Exam 3 Report - Tuo Li

Note: This report primarily aims to answer exam questions and introduce the workflow, with minimal code exposure. For the complete executed code, please refer to **LI_Code.ipynb** or **LI_Code.html**.

To enhance the display of wide datasets/tables and ensure smooth navigation of links, this report is formatted using horizontal pages. Thank you!

Summary

This project serves as my answer to Exam 3, with the objective of developing a machine learning model to predict positive market movements (uptrends).

The selected asset is QQQ, an exchange-traded fund (ETF) that tracks the Nasdaq 100 Index. The prediction focuses on daily returns using a binomial classification approach, leveraging ten years of historical market data, with the dependent variable labeled as $[0, 1]$.

Throughout the model development process, I will address all exam questions while covering key steps, including data preparation, entropy analysis, feature selection, model building, tuning, and evaluation. Finally, I will implement the selected model to conduct backtesting, assessing its real-world performance.

Content

Preparation

P1. Obtain Trading Data

Obtain the 10 year QQQ price data from 2015 all the way till right before the exam, 15 May 2025. The data was extracted from Yahoo finance with Python.

High-level overview of the data:

- Index: 2608 entries (2015-01-02 to 2025-05-15)
- Total Columns: 6

#	Column	Non-Null Count	Dtype
0	Adj Close	2608 non-null	float64
1	Close	2608 non-null	float64
2	High	2608 non-null	float64
3	Low	2608 non-null	float64
4	Open	2608 non-null	float64
5	Volume	2608 non-null	int64

First 5 rows of the data:

Date	Adj Close	Close	High	Low	Open	Volume
2015-01-02	95.123268	102.940002	104.199997	102.440002	103.760002	31,314,600
2015-01-05	93.727943	101.430000	102.610001	101.139999	102.489998	36,521,300
2015-01-06	92.471214	100.070000	101.750000	99.620003	101.580002	66,205,500
2015-01-07	93.663261	101.360001	101.599998	100.489998	100.730003	37,577,400
2015-01-08	95.455971	103.300003	103.500000	102.110001	102.220001	40,212,600

Statistical summary of the data:

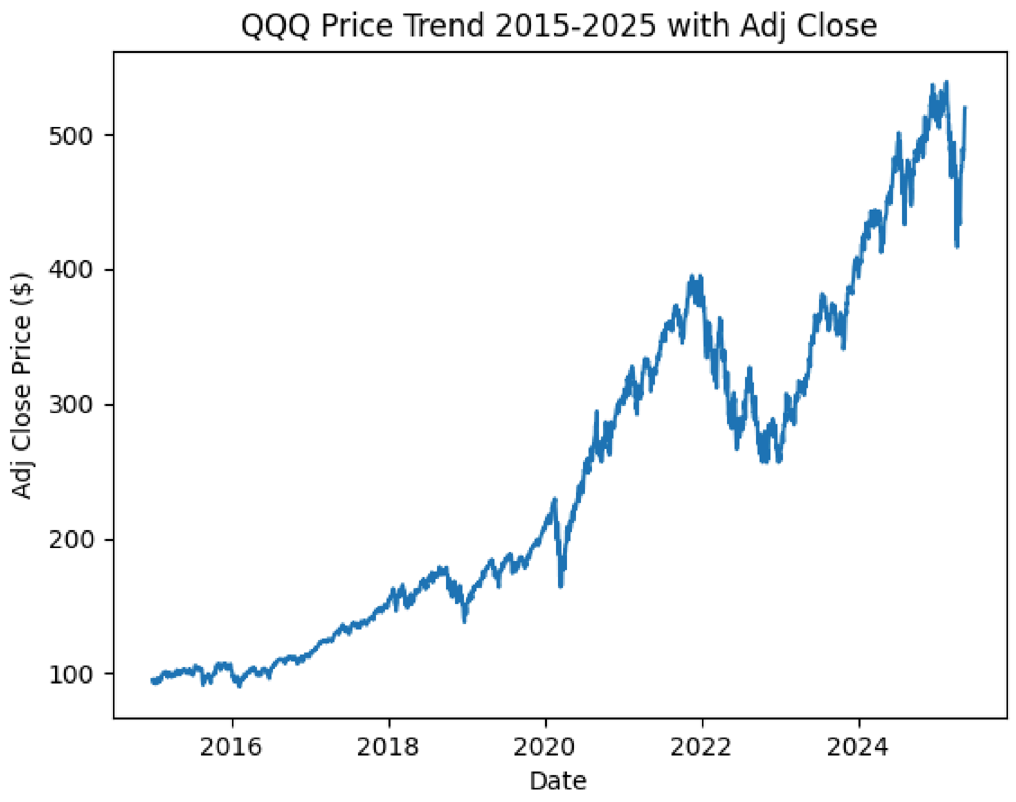
Column	Count	Mean	Std Dev	Min	25%	50%	75%	Max
Adj Close	2608.0	246.69	125.57	89.91	135.83	209.06	345.81	538.72
Close	2608.0	252.92	124.10	96.32	143.43	215.56	352.16	539.52
High	2608.0	254.74	125.05	97.05	143.88	216.99	355.19	540.81
Low	2608.0	250.82	123.03	84.74	142.67	213.63	349.32	536.46
Open	2608.0	252.86	124.10	94.23	143.51	215.91	352.78	539.73
Volume	2608.0	41,463,810	22,064,210	7,079,300	25,628,580	36,331,900	51,633,750	198,685,800

Back to Content

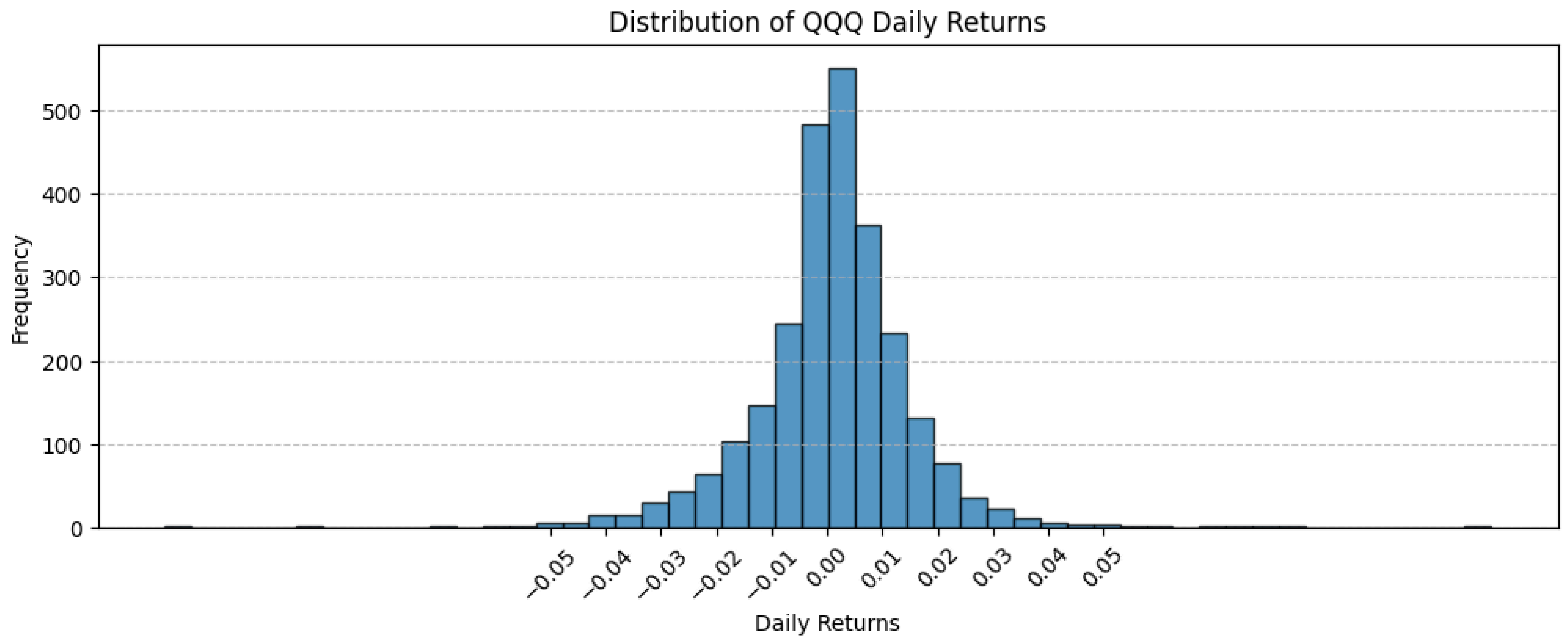
P2. Decide Target

Since the project is about predicting the trend for daily returns in binomial classification $[0, 1]$, I will develop this column first, and call it `Target` .

Before deciding the classification threshold to for the target, let's observe the data . I will use `Adj Close` to represent the day's price as it counts more process factors (e.g. corporate action, splits , dividends) to the share price.



Then, create the column of `Daily Return` and observe the distribution.



As illustrated in the histogram, QQQ's daily returns follow a roughly normal distribution, with a slight positive skew in the mean.

To optimize classification performance, **I will set the threshold at -1%** for following reasons:

- An uneven class split helps maintain model robustness while mitigating overfitting.
- A key objective is to generate meaningful predictions for positive market moves (class 1), and a broader class distribution enhances precision in identifying the positive movement.

Under this target definition, returns below -1% are labeled as 0. A target value of 1 is assigned if the next day's closing price is at least 99% of the current day's closing price, indicating a buying opportunity; otherwise, no action is taken.

After setting up threshold and creating the column of `Target`, 2191 instances fall under Class 1, while 417 belong to Class 0, resulting in an approximate 5:1 class distribution, which is within an acceptable range.

Check the first 5 rows of the data now:

Date	Adj Close	Close	High	Low	Open	Volume	Daily Return	Target
2015-01-02	95.123268	102.940002	104.199997	102.440002	103.760002	31,314,600	NaN	0
2015-01-05	93.727943	101.430000	102.610001	101.139999	102.489998	36,521,300	-0.014669	0
2015-01-06	92.471214	100.070000	101.750000	99.620003	101.580002	66,205,500	-0.013408	1
2015-01-07	93.663261	101.360001	101.599998	100.489998	100.730003	37,577,400	0.012891	1
2015-01-08	95.455971	103.300003	103.500000	102.110001	102.220001	40,212,600	0.019140	1

The prepration is done, let's visit the questions.

Back to Content

A. Explanation of Entropy in Classification

What does entropy reveal about the quality of the partitions in a classification problem?

Answer below with True / False and explain the reasoning behind your choice.
(a) High entropy means the partitions are pure.
(b) High entropy means the partitions are impure.

Answer:

Entropy quantifies the impurity or uncertainty within a partition.

High entropy implies that the classes in a partition are mixed (e.g., 50% Class A and 50% Class B). This means the partition is impure, as there is no clear dominance of a single class. Low entropy (near 0) indicates a pure partition where one class dominates (e.g., 100% Class A).

Therefore, Statement (a) is False, and Statement (b) is True.

This is also reflected with the formula of entropy:

$$-\sum_{k=1}^{K} p_k log_2(p_k)$$

where K is total amount of classes and p_k is the probability of class k.

$$\because 0 \leq p_k \leq 1$$

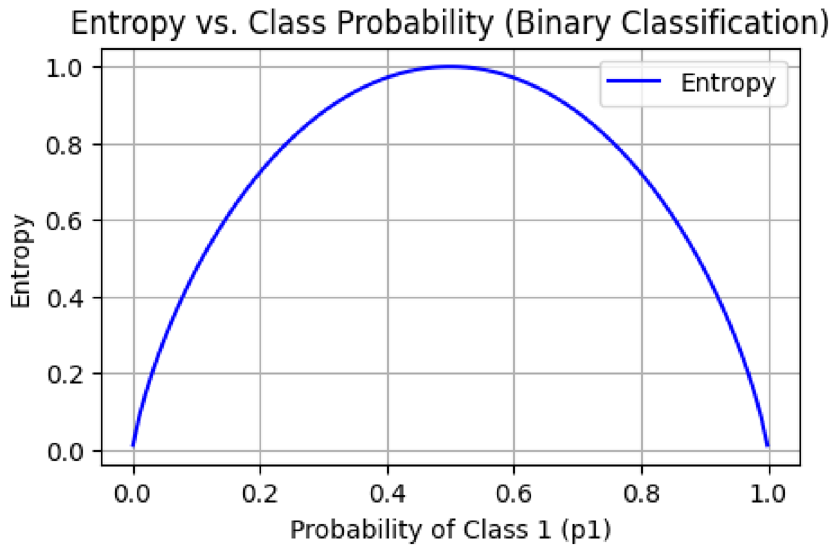
$$\therefore -1 \leq p_k log_2(p_k) \leq 0$$

And as shown above, the value of $p_k log_2(p_k)$ approaches 0 as p_k nears either 0 or 1. This negative term becomes most significant when p_k is close to the middle point: 0.5, where uncertainty is highest. Consequently, entropy, computed as the sum of these negative values, multiplied by -1, results in a larger positive value, reflecting greater disorder in the system.

In our case, $k = 2$, and the entropy formula will be:

$$-p_k log_2(p_k) - (1 - p_k) log_2(1 - p_k)$$

As discussed, the entropy values will be 0 when $p_k = 0 \; or \; 1$, and it will reach to its maximum 1 when $p_k = 0.5$. The visualization is below:



In our case, the probability of QQQ ETF from 2015-2025 to reach Class 1 is:

$$p_1 = \frac{Count \; of \; Class \; 1}{Total} = \frac{2191}{2608} \approx 84.01\%$$

Therefore, the entropy is:

$$-p_1 log_2(p_1) - (1 - p_1) log_2(1 - p_1) \approx 0.634$$

Back to Content

B. Feature Selection Using the Funnelling Approach

Perform feature selection for a machine learning model using a multi-step process by combining techniques from fillter, wrapper, and embedded methods.
(a) Explain the feature selection process using the three categories of feature selection methods, step by step.
(b) Justify the selection of features retained at each step.
(c) Provide the final list of selected features.

Answer:

B1. Create Features

B1.1 Create Features from Trading Data

At the first step of creating features, we will follow some common practices and include following candidtates:

SMA (Simple Moving Average):

$$SMA = \frac{1}{N}\sum_{i=1}^{N} P_i$$

where P_i is the praice at period i, and N is the number of days in our case.

