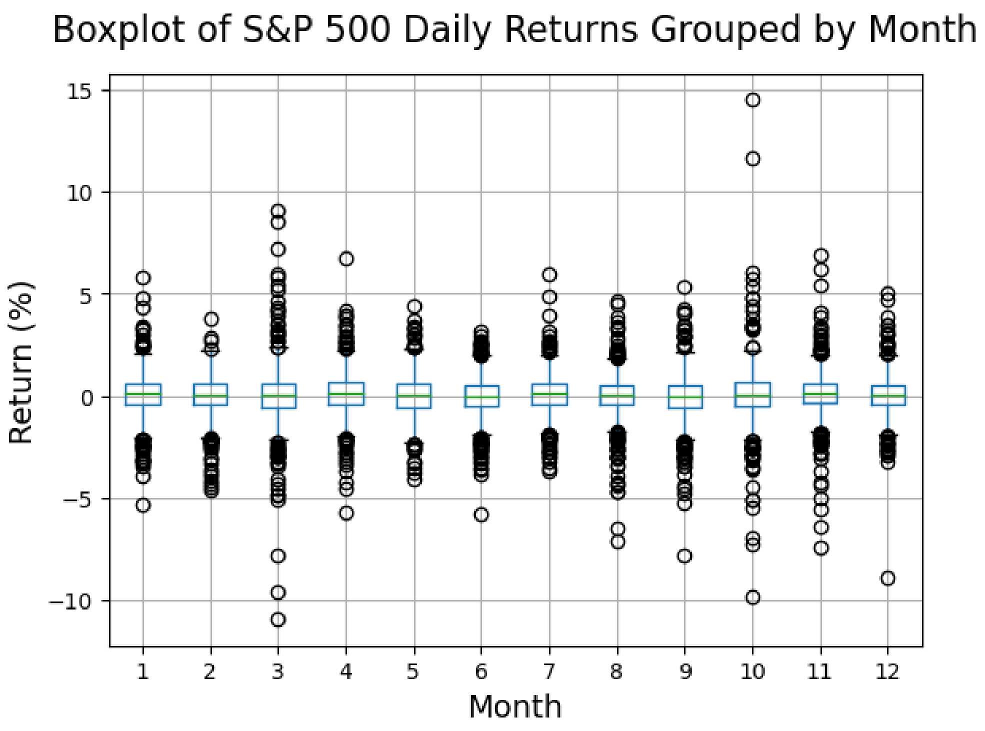
I found that the distribution of next day returns is normally distributed with a mean of 0.036% and a standard deviation of 1.2%. When training models to predict this variable, models would simply predict a constant value slightly above zero, as that was the percent change that minimized the root mean squared error for the training/validation sets, so classification was researched instead.



53% of the next day values are positive and 47% are negative, as shown in the bar chart below, but I found that these percentages change depending on the dataset, with the test set having 54.7% positive values.



Returns by month vary and some months express higher variance than others, thus a month feature was added to the feature matrix to account for seasonality effects and was ordinally encoded. March and October seem to have more tail-heavy events which may offer some probabilistic properties for predictions during those months.



Methods

Splitting Strategy

Initially, I split the data into sets based on time, with training composing the first 80% of the data, validation data composing 80-90% and the test data composing the final 10% of the data. I found that this approach resulted in overfitting and all models simply returned 1 regardless of input (baseline model result). I thus experimented with different approaches, but I found that splitting the first 90% of data randomly into train and validation and tuning the critical probability with the validation data resulted in results that did not simply return 1 for all inputs, so this splitting strategy was implemented.

Data Preprocessing

All features besides the “Month” feature were scaled with a StandardScaler. An OridinalEncoder was used for the “Month” feature. The optimized indicators had a lookback window of up to 200 days to calculate the value for a given point, thus the first 200 points in the data before splitting and preprocessing were dropped. Additionally, since there is no data for the directionality of the last point, it also was dropped.

ML Pipeline

4 Machine Learning algorithms, LogisticRegression, DecisionTreeClassifier, RandomForestClassifer, and Support Vector Classifier (SVC) were optimized to produce the best accuracy/profitability score as their evaluation metric in the test set (last 10% of the data).