EMA (Exponential Moving Average):

$$EMA_t = \alpha P_t + (1-\alpha) EMA_{t-1}$$

where α is the smoothing factor, typically calculated as $\frac{2}{N+1}$. EMA gives more weight to recent prices and eacts faster to price changes than SMA, making it useful for short-term trading signals.

We will select features of SMA EMA with N range from 5 days to 50 days at the step of 5 days.

ATR (Average True Range):

$$ATR = \frac{1}{N}\sum_{i=1}^{N} TR_i$$

where TR_i means 'True Range' and is the greatest of the gaps between Current High and Current Low, Current High and Previous Close, or Current Low and Previous Close. In quantmod, N by default is 14 days. ATR is common way to measures volatility.

Bollinger Bands (BBANDS_L, BBANDS_U):

$$BBANDS_U = SMA + 2\sigma$$

$$BBANDS_L = SMA - 2\sigma$$

where σ is the standard deviation. These 2 bands represents the range of 95% price movement in the past N days. In quantmod, N here by default is 5 days.

RSI (Relative Strength Index):

$$RSI = 100 - \frac{100}{1 + RS}$$

$$RS = \frac{Average\ Gain\ over\ N\ periods}{Average\ Loss\ over\ N\ periods}$$

Here, we will set N to be 14 days. RSI measures momentum by comparing recent gains and losses.

MACD (Moving Average Convergence Divergence):

$$MACD = EMA_{12} - EMA_{26}$$

Here, we will set N to be 5 days. MACD tracks trend strength and reversals using two moving averages.

The price difference between high and low, open and close in one day

$$H - L = High - Low$$

$$O - C = Open - Close$$

These are simple features to show the variance within the day.

After generating these features, I removed rows with missing values, adjusting the dataset's starting date from 2015-01-01 to 2015-03-09.

Back to Content

B1.2 Create Macro Economy Features

Next, I believe the Federal Funds Effective Rate likely influences QQQ ETF movements, as the technology sector is highly sensitive to borrowing costs and liquidity, both of which are directly affected by rate fluctuations.

I have sourced monthly data from the Federal Reserve Bank of St. Louis, here is the last 5 rows of the Federal Rate data for reference:

Date	FEDFUNDS Rate
1954-07-01	0.80
1954-08-01	1.22
1954-09-01	1.07
1954-10-01	0.85
1954-11-01	0.83

As the Federal Rate data is recorded on a monthly basis, I mapped the monthly Federal Rate to our daily trading data using a forward-filling technique, ensuring that in the merged dataset, each day's federal rate correctly aligns with the corresponding monthly value.

The main dataset now contains 35 columns. Below is the data of 2015-03-09, as an example:

Column Name	Value	Column Name	Value	Column Name	Value
Adj Close	99.54	**SMA_15**	100.01	**SMA_35**	97.30
Close	107.72	**EMA_15**	99.49	**EMA_35**	98.01
High	107.99	**SMA_20**	99.26	**SMA_40**	96.83
Low	107.19	**EMA_20**	99.07	**EMA_40**	97.75
Open	107.63	**SMA_25**	98.43	**SMA_45**	96.53
Volume	26,237,100	**EMA_25**	98.67	**EMA_45**	97.53
Daily Return	0.00289	**SMA_30**	97.72	**ATR**	0.8657

Column Name	Value	Column Name	Value	Column Name	Value
Target	0.00	**EMA_30**	98.32	**BBANDS_L**	98.84
SMA_5	100.00	**BBANDS_M**	100.00	**BBANDS_U**	101.15
EMA_5	99.86	**RSI**	55.43	**MACD**	1.1287
SMA_10	100.23	**H-L**	0.7999	**O-C**	-0.0900
EMA_10	99.84	**FEDFUNDS**	0.11		

Back to Content

B1.3 Create Features from Daily News

News serves as a valuable indicator of market sentiment, particularly in the technology sector, where changes occur frequently. Daily headlines provide insight of how investor behavior is influenced by trending topics.

To analyze this impact, I used the GNews Python library to extract news titles from four major financial and technology media sources — Bloomberg, CNBC, MarketWatch, and TechCrunch, spanning 2015 to 2025. This dataset will help explore how news sentiment affects QQQ's price movements.

B1.3.1 Include the News Titles

I added 4 new columns to store daily news titles from each media source. If a media outlet has no news on a given day, the corresponding column remains blank.

Currently, these values are raw text and not yet usable as features in the dataset. Now we have 39 columns, it is not convenient to showcase all of them, but below is an example at 2015-03-10, showing **only the 4 new columns**:

Date	Bloomberg News Title	CNBC News Title	MarketWatch News Title	TechCrunch News Title
2015-03-10	Apple CEO Cook Sidesteps Shareholder Enthusiasm for Tesla	Randy Komisar Jason Fried Bayard Winthrop Neil Blumenthal Credit Suisse appoints Tidjane Thiam as new CEO Kate Rogers Billionaire teams up with	Montel Williams can no longer pitch payday loans to New Yorkers Bill Clinton still doesn't use email 'Shark Tank' winners: boomers' booming businesses	Yik Yak Quietly Dropped From Google Play Charts In October IAC Hires David Siegel As CEO Of Finance Website Investopedia Community-Based Shipping Service Roadie Launches Nationwide YC-Backed BuildScience's Platform Ties Hardware Systems Together In Big Office Buildings

Date	Bloomberg News Title	CNBC News Title	MarketWatch News Title	TechCrunch News Title
		NASA to mine the moon Marcus Lemonis		

B1.3.2 Use AI for Sentiment Analysis and Generate Relevant Features

All news titles are now stored in our dataset as strings. To extract their full value and transform them into meaningful features, I will use an AI LLM to interpret their content as numeric values. Specifically, for all news titles from a single media source on a given day, they will be converted into the following features:

- **Relevancy Score ("Media_Relevancy_Score")**: shows the relevance of news titles to QQQ ETF. The value is between 0 and 1, and 0 means not relevant at all while 1 means highly relevant.
- **Sentiment Score ("Media_Sentiment_Score")**: indicates whether the news titles expressed a positive and negative feeling that could impact QQQ's price. The value is between -1 and 1, and -1 means highly negative while 1 means highly positive.
- **Emotion ("Media_Emotion")** : identified if the day's news conveys any of the 4 emotions: Fear, Greed, Optimism, or Uncertainty. For example:
  - "AI stock bubble bursts" - Emotion: Fear
  - "Tech earnings beat expectations" - Emotion: Optimism
  - Value map - "Uncertainty": 0, "Fear": 1, "Optimism": 2, "Greed": 3
- **Emotional Intensity ("Media_Emotion_Intensity")** determines if the titles' emotion is either High or Low. For example:
  - "AI stock bubble bursts" - Emotion_Intensity: High
  - "Tech earnings beat expectations" - Emotion_Intensity: Low
  - Value map - "Low": 0, "High": 1
- **Keyword Impact ("Media_Keyword_Impact")** : counts occurrences of market-moving words, majorly focused on financial keywords like "crash," "bullish," "sell-off," "rate hike," "inflation," "earnings". The value should be an integer.
- **Market Event("Media_Market_Event")**: detects major macroeconomic events, such as Federal Reserve statements, inflation reports, earnings announcements, revolutionary technology launch. This is the most free-style output. I don't know how it will loooks like and how to use it yet.

Additionally, since some news may be entirely unrelated to QQQ and could distort the AI’s interpretation of the day's overall sentiment, I will instruct the AI to ignore titles that do not reference finance, economy, technology, politics, geopolitics, or macro-economic events.

B1.3.3 Make Decisions on 'Market_Event' and Add Sentiment Polarity for the News Titles

Unfortunately, after analyzing the columns of 'Market_Event' of the 4 media with checking samples, I found that they contain hundreds of diverse strings, and are not very accurate on summarizing the key events from the titles. Converting these into numeric data using one-hot encoding would significantly expand the number of columns, making the process inefficient without strong justification for its impact. Therefore, I have decided to exclude these columns from the later analysis.

Instead, I plan to explore the possibility of introducing **'Sentiment Polarity'** for the 4 media, to quantify the degree of polarization in daily news titles. The classification mechanism is straightforward:

- 1 represents positive sentiment if Sentiment_Score > 0.2
- 0 indicates neutral sentiment or no polarity if Sentiment_Score is between -0.2 and 0.2
- -1 represents negative sentiment if Sentiment_Score < -0.2

Currently, the dataset contains 71 columns, with some serving as features and others not. Displaying all columns isn't practical, so below is a snapshot of 28 sentiment analysis feature columns on 2015-03-26, as an example.

Column Name	Value	Column Name	Value	Column Name	Value
Bloomberg_Relevancy_Score	0.4	**CNBC_Emotion_Intensity**	0	**MarketWatch_Sentiment_Polarity**	-1
Bloomberg_Sentiment_Score	0.1	**CNBC_Keyword_Impact**	0	**TechCrunch_Relevancy_Score**	0.3
Bloomberg_Emotion	0	**CNBC_Market_Event**		**TechCrunch_Sentiment_Score**	0.2
Bloomberg_Emotion_Intensity	0	**CNBC_Sentiment_Polarity**	0	**TechCrunch_Emotion**	2
Bloomberg_Keyword_Impact	0	**MarketWatch_Relevancy_Score**	0.7	**TechCrunch_Emotion_Intensity**	0
Bloomberg_Market_Event		**MarketWatch_Sentiment_Score**	-0.4	**TechCrunch_Keyword_Impact**	0
Bloomberg_Sentiment_Polarity	0	**MarketWatch_Emotion**	1	**TechCrunch_Market_Event**	
CNBC_Relevancy_Score	0.0	**MarketWatch_Emotion_Intensity**	1	**TechCrunch_Sentiment_Polarity**	0
CNBC_Sentiment_Score	0.0	**MarketWatch_Keyword_Impact**	2		
CNBC_Emotion	0.	**MarketWatch_Market_Event**	Earnings announcements		

Back to Content

B2. Select Features

As required, I will use filter, wrapper, and embedded methods to select the features for model building.

Here are all the 71 columns we have now:

`Adj Close` `Close` `High` `Low` `Open` `Volume` `Daily_Return` `Target`

`SMA_5` `EMA_5` `SMA_10` `EMA_10` `SMA_15` `EMA_15` `SMA_20` `EMA_20` `SMA_25` `EMA_25`

`SMA_30` `EMA_30` `SMA_35` `EMA_35` `SMA_40` `EMA_40` `SMA_45` `EMA_45`

`ATR` `BBANDS_L` `BBANDS_M` `BBANDS_U` `RSI` `MACD` `H-L` `O-C` `FEDFUNDS`

`Bloomberg News Title` `CNBC News Title` `MarketWatch News Title` `TechCrunch News Title`

`Bloomberg Analysis` `CNBC Analysis` `MarketWatch Analysis` `TechCrunch Analysis`

`Bloomberg_Relevancy_Score` `Bloomberg_Sentiment_Score` `Bloomberg_Emotion` `Bloomberg_Emotion_Intensity` `Bloomberg_Keyword_Impact` `Bloomberg_Market_Event`

`CNBC_Relevancy_Score` `CNBC_Sentiment_Score` `CNBC_Emotion` `CNBC_Emotion_Intensity` `CNBC_Keyword_Impact` `CNBC_Market_Event`

`MarketWatch_Relevancy_Score` `MarketWatch_Sentiment_Score` `MarketWatch_Emotion` `MarketWatch_Emotion_Intensity` `MarketWatch_Keyword_Impact` `MarketWatch_Market_Event`

`TechCrunch_Relevancy_Score` `TechCrunch_Sentiment_Score` `TechCrunch_Emotion` `TechCrunch_Emotion_Intensity` `TechCrunch_Keyword_Impact` `TechCrunch_Market_Event` `Bloomberg_Sentiment_Polarity` `CNBC_Sentiment_Polarity` `MarketWatch_Sentiment_Polarity` `TechCrunch_Sentiment_Polarity`

First, remove the non-feature columns:

- `Adj Close` `Close` `High` `Low` `Open` `Target`

Also remove the string feature columns that could not be included in the model building process:

- News titles: `Bloomberg News Title` `CNBC News Title` `MarketWatch News Title` `TechCrunch News Title`
- Analysis content: `Bloomberg Analysis` `CNBC Analysis` `MarketWatch Analysis` `TechCrunch Analysis`
- Market events: `Bloomberg_Market_Event` `CNBC_Market_Event` `MarketWatch_Market_Event` `TechCrunch_Market_Event`

After this, we have 53 columns now, and they can all be considered as **feature candicates**:

`Volume` `Daily_Return` `SMA_5` `EMA_5` `SMA_10` `EMA_10` `SMA_15` `EMA_15` `SMA_20` `EMA_20`

`SMA_25` `EMA_25` `SMA_30` `EMA_30` `SMA_35` `EMA_35` `SMA_40` `EMA_40` `SMA_45` `EMA_45`

`ATR` `BBANDS_L` `BBANDS_M` `BBANDS_U` `RSI` `MACD` `H-L` `O-C` `FEDFUNDS`

`Bloomberg_Relevancy_Score` `Bloomberg_Sentiment_Score` `Bloomberg_Emotion` `Bloomberg_Emotion_Intensity` `Bloomberg_Keyword_Impact`

`CNBC_Relevancy_Score` `CNBC_Sentiment_Score` `CNBC_Emotion` `CNBC_Emotion_Intensity` `CNBC_Keyword_Impact`

`MarketWatch_Relevancy_Score` `MarketWatch_Sentiment_Score` `MarketWatch_Emotion` `MarketWatch_Emotion_Intensity` `MarketWatch_Keyword_Impact`

`TechCrunch_Relevancy_Score` `TechCrunch_Sentiment_Score` `TechCrunch_Emotion` `TechCrunch_Emotion_Intensity` `TechCrunch_Keyword_Impact` `Bloomberg_Sentiment_Polarity` `CNBC_Sentiment_Polarity` `MarketWatch_Sentiment_Polarity` `TechCrunch_Sentiment_Polarity`

Back to Content

B2.1 Filter

The goal of this step is to detect and remove features with high multicollinearity.

B2.1.1 Variable Inflation Factors (VIF)

Multicollinearity occurs when two or more independent variables are highly correlated with one another in a regression model, and it can be detected using Variable Inflation Factors (VIF).