For LogisticRegression, the ‘lbfgs' solver which is only compatible with ‘l2’ regularization was used and the “C” regularization parameter with values between 0.1 to 1000 spaced in log was tuned.

For the DecisionTreeClassifier, the criterion for splits, 'gini', 'entropy', 'log_loss', as well as which splitting method (either ‘random’ or ‘best’) were tuned. The max_depth parameter was tuned with various values between 5 and 100 and for ‘None’, which increases depth until leaves are pure.

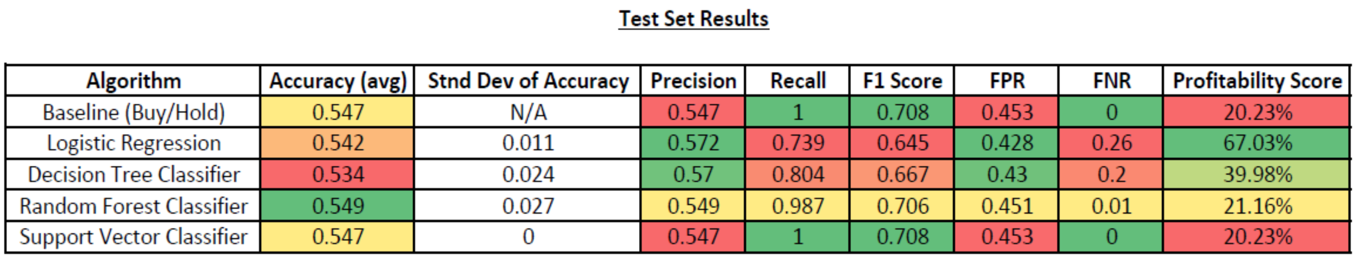
For the RandomForestClassifier, max_depth parameter for values between 5 and 100 as well as the max_features parameter that determines how many of the features to use in the splits were both tuned.

Finally, for the SVC, the C parameter with values between 0.1 and 1000 spaced in log and gamma parameter between “scale” and “auto” were tuned.

Training all algorithms was conducted by the “optimize_algorithm” function in “results/results.ipynb” and optimized models used a GridSearchCV with a ColumnTransformer To deal with uncertainties in splitting and nondeterministic algorithms, it loops through 10 different random states and calculates the average accuracy for the random states and the standard deviation of these accuracies. A ‘val_size’ param was optimized, and this parameter tunes how much of the randomly shuffled data is used to get the best critical probabilities for the algorithms based on accuracy.

Results

Results vs. Baseline



Test Set Results

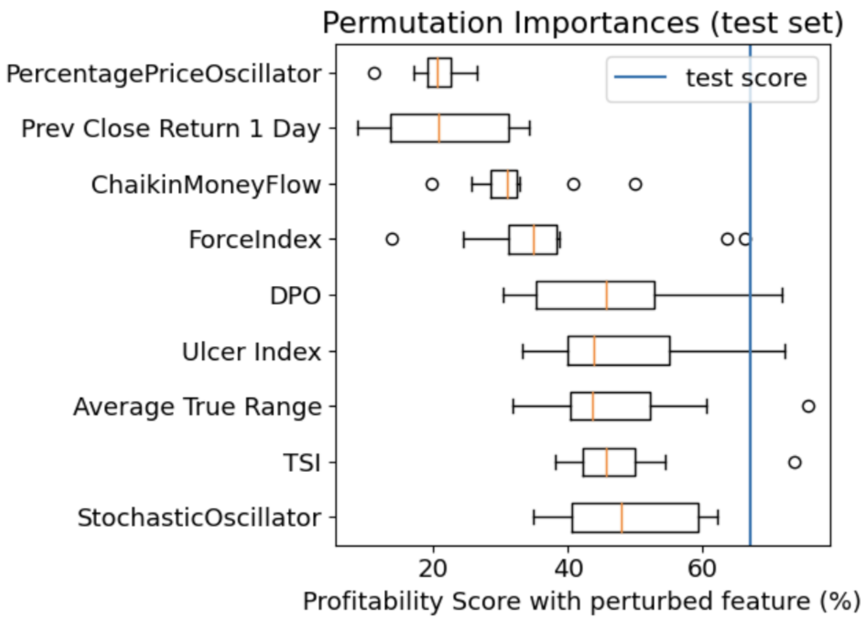
Algorithm	Accuracy (avg)	Stnd Dev of Accuracy	Precision	Recall	F1 Score	FPR	FNR	Profitability Score
Baseline (Buy/Hold)	0.547	N/A	0.547	1	0.708	0.453	0	20.23%
Logistic Regression	0.542	0.011	0.572	0.739	0.645	0.428	0.26	67.03%
Decision Tree Classifier	0.534	0.024	0.57	0.804	0.667	0.43	0.2	39.98%
Random Forest Classifier	0.549	0.027	0.549	0.987	0.706	0.451	0.01	21.16%
Support Vector Classifier	0.547	0	0.547	1	0.708	0.453	0	20.23%

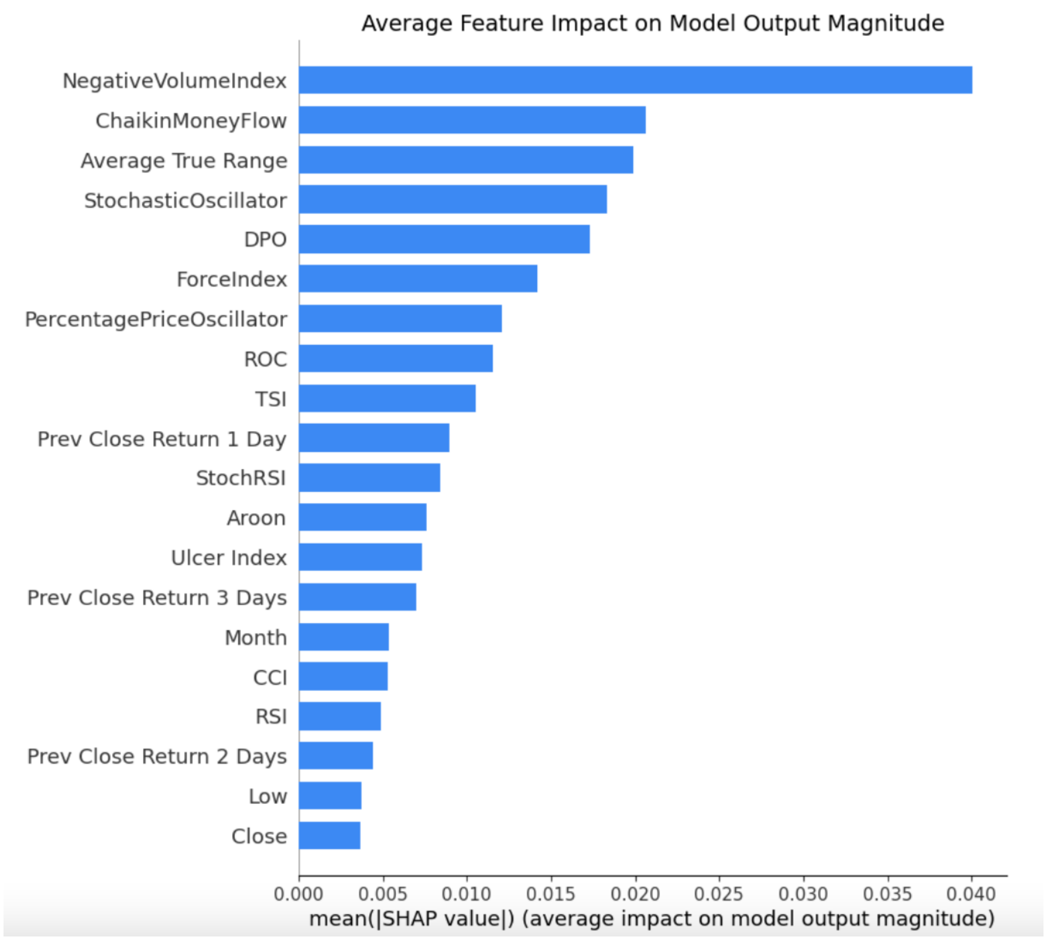
Profitability Score is a rough estimate of the rate of return one would receive if they had followed the strategy from the end of 2019 to September 2022.