VIF score of an independent variable represents how well the variable is explained by other independent variables, and it is calculated by the following formula:

$$VIF = \frac{1}{1 - R^2}$$

R^2 value is determined to find out how well an independent variable is described by the other independent variables. A high value of R^2 means that the variable is highly correlated with the other variables.

VIF starts at 1 (no correlation) and has no upper limit. In this case, we will drop those variables with VIF higher than 5.

Afther VIF analysis, we conclude that the 21 features of
`SMA_5` `EMA_5` `SMA_10` `EMA_10` `SMA_15` `EMA_15` `SMA_20` `EMA_20`
`SMA_25` `EMA_25` `SMA_30` `EMA_30` `SMA_35` `EMA_35` `SMA_40` `EMA_40`
`SMA_45` `EMA_45` `BBANDS_L` `BBANDS_M` `BBANDS_U`
are interconnected due to their formulaic relationships. This correlation is expected, given their shared dependencies on price movements. Rather than arbitrarily selecting representative features, we will use SelectKBest to determine the representative for them.

Regarding other features, while some of them, such as `CNBC_Sentiment_Score` and `ATR` exceed the regular VIF threshold of 5, the scores are not significantly concerning. Therefore, they will be retained for further analysis.

B2.1.2 SelectKBest

SelectKBest helps us to ranks features based on statistical relevance to the target variable.

Since we already understand there is heavy multicollinearity among 21 features mentioned above, we can use SelectKBest with mutual_info_classif to decide which one will be kept as the representative of them.

At this step, I re-included the columns of `Target` to observe which one ofthe 21 features are most relevant to it, as indicated by F-score.

Column Name	F-score	Column Name	F-score	Column Name	F-score
SMA_5	0.0	**EMA_5**	0.0002124	**SMA_10**	0.002434
EMA_10	0.0	**SMA_15**	0.002672	**EMA_15**	0.0
SMA_20	0.009703	**EMA_20**	0.01098	**SMA_25**	0.02164
EMA_25	0.0	**SMA_30**	0.0145	**EMA_30**	0.0083
SMA_35	0.0223	**EMA_35**	0.01019	**SMA_40**	0.02194
EMA_40	0.004729	**SMA_45**	0.009562	**EMA_45**	0.0
BBANDS_L	0.006812	**BBANDS_M**	0.0	**BBANDS_U**	0.02363

As the result shown above, `BBANDS_U` is the most relevant among these highly correlated features and will serve as their representative.

Here are the selected features after VIF and SelectKBest. We have 33 features at this moment:

`Volume` `Daily_Return` `ATR` `BBANDS_U` `RSI` `MACD` `H-L` `O-C` `FEDFUNDS`

`Bloomberg_Relevancy_Score` `Bloomberg_Sentiment_Score` `Bloomberg_Emotion` `Bloomberg_Emotion_Intensity` `Bloomberg_Keyword_Impact`

`CNBC_Relevancy_Score` `CNBC_Sentiment_Score` `CNBC_Emotion` `CNBC_Emotion_Intensity` `CNBC_Keyword_Impact`

`MarketWatch_Relevancy_Score` `MarketWatch_Sentiment_Score` `MarketWatch_Emotion` `MarketWatch_Emotion_Intensity` `MarketWatch_Keyword_Impact`

`TechCrunch_Relevancy_Score` `TechCrunch_Sentiment_Score` `TechCrunch_Emotion` `TechCrunch_Emotion_Intensity` `TechCrunch_Keyword_Impact` `Bloomberg_Sentiment_Polarity` `CNBC_Sentiment_Polarity` `MarketWatch_Sentiment_Polarity` `TechCrunch_Sentiment_Polarity`

Review the VIF scores of the current features. While a few exceed 5 slightly, they remain within an acceptable range. We are moving forward with the next steps.

Column Name	VIF Score	Column Name	VIF Score	Column Name	VIF Score
Volume	2.52	**Daily_Return**	2.25	**ATR**	5.56
BBANDS_U	6.20	**RSI**	1.95	**MACD**	2.95
H-L	3.82	**O-C**	2.13	**FEDFUNDS**	1.78
Bloomberg_Relevancy_Score	2.02	**Bloomberg_Sentiment_Score**	3.79	**Bloomberg_Emotion**	1.49
Bloomberg_Emotion_Intensity	2.05	**Bloomberg_Keyword_Impact**	1.92	**CNBC_Relevancy_Score**	2.35
CNBC_Sentiment_Score	5.04	**CNBC_Emotion**	1.68	**CNBC_Emotion_Intensity**	2.21
CNBC_Keyword_Impact	2.19	**MarketWatch_Relevancy_Score**	2.24	**MarketWatch_Sentiment_Score**	4.88
MarketWatch_Emotion	1.48	**MarketWatch_Emotion_Intensity**	2.06	**MarketWatch_Keyword_Impact**	2.14
TechCrunch_Relevancy_Score	2.18	**TechCrunch_Sentiment_Score**	5.32	**TechCrunch_Emotion**	2.04
TechCrunch_Emotion_Intensity	1.32	**TechCrunch_Keyword_Impact**	1.21	**Bloomberg_Sentiment_Polarity**	3.64
CNBC_Sentiment_Polarity	4.88	**MarketWatch_Sentiment_Polarity**	4.53	**TechCrunch_Sentiment_Polarity**	4.01

Back to Content

B2.2 Wrapper

The Wrapper methods can help us differentiate feature subsets using a predictive model. We will use RFE and RFECV with XGBoost as classifier to detect and remove some of the least-important features.

B2.2.1 Recursive Feature Elimination (RFE)

This method is to use a model (XGBoost in this case) to recursively remove least-important features. The `Target` column is reintroduced as a judging criterion to guide feature selection.

Below are the 5 least important features identified in the RFE results:

`TechCrunch_Emotion_Intensity` `Bloomberg_Sentiment_Polarity` `CNBC_Sentiment_Polarity` `MarketWatch_Sentiment_Polarity` `TechCrunch_Sentiment_Polarity`

B2.2.2 Recursive Feature Elimination with Cross-Validation (RFECV)

RFECV is similar to RFE but adds cross-validation to improve feature selection.

RFECV suggests removing 21 features, including 5 features that were also flagged for removal by RFE. Given the current progress, this approach feels overly aggressive. To maintain a conservative strategy, I will begin by eliminating only the 5 overlapping features.

Below are the retained features after applying RFE and RFECV, leaving us with a total of 28 features:

`Volume` `Daily_Return` `ATR` `BBANDS_U` `RSI` `MACD` `H-L` `O-C` `FEDFUNDS`

`Bloomberg_Relevancy_Score` `Bloomberg_Sentiment_Score` `Bloomberg_Emotion` `Bloomberg_Emotion_Intensity` `Bloomberg_Keyword_Impact`

`CNBC_Relevancy_Score` `CNBC_Sentiment_Score` `CNBC_Emotion` `CNBC_Emotion_Intensity` `CNBC_Keyword_Impact`

`MarketWatch_Relevancy_Score` `MarketWatch_Sentiment_Score` `MarketWatch_Emotion` `MarketWatch_Emotion_Intensity` `MarketWatch_Keyword_Impact`

`TechCrunch_Relevancy_Score` `TechCrunch_Sentiment_Score` `TechCrunch_Emotion` `TechCrunch_Keyword_Impact`

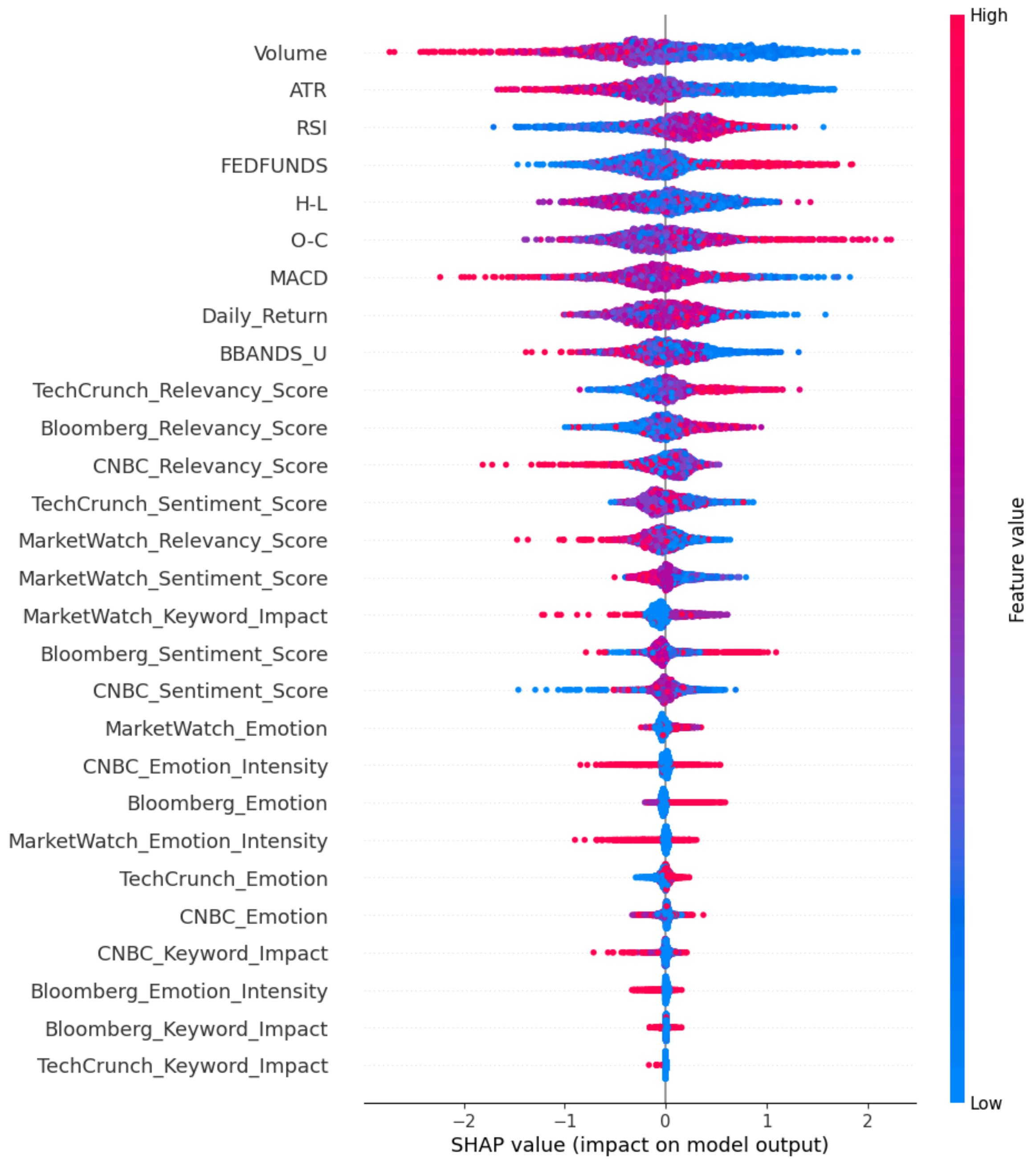
Back to Content

B2.3 Emebeded Methods

Embedded methods allow us to directly utilize the final models to evaluate feature importance. Specifically, we will apply the SHAP method for a more comprehensive assessment.

SHAP quantifies feature importance by assessing each feature’s impact on model predictions, measuring how much it positively or negatively influences outcomes. The `Target` column will also be used as a judging criterion to guide feature selection.

After deriving the SHAP values for the features, I created two visualizations. The first illustrates how feature values influence the model's output:

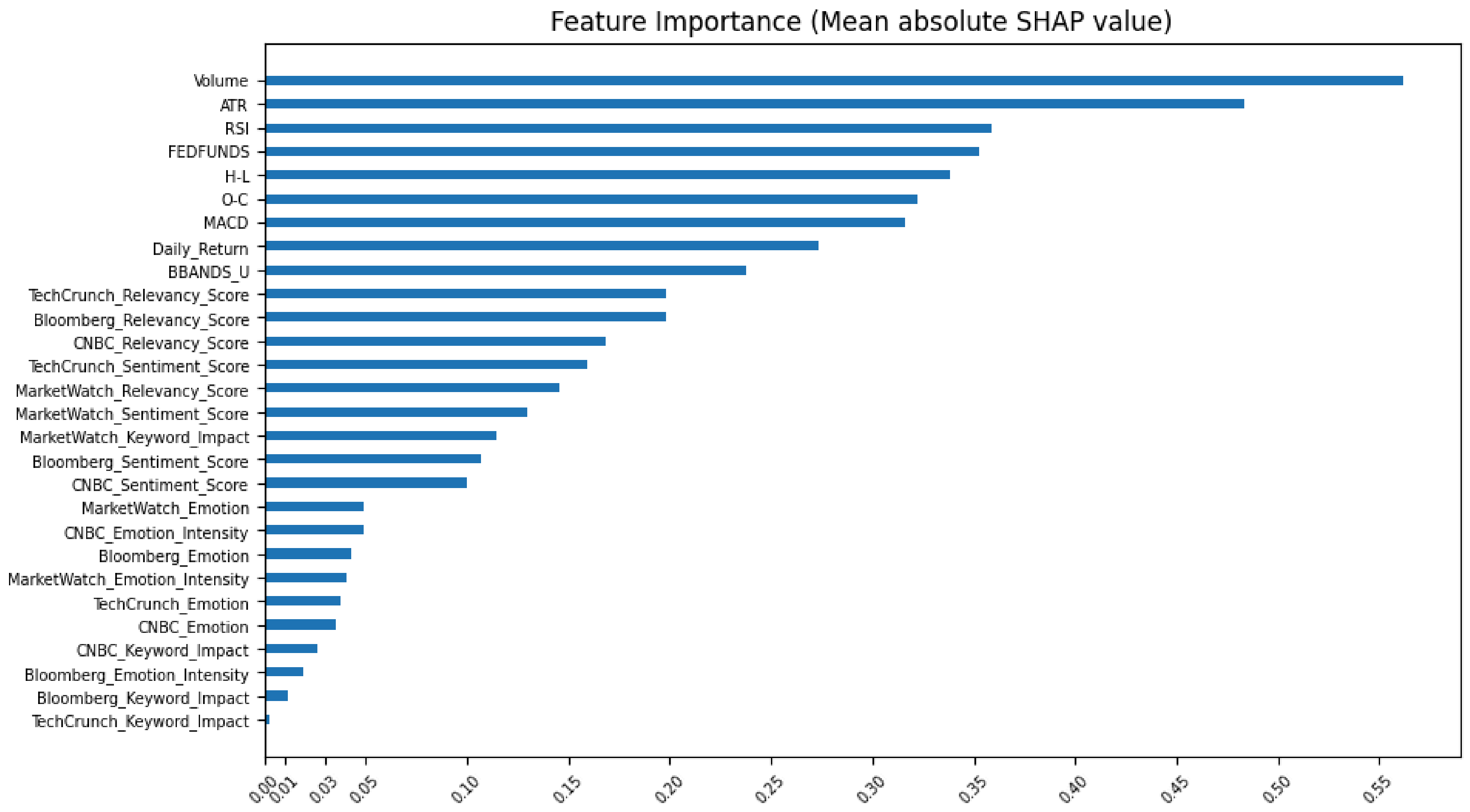


The plot above ranks feature importances from high to low, as evaluated using XGBoost.