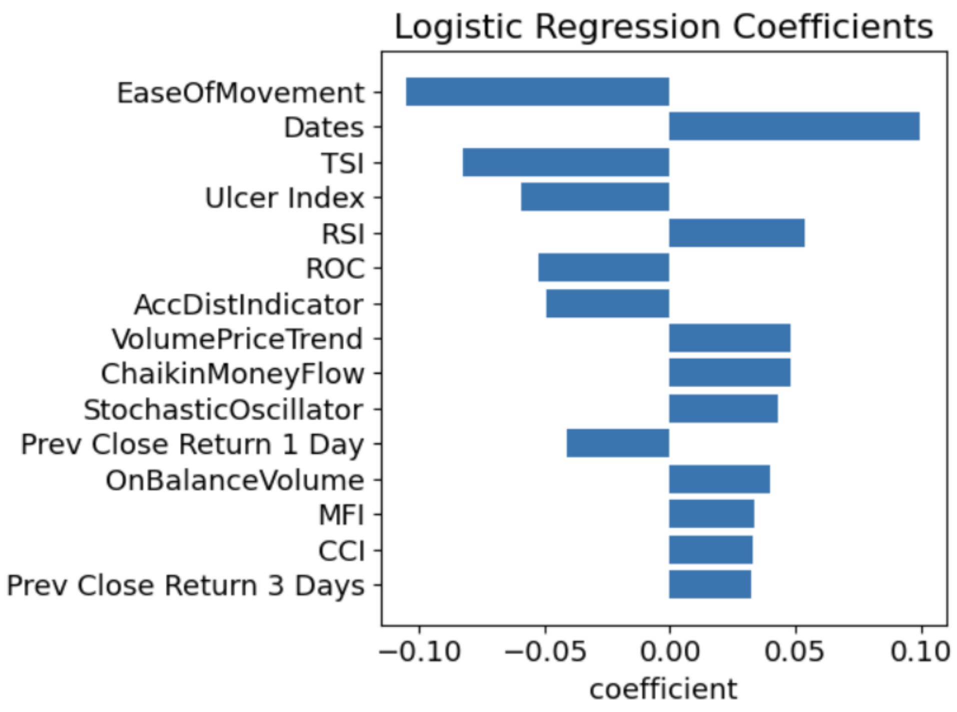
Best Model

The best model found was LogisticRegression. It had an accuracy almost in line with the baseline test score, while having a higher precision and a recall and F1 Score that produced a higher profitability score.

Global Feature importances



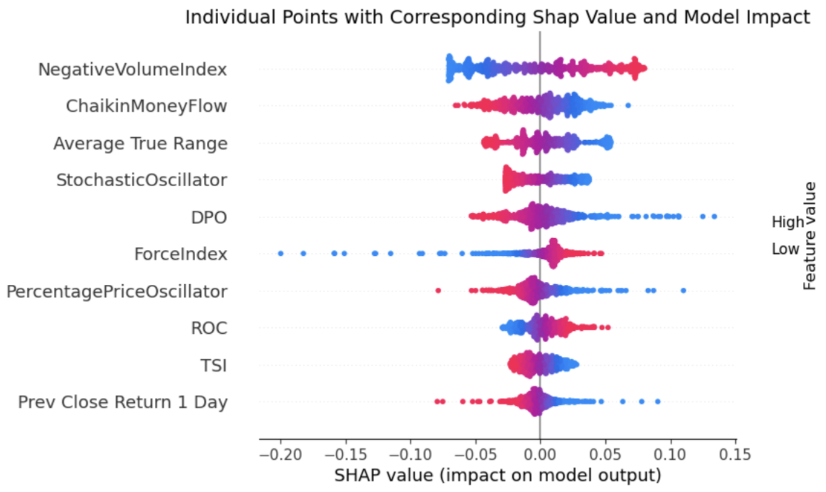

Average Feature Impact on Model Output Magnitude
NegativeVolumeIndex
ChaikinMoneyFlow
Average True Range
StochasticOscillator
DPO
ForceIndex
PercentagePriceOscillator
ROC
TSI
Prev Close Return 1 Day
StochRSI
Aroon
Ulcer Index
Prev Close Return 3 Days
Month
CCI
RSI
Prev Close Return 2 Days
Low
Close
0.000
0.005
0.010
0.015
0.020
0.025
0.030
0.035
0.040
mean(|SHAP value|) (average impact on model output magnitude)