Each dot represents a single day's data entry, with its horizontal position indicating whether the feature's value pushes the prediction toward 1 (right) or 0 (left). For example:

- A red dot for `BBANDS_U` on the left suggests that a higher lower Bollinger Band value contributes to a prediction of 0.
- A blue dot for `Daily_Return` on the right indicates that a lower recent return pushes the prediction toward 1.

One the other hand, we can directly have the absolute mean of SHAP values for each features. This will give us a more intuitive feeling of their impacts, regardless of positive or negative, to the prediction. Therefore, the second visualization is about the mean absolute SHAP value.



As shown above, the mean absolute SHAP values range from above 0.01 to approximately 0.55.

Examining the features with the lowest mean absolute SHAP values with consideration of the plot of impact on model output:

- `TechCrunch_Keyword_Impact` has the lowest value — even significantly lower than the usual threshold 0.01, making it a strong candidate for removal.
- `Bloomberg_Keyword_Impact` and `Bloomberg_Emotion_Intensity` rank as the second and third lowest, exhibiting impact patterns similar to `CNBC_Emotion_Intensity` and `MarketWatch_Emotion_Intensity` respectively, suggesting they may need to be removed.
- `CNBC_Keyword_Impact`, the fourth lowest, behaves somehow also similarly to `MarketWatch_Emotion_Intensity`. It could be removed as well.
- Next one, `CNBC_Emotion` looks slightly unique and its mean absolute SHAP value is already well above 0.03, so I will take a stop here.

While further feature evaluation is possible, I will adopt a conservative approach and limit removals to 4 features at this stage, setting 0.03 as a reasonable cutoff threshold. This decision is subjective, aimed at maintaining a balanced refinement process.

Eliminating the 4 features identified using SHAP concludes the feature selection process.

After applying all selection methods, 24 features have been finalized, listed below:

`Volume` `Daily_Return` `ATR` `BBANDS_U` `RSI` `MACD` `H-L` `O-C` `FEDFUNDS`

`Bloomberg_Relevancy_Score` `Bloomberg_Sentiment_Score` `Bloomberg_Emotion`

`CNBC_Relevancy_Score` `CNBC_Sentiment_Score` `CNBC_Emotion` `CNBC_Emotion_Intensity`

`MarketWatch_Relevancy_Score` `MarketWatch_Sentiment_Score` `MarketWatch_Emotion`
`MarketWatch_Emotion_Intensity` `MarketWatch_Keyword_Impact`

`TechCrunch_Relevancy_Score` `TechCrunch_Sentiment_Score` `TechCrunch_Emotion`

Back to Content

C. Model Building, Tuning and Evaluation

Predicting Positive Market Moves Using Gradient Boosting.
(a) Build a model to predict positive market moves (uptrend) using the feature subset derived above.
(b) Tune the hyperparameters of the estimator to obtain an optimal model.
(c) Evaluate the model's prediction quality using the area under the receiver operating characteristic(ROC) curve, confusion matrix, and classification report.

Answer:

In this section, I will use 3 common Gradien Boosting algorithms: **XGBoost, AdaBoost and LightGBM**, to build models and compare their performance based on the evaluation.

The 24 selected features will serve as the independent variables (X) and will be scaled by removing the mean and normalizing to unit variance. This ensures that all features contribute equally to the model, preventing high-magnitude variables from dominating.

The dependent variable (y) is the `Target` column, which consists of Class 0 and Class 1.

The dataset will be split into training and test sets using an 80-20 ratio.

Due to the imbalance between Class 0 and Class 1, the weights of the dependent variable in the training and test sets will be assigned accordingly to account for this distribution.

Back to Content

C1. XGBoost

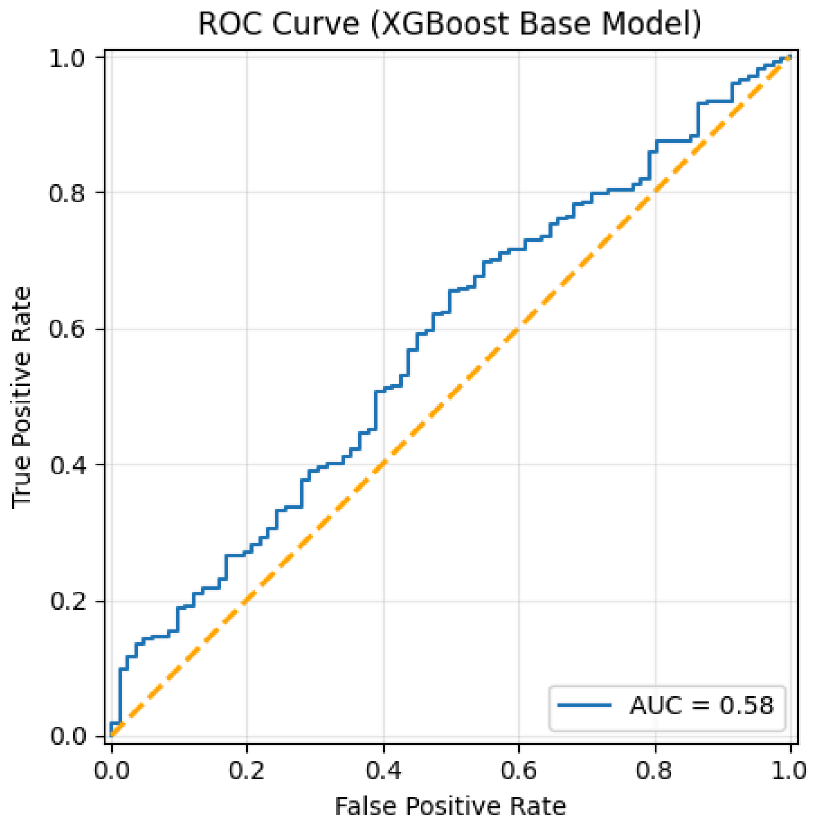
C1.1 Build A Model (XGBoost)

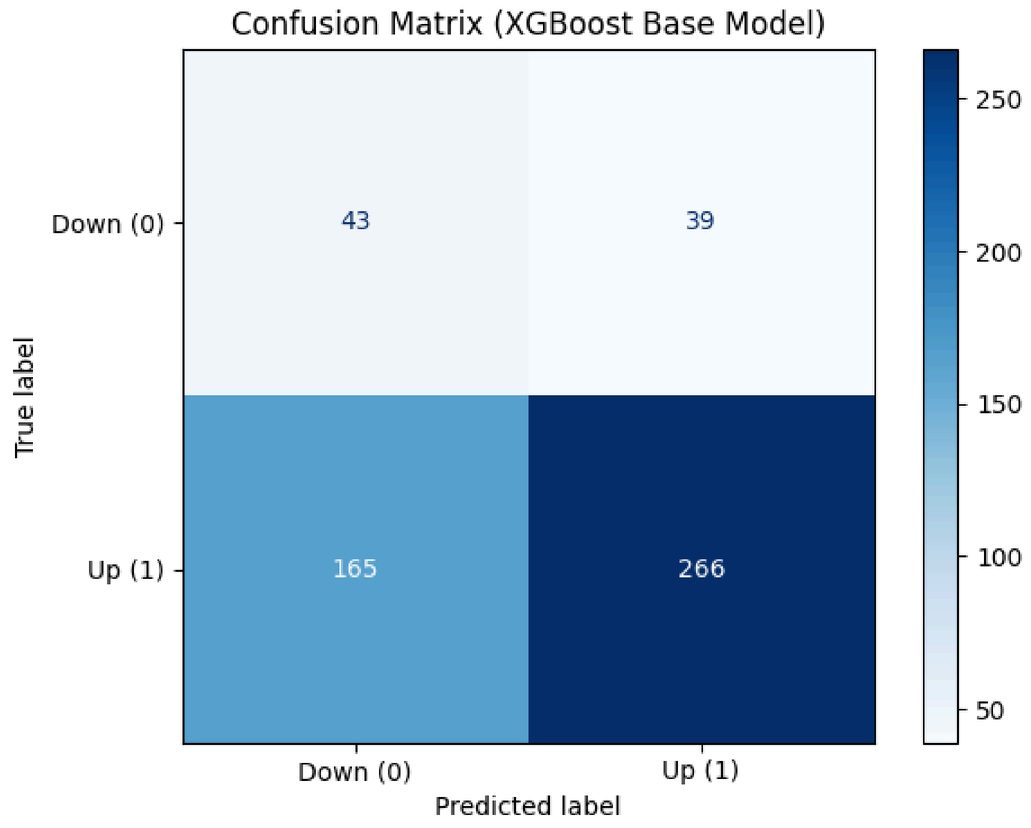
The base model was built using the following regular parameters as the default configuration:

- `objective`: 'binary:logistic'

- `eval_metric` : 'auc'
- `max_depth` : 3
- `eta` :0.1
- `subsample` : 0.6
- `colsample_bytree` : 0.6
- `min_child_weight` : 1
- `gamma` : 0
- `num_boost_round` : 100
- `early_stopping_rounds` : 30

Result - ROC AUC, confusion matrix, classification report, and training test accuracy (XGBoost Base Model)





Classification Report (XGBoost Base Model):

	Precision	Recall	F1-Score	Support
0	0.21	0.52	0.30	82
1	0.87	0.62	0.72	431
Accuracy			0.60	513
Macro Avg	0.54	0.57	0.51	513
Weighted Avg	0.77	0.60	0.65	513

Accuracy Performance (XGBoost Base Model):

Metric	Value
Training Accuracy	0.7260
Test Accuracy	0.6023
Training-Test Difference	0.1236

Result review (XGBoost Base Model)

Using some regular parameters, we established our initial XGBoost model for this project.

- The AUC value of 0.58 suggests limited predictive separation, meaning the model struggles to distinguish between classes effectively.
- Class 1 (positive market moves) demonstrates strong precision (0.87), confirming reliable predictions.
- Class 0 (negative market moves) has significantly lower precision (0.21), indicating a need for improvement in identifying negative trends.
- The training accuracy (0.7260) exceeds the test accuracy (0.6023) by more than 0.1, signaling over-fitting, as the model performs well on training data but loses accuracy on unseen data.

In summary, while this base model provides some insights, it remains far from optimal. To enhance performance, we will proceed with fine-tuning key parameters.

Back to Content

C1.2 Tune the Hyperparameters (XGBoost)

Hyperparameters are external model parameters that are not learned directly during training. Instead, they are specified when initializing the classifier. Any parameter defined at the estimator's construction can be fine-tuned in this manner to improve model performance.

Below are the hyperparameters we are going to explore and fine-tune:

- `max_depth` : maximum depth of a tree, range: $[0, \infty)$
- `eta/learning rate` : step size shrinkage used in update to prevents overfitting, range: $[0, 1]$
- `subsample` : ratio of the training instances, range: $(0, 1]$
- `colsample_bytree` : percentage of features used per tree. High value can lead to overfitting, range: $(0, 1]$
- `min_child_weight` : minimum sum of instance weight needed in a child, range: $[0, \infty)$
- `gamma` : minimum loss reduction required to make a further partition on a leaf node of the tree, range: $[0, \infty)$

Methods of fine-tuning
I explored several methods to deliver the fine-tuning work:

Methods	Pros	Cons
RandomizedSearchCV	- Moderately acceptable speed - Cross-validation prevents overfitting	- Does not cover all combinations
GridSearchCV	- Exhaustively searches all hyperparameter combinations - Cross-validation prevents overfitting	- Computationally expensive
Customized Loop	- Flexible—can be designed to meet specific demands - Faster since validation is not included	- Requires manual input for multiple parameters - Must manually check overfitting after results are out

While **RandomizedSearchCV** and **GridSearchCV** are valuable tools for hyperparameter tuning, I encountered some challenges using them.

RandomizedSearchCV does **not** test all possible parameter combinations, and there were situations when I mannualy tried out a better combination. Also, since it selects combinations randomly, the results can sometimes be unstable.

GridSearchCV works well for a small number of combinations, but testing 6 parameters with 3-5 options each makes the process **very slow**. While breaking down the parameter space into smaller search chunks was an option, I opted to write custom loop-based tuning instead.

I implemented a **simple for-loop** to explore all 6 parameters with 3-5 possible values each:

- `max_depth` , `eta` , and `subsample` are iterated through predefined lists.
- `colsample_bytree` , `min_child_weight` , and `gamma` are manually adjusted to balance speed and flexibility.
- The manually adjusted parameters typically require less variation than the looped ones.
- The loop selects the best AUC value across all tested combinations without cross-validation.

- However, to mitigate overfitting risks, I will manually review results to ensure no over-fitting.