Logistic Regression Coefficients
EaseOfMovement
Dates
TSI
Ulcer Index
RSI
ROC
AccDistIndicator
VolumePriceTrend
ChaikinMoneyFlow
StochasticOscillator
Prev Close Return 1 Day
OnBalanceVolume
MFI
CCI
Prev Close Return 3 Days
−0.10
−0.05
0.00
0.05
0.10
coefficient

The three global feature importance measures above (using the best model) show that Average True Range, PercentagePriceOscillator, and ChalkinMoneyFlow all have significant global SHAP values, large negative effects on model profitability when they are perturbed, and large coefficients in the LogisticRegression model. This is strong evidence that these are important features. Some notable features with low global SHAP values such as Prev Close Return 2 Days, Low, and Close may be less important features and this is confirmed by permutation tests for these features and a coefficient analysis of all features.

SHAP Values for Local Feature Importance

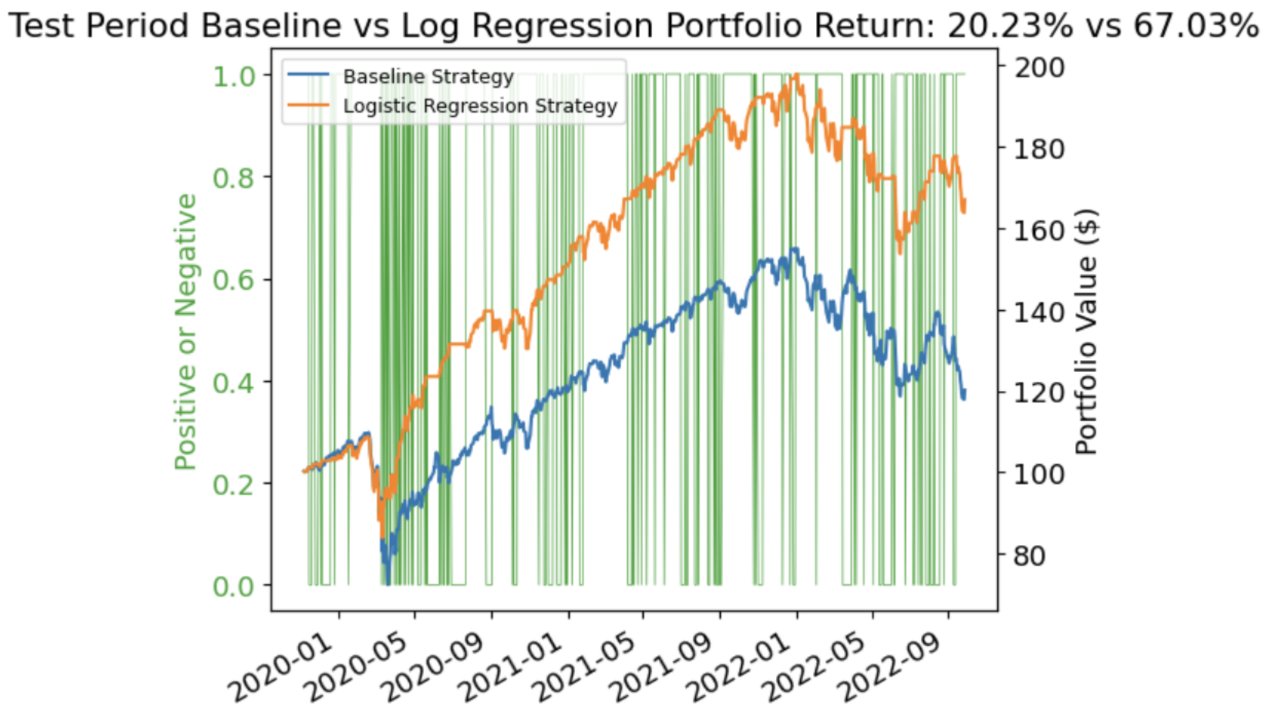
The below chart shows how the features of each individual point make up the distribution of the feature SHAP values and their impact on the model's output. This shows that for many features the actual feature value can have a powerful effect either positively or negatively on the model output (such as NegativeVolumeIndex), but for other features it favors positive or negative (i.e. PercentagePriceOscillator).