Below is the range and selection method (loop-based or manual) for each parameter used in this fine-tuning process:

Parameter	Range	Method
max_depth	3, 4, 5	loop
eta	0.01, 0.03, 0.05, 0.1, 0.12	loop
subsample	0.6, 0.7, 0.8, 0.9	loop
colsample_bytree	0.6, 0.65, 0.7, 0.75, 0.8	manual
min_child_weight	1, 2, 3	manual
gamma	0, 0.1, 0.2	manual

The fine-tuning process identified the best parameter combination as follows, achieving an AUC of 0.67:

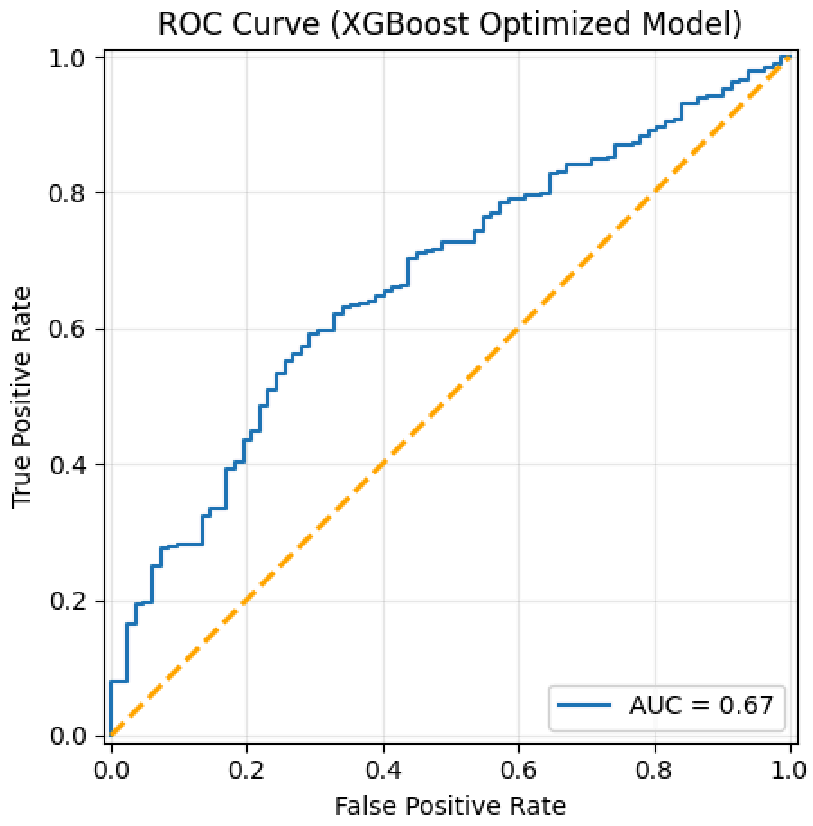
- `max_depth` : 4
- `eta/learning rate` : 0.1
- `subsample` : 0.7
- `colsample_bytree` : 0.65
- `min_child_weight` : 1
- `gamma` : 0

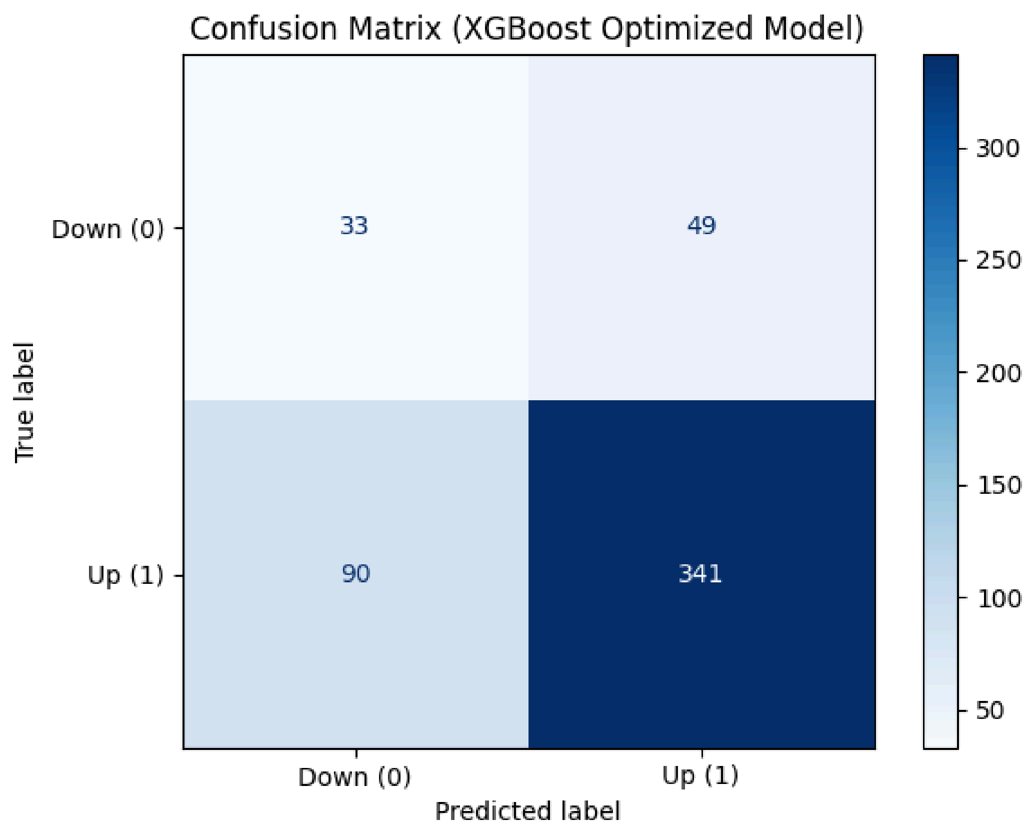
Back to Content

C1.3 Evaluate the Optimized Model's Prediction Quality (XGBoost)

Let's apply this parameter combination, build the optimized XGBoost model, and evaluate its performance.

Result - ROC AUC, confusion matrix, classification report, training test accuracy, and feature importance (XGBoost Optimized Model)


ROC Curve (XGBoost Optimized Model)
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0
AUC = 0.67

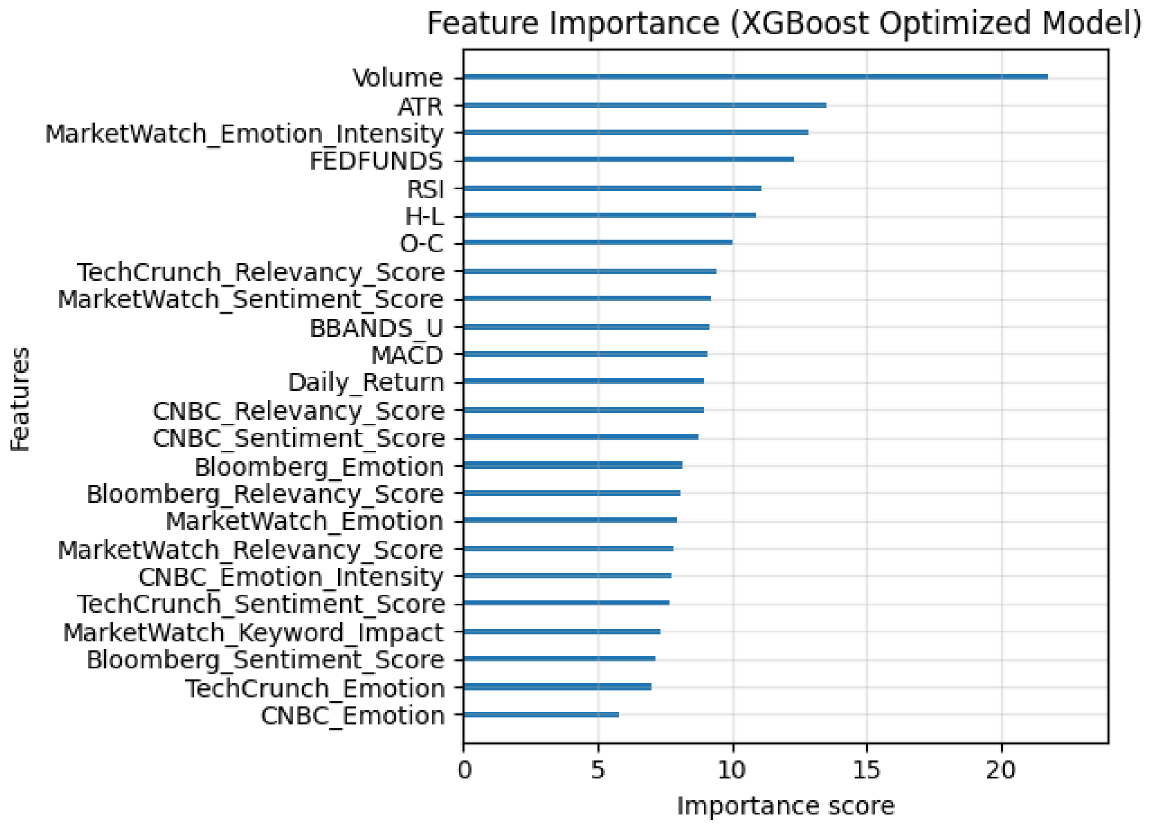


Classification Report (XGBoost Optimized Model):

	Precision	Recall	F1-Score	Support
0	0.27	0.40	0.32	82
1	0.87	0.79	0.83	431
Accuracy			0.73	513
Macro Avg	0.57	0.60	0.58	513
Weighted Avg	0.78	0.73	0.75	513

Accuracy Performance (XGBoost Optimized Model):

Metric	Value
Training Accuracy	0.7840
Test Accuracy	0.7290
Training-Test Difference	0.0550



Result review (XGBoost Optimized Model)

With the fine-tuned parameters, we established the optimized XGBoost model for this project.

- AUC improved to 0.67 (previously 0.58), enhancing the model’s ability to separate positive and negative movements more effectively.
- Class 1 (positive market moves) maintains high precision (0.87), while recall has increased to 0.79 (previously 0.62), demonstrating an improved ability to detect positive movements without missing too many instances.

- Class 0 (negative market moves) sees a precision boost to 0.27 (previously 0.21), though it's still not ideal.
- Training accuracy (0.7840) vs. test accuracy (0.7290) shows a difference of 0.0550, indicating better generalization and reduced overfitting compared to the initial version.
- The top influential features span multiple sources: trading data(Volume, ATR), news sentiment (MarketWatch_Emotion_Intensity), and macro emonomy indicator (FEDFUNDS). This diverse feature set confirms effective feature selection.

In summary, this fine-tuning process has meaningfully improved the model, making it more practical for predictive use.

Back to Content

C2. AdaBoost

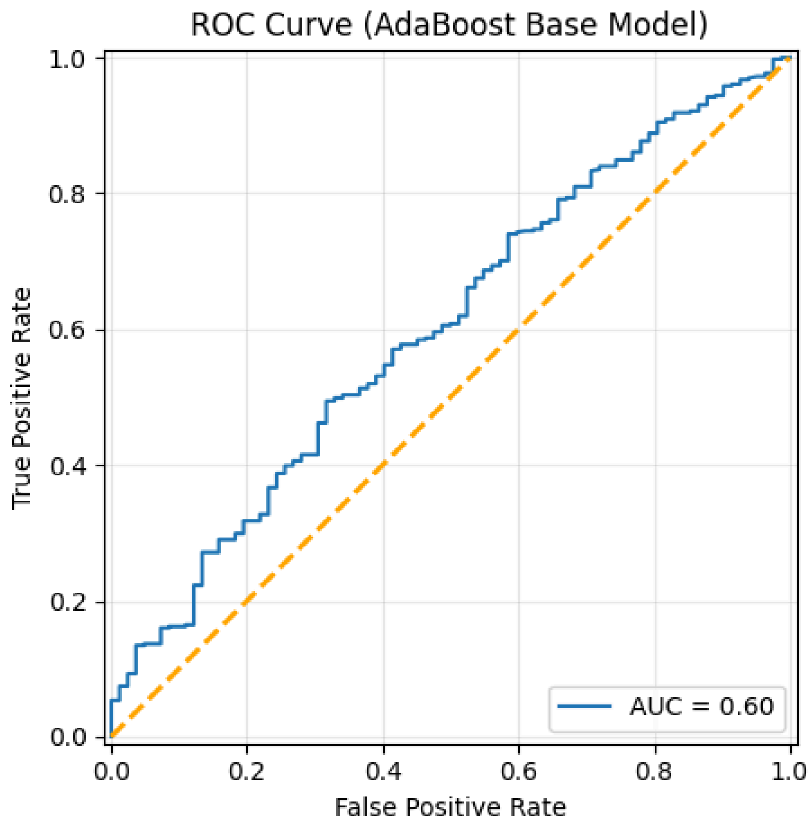
C2.1 Build A Model (AdaBoost)

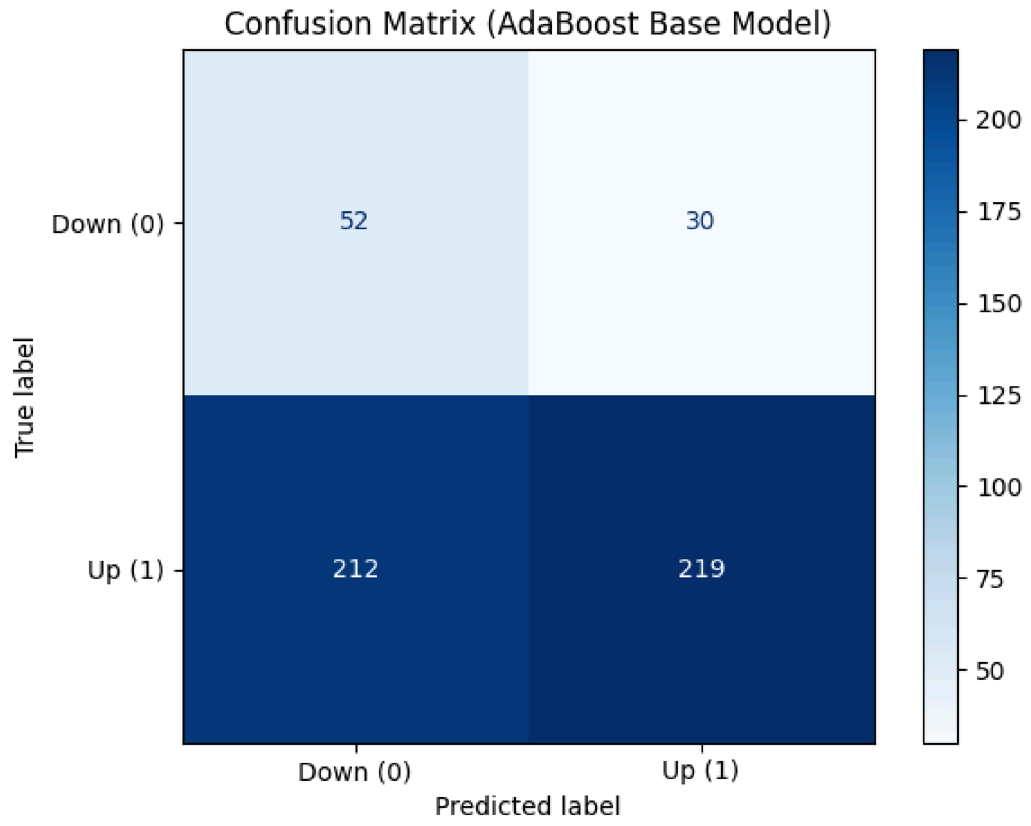
We will use Decision Tree as the base estimator for AdaBoost, with the initial model constructed using the following regular parameters as its default configuration:

- `max_depth` : 4
- `n_estimators` : 100
- `learning_rate` :0.1
- `random_state` : 42

Note: `max_depth` is a Decision Tree parameter, while `random_state` ensures consistent output but does not impact performance.