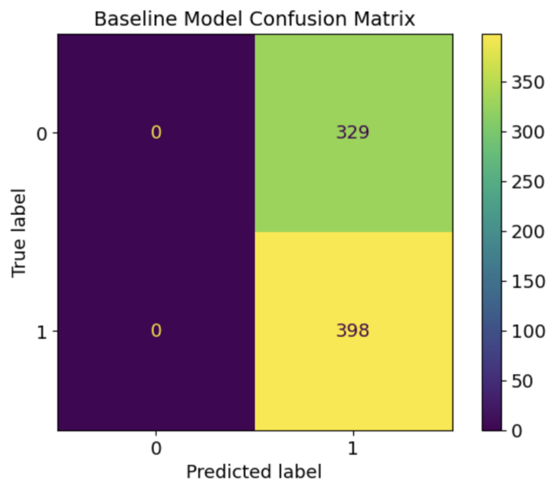


Interpretation of Model Results

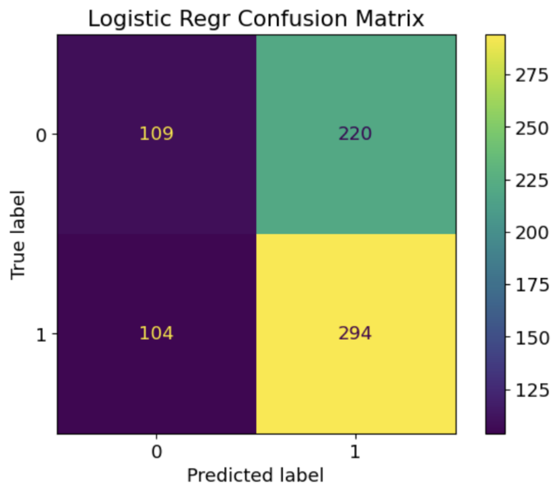
The charts below showcase the effectiveness of the model and its impact on profitability to correctly predict 109 of the 329 negative direction days, without letting too many crucial positive days get away (only predicting 104 of 398 positive days incorrectly).

It is important to note that these are only hypothetical returns, as one would have to run the model close to the end of a trading day and take a position before the market closed in order to receive the next day's return.





```
False Negative Rate: 0.0
False Positive Rate: 0.45
```



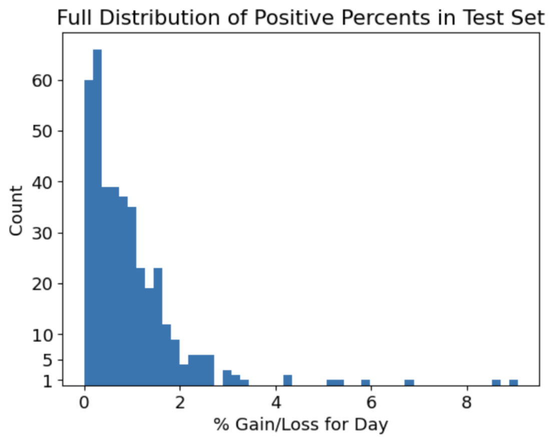
```
False Negative Rate: 0.26
False Positive Rate: 0.43
```

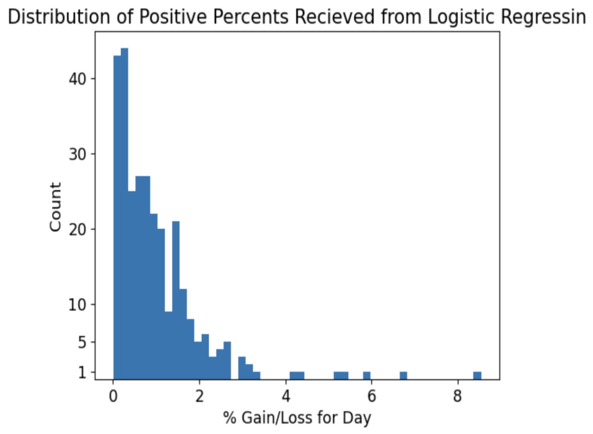
Surprising/Interesting

Although the LogisticRegression model misclassified 104 positive points as negative, or roughly 25% of all positive points in the test set, it manages *not* to miss much of the upside offered by the market during the test period and produce a return roughly 3 times that of the market while having roughly the same level of drawdown (max peak to trough percentage decline during the period).

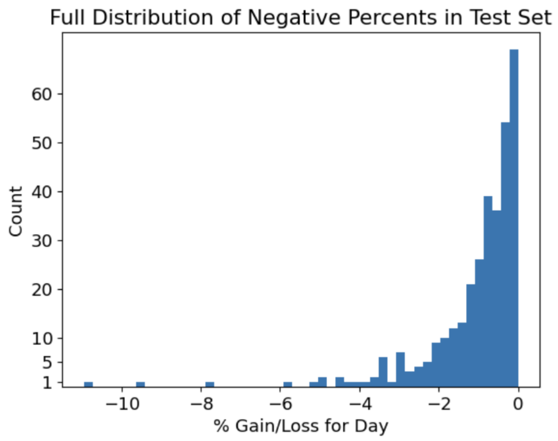
We can further explain how the model does so well against the baseline by showing the distribution of returns received by the model.

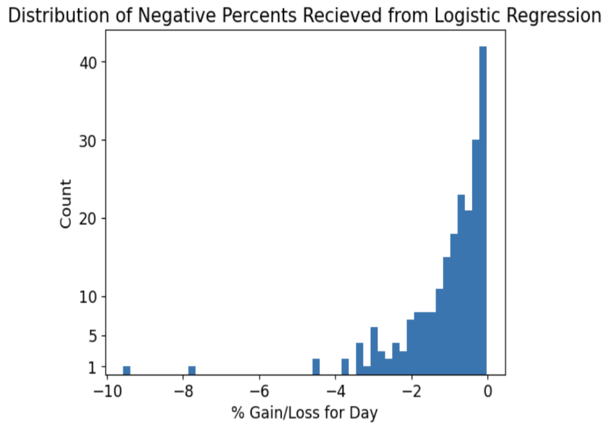
Below, we see that the model is able to get most of the important positive returns and mostly misses returns close to 0%, which, in the long run, do not affect portfolio returns too severely.





The model does well at predicting negative points that are important (points in the -4% to -6% range, as well as one point that was -10%+), as illustrated below:





In the long run, if the model is able to receive most of the positive percent returns, while avoiding negative percent returns above -4%, this will drastically affect model returns.

Below is a brief hypothetical representation of this phenomenon. The portfolio that avoids a few -5% returns has a drastically more positive return. This fact was particularly important when the model predicted March 14th, 2020 to be negative when the S&P 500 declined -10%+.

Volatile 5% Return Stream (starting at $100)

Return:	+5%	-5%	+2%	-5%	+2%
Portfolio Value:	105	99.8	101.8	96.7	**98.6**

Overall -1.4% return

Steady 5% Return Stream (starting at $100)

Return:	+5%	0%	+2%	0%	+2%
Portfolio Value:	105	105	107.1	107.1	**109.1**

Overall 9.1% return

Outlook

Weak Spots of Modeling Approach

Due to the time series nature of the data, a more sophisticated splitting approach for training and validation should be explored to better account for autocorrelation.

How to improve model or interpretability

More indicators can be added and tested and others can be removed based on the current global feature importance analysis in order to create a more robust feature matrix with higher explanatory power.

Additional Techniques

Deep Learning models such as Neural Networks to create a better model from the engineered features. Additionally, adding more lagged features to the feature matrix might be explored.

Additional Data

Additional data should be added as new stock market data comes in and the model should be retrained periodically to account for feature and target variable distribution shifts in the future.

References

[a] Kamaloc, Firuz, Smail, Linda, Gurrib, Ikhlaas (2021). FORECASTING WITH DEEP LEARNING: S&P 500 INDEX.

[b] Fuster, Alex, Zou, Zhichao (2018). Using Machine Learning Models to Predict S&P500 Price.