Result - ROC AUC, confusion matrix, classification report, and training test accuracy (AdaBoost Base Model)


ROC Curve (AdaBoost Base Model)
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0
AUC = 0.60



Classification Report (AdaBoost Base Model):

	Precision	Recall	F1-Score	Support
0	0.20	0.63	0.30	82
1	0.88	0.51	0.64	431
Accuracy			0.53	513
Macro Avg	0.54	0.57	0.47	513
Weighted Avg	0.77	0.33	0.59	513

Accuracy Performance (AdaBoost Base Model):

Metric	Value
Training Accuracy	0.7138
Test Accuracy	0.5283
Training-Test Difference	0.1855

Result review (AdaBoost Base Model)

Using some regular parameters, we established our initial AdaBoost model for this project.

- The AUC value of 0.60 suggests the model has moderate challenge to differentiate between positive and negative movements effectively.
- While the precision for class 1 (positive market moves) is strong at 0.88, recall (0.51) means the model misses nearly half of positive moves.
- The precision for class 0 (negative market moves) remains low at 0.20, meaning many false positives for negative moves..
- The gap training accuracy (0.7138) and the test accuracy (0.5283) is almost 0.2, away above 0.1, showing significant over-fitting and making generalization weak on unseen data.

In summary, while this base model offers some insights, it is still not good enough. To enhance performance, we will proceed with fine-tuning key parameters.

Back to Content

C2.2 Tune the Hyperparameters (AdaBoost)

AdaBoost has fewer tunable parameters compared to XGBoost. Here are some key ones we will explore with a customized loop:

- `max_depth` : parameter of the Decision Tree as the base estimator, it is the maximum depth of a tree, range: $[0, \infty)$
- `learning rate` : weight applied to each classifier at each boosting iteration, range: $(0, \infty)$
- `n_estimator` : the maximum number of estimators at which boosting is terminated, range: $[1, \infty)$

Below is the range and selection method (loop-based or manual) for each parameter used in this fine-tuning process:

Parameter	Range	Method
max_depth	3, 4, 5	loop

Parameter	Range	Method
learning_rate	0.01, 0.05, 0.15, 0.2	loop
n_estimator	100, 200, 300, 400	loop

The fine-tuning process identified the best parameter combination as follows, achieving an AUC of 0.62:

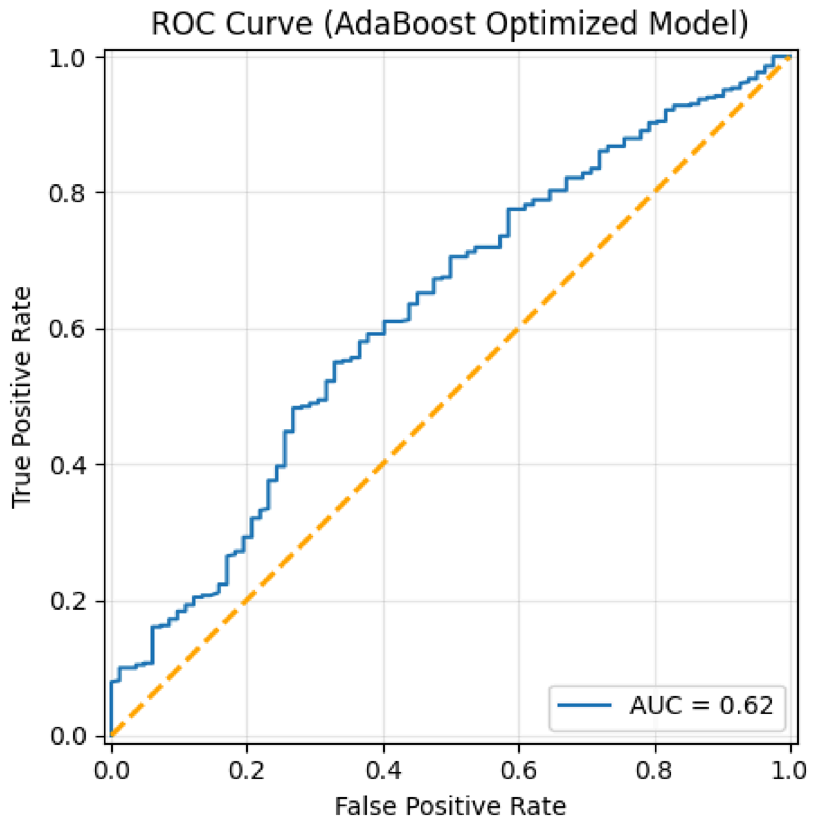
- `max_depth` : 4
- `learning_rate` : 0.2
- `n_estimator` : 200

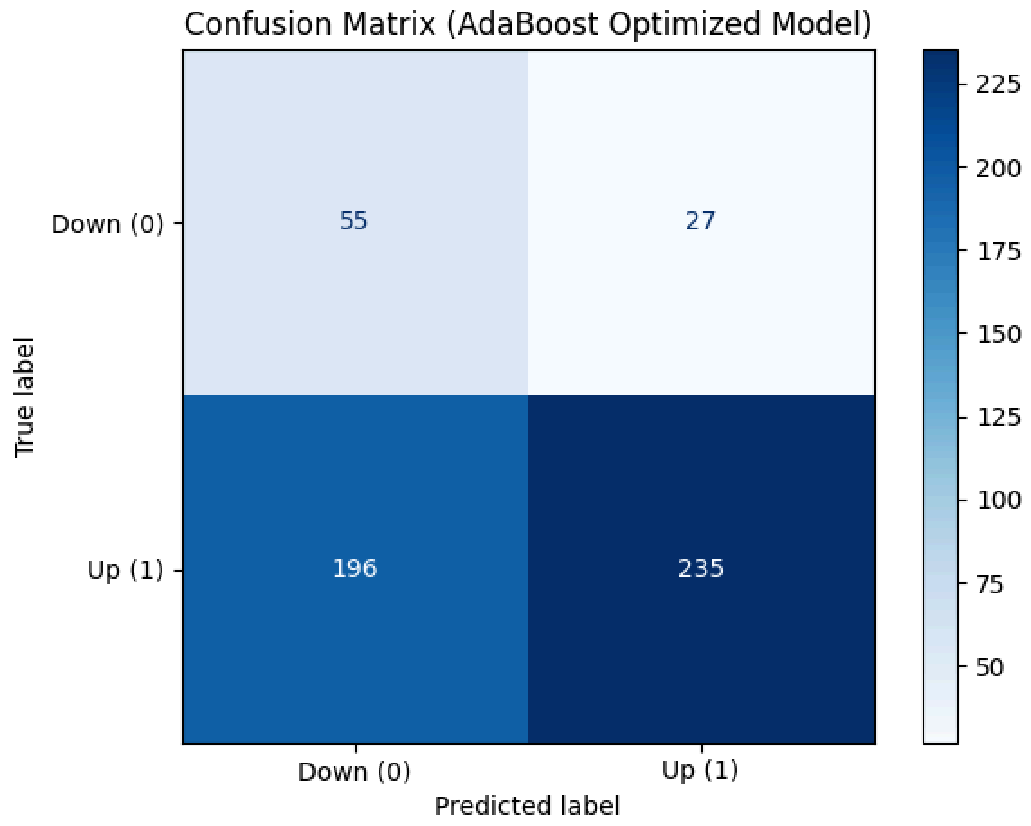
Back to Content

C2.3 Evaluate the Optimized Model's Prediction Quality (AdaBoost)

Let's apply this parameter combination, build the optimized AdaBoost model, and evaluate its performance.

Result - ROC AUC, confusion matrix, classification report, training test accuracy, and feature importance (AdaBoost Optimized Model)


ROC Curve (AdaBoost Optimized Model)
True Positive Rate
False Positive Rate
AUC = 0.62
0.0
0.2
0.4
0.6
0.8
1.0

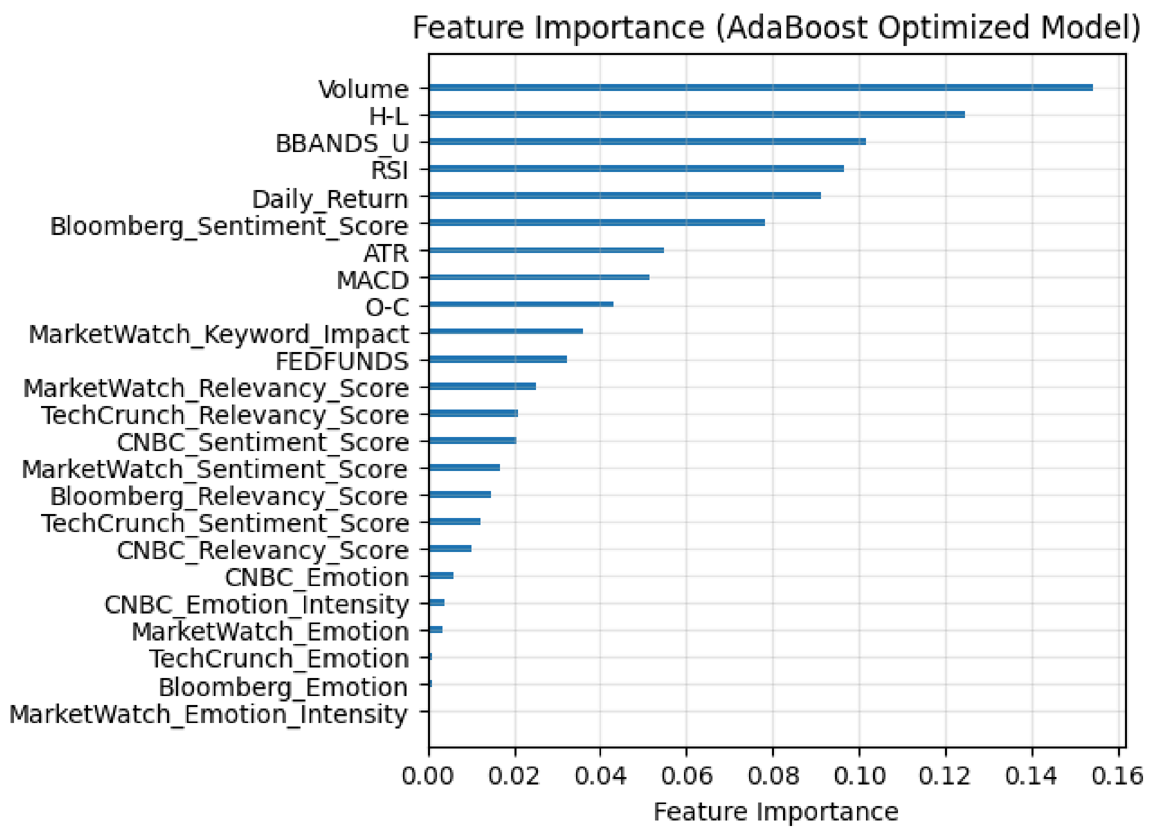


Classification Report (AdaBoost Optimized Model):

	Precision	**Recall**	**F1-Score**	**Support**
0	0.22	0.67	0.33	82
1	0.90	0.55	0.68	431
Accuracy			0.57	513
Macro Avg	0.56	0.61	0.50	513
Weighted Avg	0.79	0.57	0.62	513

Accuracy Performance (AdaBoost Optimized Model):

Metric	Value
Training Accuracy	0.7474
Test Accuracy	0.5653
Training-Test Difference	0.1821



Result review (AdaBoost Optimized Model)

With the fine-tuned parameters, we established the optimized AdaBoost model for this project.

- AUC is improved to 0.62 (previously 0.60), showing a slight enhancement in predictive separation.
- Class 1 (positive market moves) shows slightly improved precision (0.90) compared to 0.88, while recall has increased to 0.55 (previously 0.51), indicating better detection of positive movements.

- Class 0 (negative market moves) continues to struggle with low precision (0.22), though recall (0.67) has improved from 0.63, allowing the model to identify more negative moves despite imbalanced predictions.
- The gap between training accuracy (0.7474) and test accuracy (0.5567) remains high, indicating persistent overfitting, which overshadows the performance gains.
- The most influential features primarily originate from trading data (Volume, H-L, BBANDS_U), suggesting that the model may not be leveraging diverse feature sources as effectively as the XGBoost optimized model,

In summary, while the fine-tuned AdaBoost model shows moderate improvements, the significant overfitting risk limits its practical usability.

Back to Content

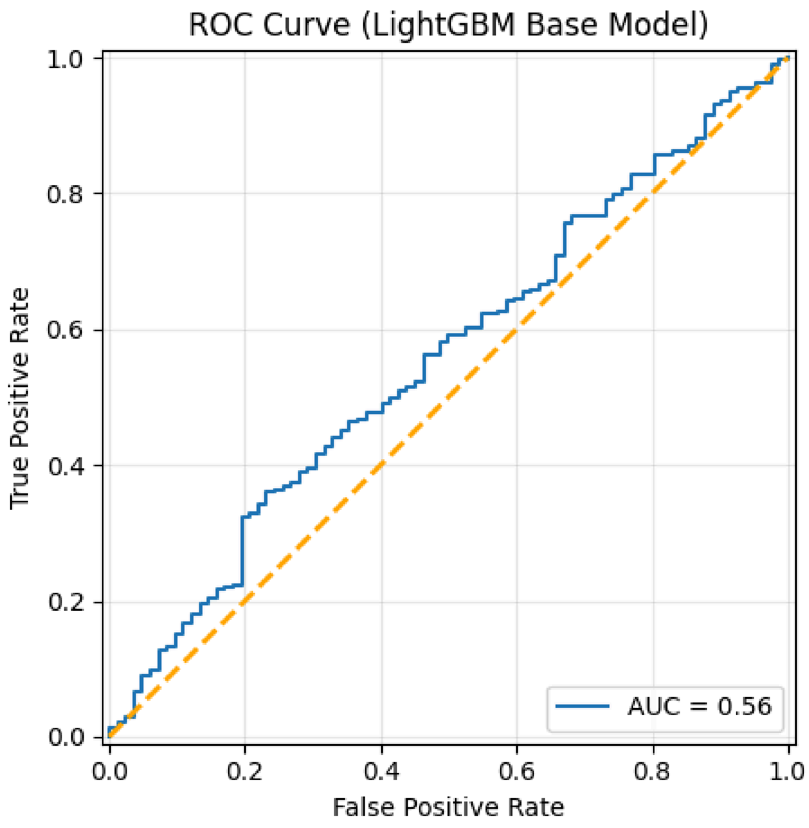
C3. LightGBM

C3.1 Build a model (LightGBM)

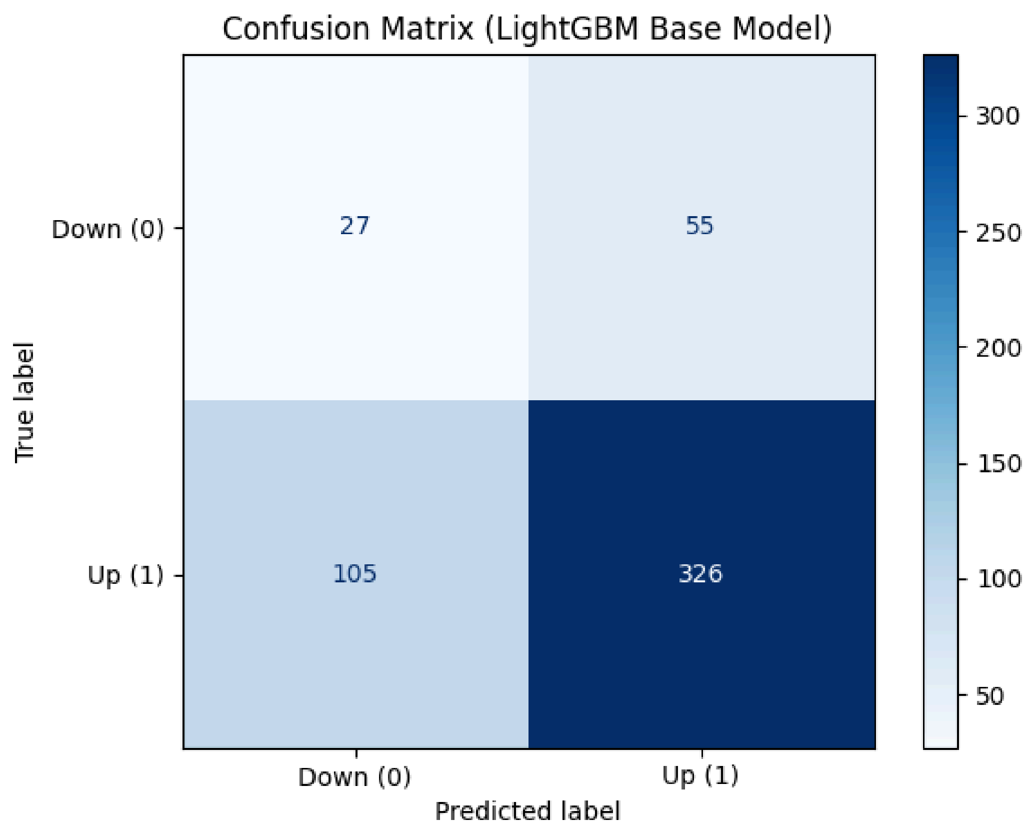
The base model was built using the following regular parameters as the default configuration:

- `objective` : 'binary:logistic'
- `metric` : 'auc'
- `max_depth` : 3
- `learning_rate` :0.1
- `subsample` : 0.7
- `colsample_bytree` : 0.6
- `min_child_weight` : 1
- `lambda_l1` : 0
- `num_boost_round` : 100

Result - ROC AUC, confusion matrix, classification report, and training test accuracy (LightGBM Base Model)



ROC Curve (LightGBM Base Model)
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0
AUC = 0.56



Classification Report (LightGBM Base Model):

	Precision	Recall	F1-Score	Support
0	0.20	0.33	0.25	82
1	0.86	0.76	0.80	431
Accuracy			0.69	513
Macro Avg	0.53	0.54	0.53	513
Weighted Avg	0.75	0.69	0.71	513

Accuracy Performance (LightGBM Base Model):

Metric	Value
Training Accuracy	0.8489
Test Accuracy	0.6881
Training-Test Difference	0.1607

Result review (LightGBM Base Model)

Using some regular parameters, we established our initial LightGBM model for this project.

- The AUC value of 0.56 suggests quite limited discriminatory power, meaning the model struggles to separate classes effectively.
- Class 1 (positive market moves) demonstrates strong precision (0.86) and decent recall (0.76), effectively capturing a significant portion of positive movements.
- Class 0 (negative market moves) remains challenging, with low precision (0.20) and recall (0.33), indicating a need for improvement in detecting negative trends.
- The training accuracy (0.8489) exceeds the test accuracy (0.6881) by around 0.16, pointing to moderate overfitting, which could reduce performance on unseen data.

In summary, this base model provides a foundation, it is far from optimal. To enhance its performance, we will proceed with fine-tuning key parameters.

Back to Content

C3.2 Tune the Hyperparameters (LightGBM)

Here are some key parameters of LightGBM we will explore with a customized loop:

- `max_depth` : maximum depth of a tree, range: $[0, \infty)$
- `learning_rate` : Step size shrinkage to prevent overfitting by controlling how quickly the model updates weights, range: $[0, 1]$
- `subsample` : ratio of the training instances, range: $(0, 1]$
- `colsample_bytree` : percentage of features used per tree. High value can lead to overfitting, range: $(0, 1]$
- `min_child_weight` : minimum sum of instance weight needed in a child, range: $[0, \infty)$

`max_depth` , `learning_rate` and `subsample` will be looped with their respective lists, while `colsample_bytree` and `min_child_weight` will be manually adjusted.

Below is the range and selection method (loop-based or manual) for each parameter used in this fine-tuning process:

Parameter	Range	Method
max_depth	3, 4, 5	loop
learning_rate	0.01, 0.03, 0.05, 0.1, 0.15	loop
subsample	0.6, 0.7, 0.8, 0.9	loop
colsample_bytree	0.4, 0.5, 0.6, 0.7	manual
min_child_weight	1, 2, 3	manual

The fine-tuning process identified the best parameter combination as follows, achieving an AUC of 0.60:

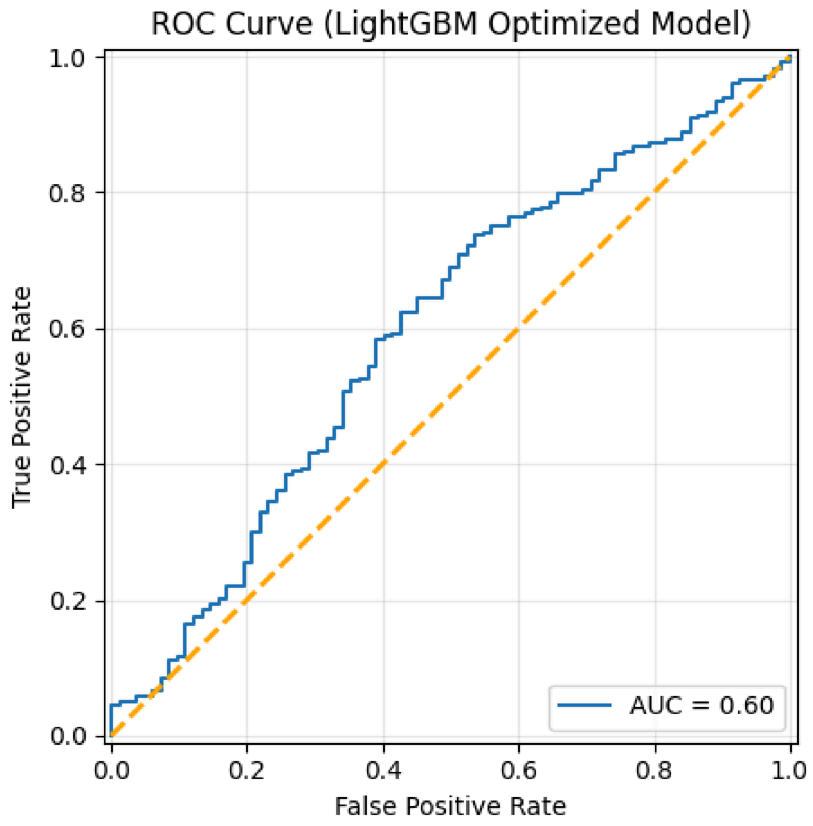
- `max_depth` : 3
- `eta/learning rate` : 0.15
- `subsample` : 0.6
- `colsample_bytree` : 0.5
- `min_child_weight` : 1

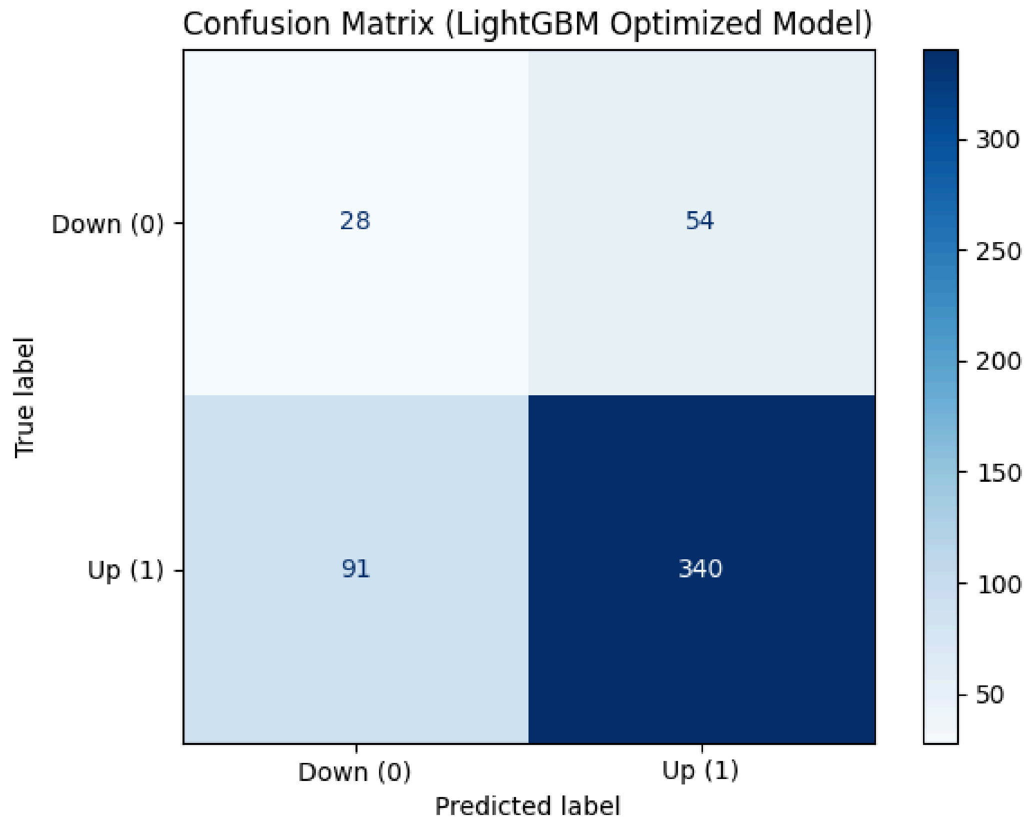
Back to Content

C3.3 Evaluate the Optimized Model's Prediction Quality (LightGBM)

Let's apply this parameter combination, build the optimized XGBoost model, and evaluate its performance.

Result - ROC AUC, confusion matrix, classification report, training test accuracy, and feature importance (LightGBM Optimized Model)


ROC Curve (LightGBM Optimized Model)
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0
AUC = 0.60

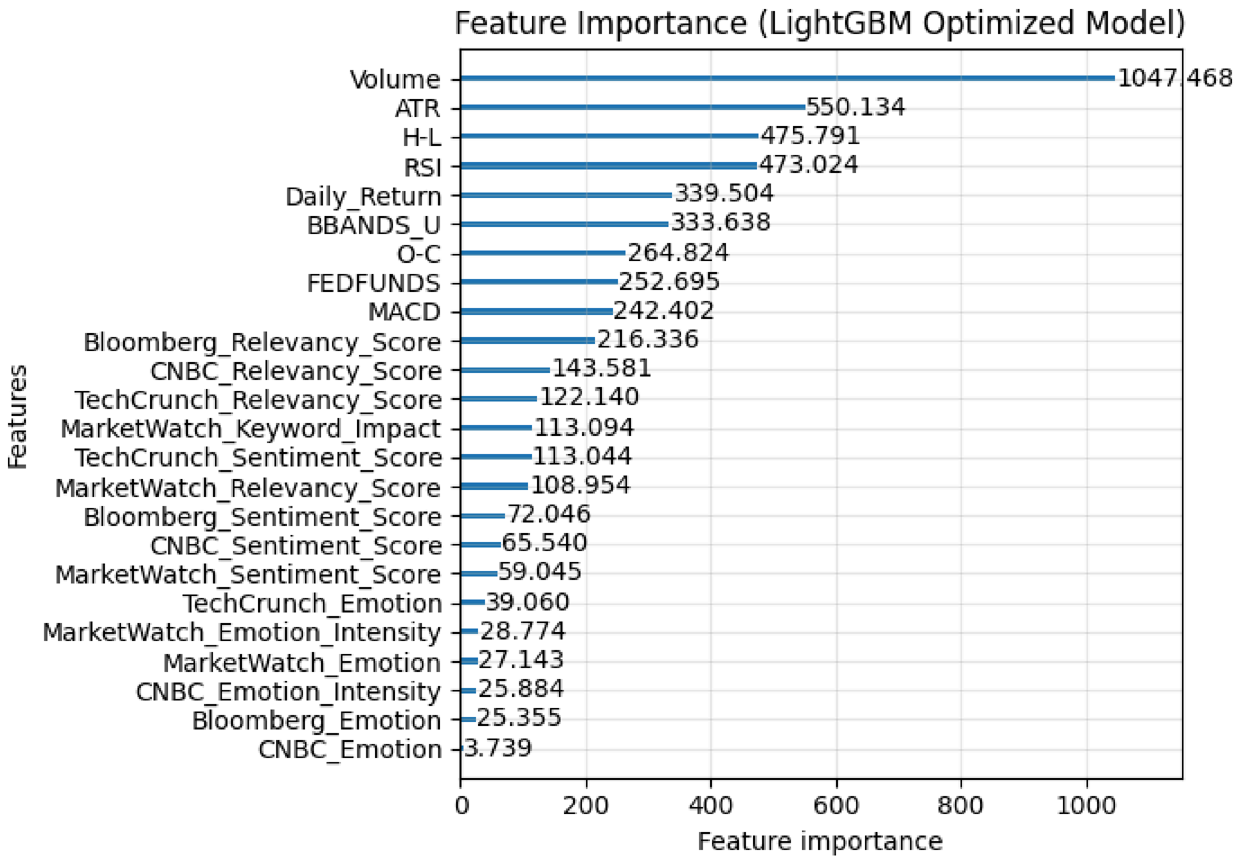


Classification Report (LightGBM Optimized Model):

	Precision	Recall	F1-Score	Support
0	0.24	0.34	0.28	82
1	0.86	0.79	0.82	431
Accuracy			0.72	513
Macro Avg	0.55	0.57	0.55	513
Weighted Avg	0.76	0.72	0.74	513

Accuracy Performance (LightGBM Optimized Model):

Metric	Value
Training Accuracy	0.8259
Test Accuracy	0.7173
Training-Test Difference	0.1086



Result review (LightGBM Optimized Model)

With the fine-tuned parameters, we established the optimized LightGBM model for this project.

- AUC is improved to 0.60 (previously 0.56), indicating better predictive separation, but still well below the 0.70+ threshold for robust classification.

- Class 1 (positive market moves) maintains strong predictive reliability, with precision at 0.86 and an improved recall of 0.79 (previously 0.76), allowing the model to capture more true positives.
- Class 0 (negative market moves) sees a slight improvement in precision (0.24, up from 0.20) and recall (0.34, up from 0.33), though its performance remains suboptimal.
- Training accuracy (0.8259) vs. test accuracy (0.7173) shows a gap of ~0.11, reducing over-fitting compared to the base model, but not enough to fully mitigate the concern of over-fitting.
- The model's most influential features primarily come from trading data (Volume, ATR, H-L), similar to the AdaBoost optimized model, suggesting that the model may not be leveraging diverse feature sources as effectively as the XGBoost optimized model.

In summary, while the fine-tuning has improved performance, the model still exhibits weaknesses that limit its practical usability.

Back to Content

C4. Conclusion of Models

Here is a camparision table of the results from all 3 optimized models:

Evaluation Metric	XGBoost	AdaBoost	LightGBM
AUC	**0.67**	0.62	0.60
Class 1 Precision	0.87	**0.90**	0.86
Class 1 Recall	**0.79**	0.55	**0.79**
Class 1 F1-score	**0.83**	0.68	0.82
Class 0 Precision	**0.27**	0.22	0.24
Class 0 Recall	**0.40**	0.67	0.34
Class 0 F1-score	0.32	**0.33**	0.28
Training Accuracy	0.7840	0.7474	**0.8259**
Test Accuracy	**0.7290**	0.5653	0.7173
Train-Test Accuracy Gap	**0.0550**	0.1821	0.1086

Among the 3 models, XGBoost emerges as the strongest performer due to several key advantages:

- Highest AUC (0.67), making it the only model approaching the 0.70 threshold, which indicates superior class separation.
- Balanced precision-recall trade-off for both Class 1 and Class 0, ensuring more reliable predictions.
- Highest test accuracy (0.7290) with minimal overfitting (gap of 0.0550), demonstrating strong generalization to unseen data.
- In contrast, AdaBoost and LightGBM performs worse but with higher overfitting, leading to lower overall stability and predictive reliability.

In conclusion, the XGBoost optimized model proves to be the most consistent and effective choice for QQQ ETF movement prediction.
Next, I will integrate it into a trading strategy to evaluate its practical performance in real-world conditions.

Back to Content

Trading Strategy Practice with Backtesting

T1. Profit Analysis

To assess the effectiveness of the XGBoost optimized model, we will conduct a simple backtesting exercise using the 10-years QQQ price data. The trading rules are straightforward:

- **Buy Signal (Predicted = 1):** Purchase one share of QQQ ETF at the day's adjusted closing price.
- **Sell on the Next Day:** If a purchase occurs, the position will be sold at the next day's adjusted closing price.
- **No Trade (Predicted = 0):** No transaction is executed on this day related to this predicition.
- **No Transaction Costs or Friction Included:** This backtest assumes a cost-free trading environment for simplicity.

Trades following these rules will be referred to as the **XGBoost Strategy** in the subsequent analysis. Now, let's put it to the test.

First, use the XGBoost optimized model to predict QQQ’s next-day price movements over the past 10 years. I constructed a new dataset incorporating `Adj Close`, ` Daily_Return`, and the just-now generated `XGBoost_Prediction data`.

Then, to accurately track daily trading activities under the XGBoost Strategy over the past 10 years, create the following columns:

- `Buy_Price`: Records the amount spent to purchase one share of QQQ ETF on a given day, if any.
- `Sell_Price`: Captures the amount received from selling the share bought on the previous day, if any.

- `Profit` : Computes the net earnings from each trade by subtracting the previous day's Buy Price from the current day's Sell Price, representing the money made from this round of transaction.

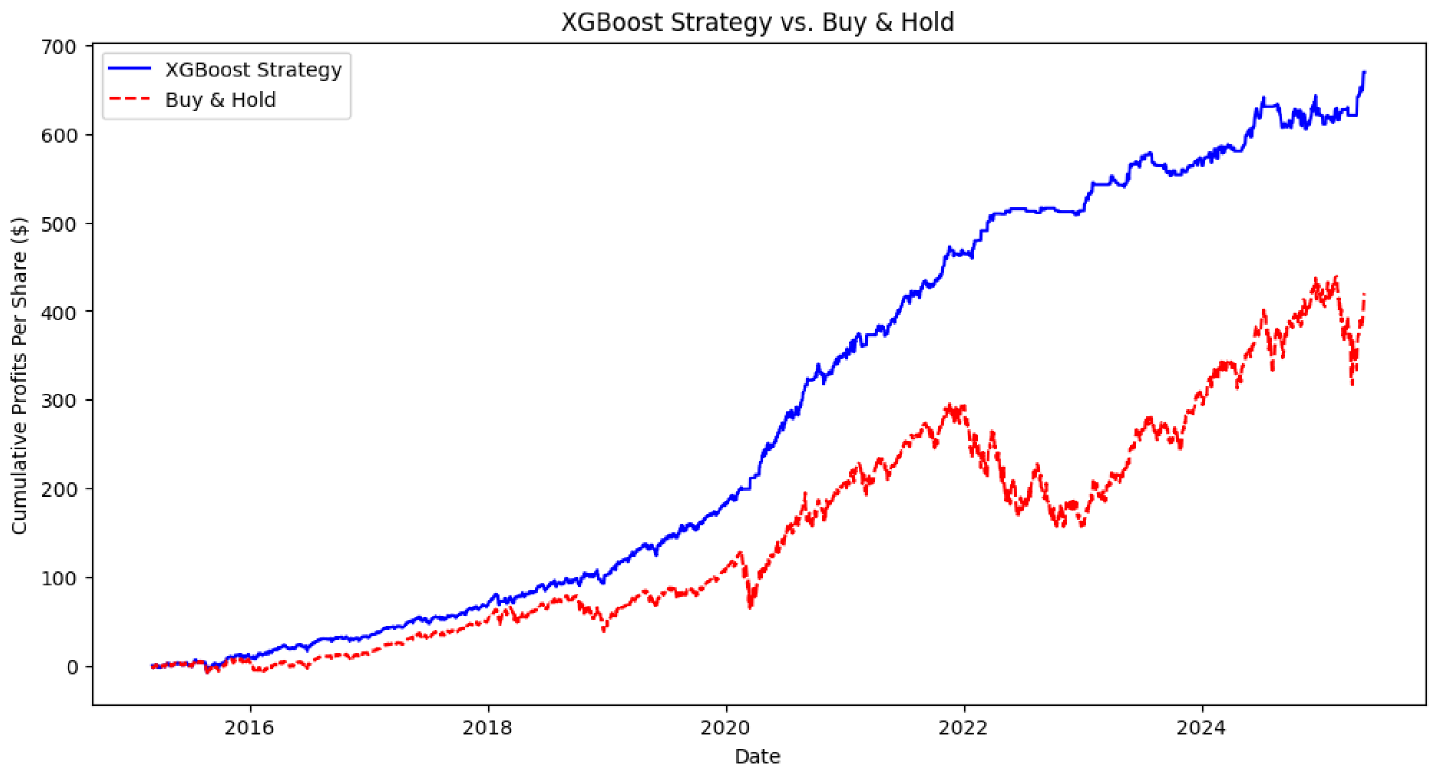
Meanwhile, to evaluate the strategy’s performance, introduce two additional columns:

- `Cumulative_Profit` : Tracks the total accumulated profit from all executed trades, providing insight into the strategy’s overall profitability.
- `Buy_Hold_Profit` : Represents the profit generated without any trading strategy—either from a simple buy-and-hold approach (holding QQQ until the last day) or daily buying and selling. This serves as a benchmark to assess how the XGBoost strategy compares to a passive investment approach.

Below are the last 5 rows of the newly created dataset as an example.

Date	Adj Close	Daily Return	XGBoost Prediction	Buy Price	Sell Price	Profit	Cumulative Profit	Buy Hold Profit
2025-05-09	487.970001	-0.000655	1	487.970001	0.000000	0.000000	649.641617	388.429703
2025-05-12	507.850006	0.040740	0	0.000000	507.850006	19.880005	669.521622	408.309708
2025-05-13	515.590027	0.015241	0	0.000000	0.000000	0.000000	669.521622	416.049728
2025-05-14	518.679993	0.005993	0	0.000000	0.000000	0.000000	669.521622	419.139694
2025-05-15	519.250000	0.001099	0	0.000000	0.000000	0.000000	669.521622	419.709702

Below is a plot comparing the profit trends of the XGBoost Strategy and Buy-and-Hold over the 10-year period:



The results demonstrate that the XGBoost strategy yields a profit of $\$669.52$ per share over the past 10 years, significantly outperforming the buy-and-hold approach, which generates $\$419.71$ per share — a 60% improvement in overall profits!

Additionally, the profit comparison plot highlights the smoother and more consistent growth of the XGBoost strategy. Unlike buy-and-hold, which experiences greater volatility and market downturns, the XGBoost-driven approach effectively navigates market fluctuations, maintaining a steadier and more robust upward trajectory throughout the period.

Back to Content

T2. Pyfolio Analysis

Pyfolio is a powerful Python library for analyzing portfolio performance and risk management, making it especially valuable for evaluating backtested trading strategies. It provides a comprehensive set of metrics and visualizations to assess a strategy's effectiveness.

To leverage this automated tool, we need to create a `XGBoost_Return` column, which captures the daily percentage return from trades executed using the XGBoost strategy. This serves as the core dataset for Pyfolio's analysis, enabling a detailed evaluation of the strategy's performance over time.

Here is the last 5 rows of the dataset now as an example:

Date	Adj Close	Daily Return	XGBoost Prediction	Buy Price	Sell Price	Profit	Cumulative Profit	Buy Hold Profit	XGBoost Return
2025-05-09	487.970001	-0.000655	1	487.970001	0.000000	0.000000	649.641617	388.429703	-0.00000
2025-05-12	507.850006	0.040740	0	0.000000	507.850006	19.880005	669.521622	408.309708	0.04074
2025-05-13	515.590027	0.015241	0	0.000000	0.000000	0.000000	669.521622	416.049728	0.00000
2025-05-14	518.679993	0.005993	0	0.000000	0.000000	0.000000	669.521622	419.139694	0.00000
2025-05-15	519.250000	0.001099	0	0.000000	0.000000	0.000000	669.521622	419.709702	0.00000

Then we submit the `XGBoost_Return` column to Pyfolio for analysis, reviewing the returns generated by the XGBoost strategy.

To benchmark performance, we also provide Pyfolio with the `Daily Return` column, representing the default returns without a strategy (buy-and-hold).

Here are the results:

Metric	XGBoost Strategy	Buy-and-Hold
Start Date	2015-03-09	2015-03-09
End Date	2025-05-15	2025-05-15
Total Months	122	122
Annual Return	32.5%	17.7%
Cumulative Returns	1657.5%	423.2%
Annual Volatility	13.5%	22.5%
Sharpe Ratio	2.16	0.84

Metric	XGBoost Strategy	Buy-and-Hold
Calmar Ratio	2.33	0.50
Stability	0.97	0.95
Max Drawdown	-13.9%	-35.1%
Omega Ratio	1.60	1.17
Sortino Ratio	3.50	1.19
Skew	0.46	-0.18
Kurtosis	7.37	7.45
Tail Ratio	1.23	0.92
Daily Value at Risk	-1.6%	-2.8%

As shown in the above table, the XGBoost strategy significantly outperforms the default buy-and-hold approach across multiple key metrics:

- It delivers a substantially higher annual return (32.5% vs. 17.7%) and greater cumulative returns (1657.5% vs. 423.2%) while maintaining lower volatility (13.5% vs. 22.5%), demonstrating a more stable trading pattern.
- The Sharpe ratio (2.16 vs. 0.84) and Sortino ratio (3.50 vs. 1.19) reinforce its superior risk-adjusted performance, showing that the strategy generates more efficient returns relative to its risk exposure.
- Capital preservation is significantly improved with the strategy, with a shallower maximum drawdown (-13.9% vs. -35.1%), protecting against extreme market downturns. Additionally, a higher Calmar ratio (2.33 vs. 0.50) highlights a stronger balance between return and drawdown risk.
- While the XGBoost model exhibits slightly higher skew (0.46 vs. -0.18) and similar kurtosis, its tail ratio (1.23 vs. 0.92) indicates better resilience to extreme price movements.

Overall, the XGBoost strategy proves to be a more profitable, risk-efficient, and stable approach for trading QQQ ETF compared to a simple buy-and-hold method. It not only enhances return potential but also mitigates volatility risks, making it a compelling choice for systematic trading.

My answer is finished, thank you!

Back to Content