FORECASTING DAILY RETURNS FOR S&P 500 CONSTITUENTS & IMPLEMENTING A TRADING STRATEGY

Authors: Takehisa Kanayama & Vladimir Zinkovski
Publication date: April 2024

1. INTRODUCTION

This project seeks to forecast daily returns for S&P 500 constituent companies. The forecast period is one-day ahead and approached as a binary classification task where we label positive returns as the positive class, and negative or zero returns as the negative class. We use two datasets to extract features that may be predictive of future returns. First, the open, high, low and closing prices, and trading volume data from Yahoo! Finance. Second, various macroeconomic indicators pertaining to the US economy from the Federal Reserve Economic Data (FRED). We use data from 2000 until 2023, which is split 18/3/3 years (i.e. 75%/12.5%/12.5%) for training/validation/testing. Using a long time horizon allows us to cover multiple black swan events, such as, for example, the dotcom bubble in 2000-2002, GFC in 2008-9, and COVID-related crash in 2020 and subsequent V-shaped recovery in 2021-22. Our intention is that incorporating outlier events into our data will result in more robust machine learning models. We train and tune three individual models using different underlying algorithms, which are then combined into two different ensemble models. Finally, we employ the best performing model into a simple trading strategy where if we predict the next day's returns as positive, we buy for a holding period of one day.

2. LITERATURE REVIEW

Forecasting stock returns has been extensively studied and is well-known as being a low signal-to-noise-ratio problem. Numerous studies, including one that tracked performance over a 20-year period, shows that even among professional money managers around 70-95% (depending on horizon length) were not able to outperform their respective benchmark – in this case the S&P 500– over the long run (Coleman, 2024). One reason has been the democratization of data where most traditional datasets have been extensively mined and any potential predictive signal already exhausted. However, the exponentially increasing amount of data available along with increasingly powerful computational resources opens up ever-more opportunities for quantitative investment strategies. Numerous quantitative hedge funds have indeed managed to display impressive performance in recent decades, notable examples being Renaissance Technologies, AQR Capital and Two Sigma, among others. This project follows the opinion that it is still possible to find signal among the noise and further takes inspiration from the works of two authors, Dr. Ernest Chan and Prof. Marcos Lopez de Prado. From the former, we incorporate ideas regarding how to combine data from different sources to produce novel datasets, thus allowing our machine learning models to potentially uncover new trading signals (Chan, 2008). From the latter, we implement the ideas of walk-forward time series cross-validation and decision threshold tuning (de Prado, 2018) for hyperparameter tuning and trading performance optimization, respectively.

3. METHODOLOGY

3.1 Datasets

Yahoo! Finance: This freely accessible API allows the user to download daily financial data over a specified date range. We begin by programmatically fetching a list of all current S&P 500 constituents from Wikipedia. This is our ticker list which we pass into the Yahoo! API, specifying a date range from 2020-01-01 up to and including 2023-12-31. We then take a subset of the returned pandas DataFrame to include only the open, high, low, close, adjusted close and trading volume columns.

FRED: The macro data gathering approach uses the “fredapi” library to fetch economic data from the Federal Reserve Economic Data (FRED) database. Key steps include:

1. Daily data**:** Collects daily data for the Federal Funds Effective Rate and the 10-year Treasury rate, forward filling missing values, and applying a one-day lag to prevent data leakage.
2. Monthly data**:** Retrieves monthly series for indicators such as CPI Core, Nonfarm Payroll, and Unemployment Rate, among others. It involves data transformations and calculation of moving averages for percentage columns using various time windows.
3. GDP data**:** Acquires quarterly GDP data, keeping the first three updates per release and converting these to daily data through forward filling.

Combined dataset: The preceding two datasets are merged using a left join, where we keep all trading days from the (left) Yahoo! dataset and discard those from the (right) FRED dataset which are non-business days.

3.2 Feature Engineering

This project employs various distinct techniques to generate additional features and thus extract additional information from our existing columns. The primary intention is to capture time series dependency, i.e. autocorrelation, for each of the features; as well as to assist our models in capturing cross-dependencies and other interaction effects between different features.

Technical indicators: We use the TA-lib library to compute various technical indicators using the adjusted close as the input feature for each stock. Adjusted closing price is chosen as it accounts for dividends, stock splits, and new stock offerings thereby creating a uniform data generating process and continuous price for effective model training and backtesting (i.e. removing certain forms of regime changes). We choose five indicators from two overarching groups: momentum and volatility. Momentum helps identify the speed of price changes, while volatility measures the magnitude of those changes.

Aggregate features: Calculates the daily mean adjusted closing price and standard deviation across all constituents.

Return features: Allows us to capture percentage returns between the value at the current timestep versus a value at timestep-*n*. We perform this operation up to 10 steps back into the past for the adjusted closing price, and up to 5 steps back for the remaining price and volume features. We reach further back in time for the adjusted close as it is likely to contain more signal and we wish to retain more of its historical information into the current timestep. Five and ten trading days align with one and two calendar weeks – common measures of comparison used in practice.

Lagged features: We compute lagged values of the percentage daily return and binary daily return features for up to 10 observations into the past. Retaining this information allows us to model the mean-reverting quality of returns.

Rolling features: We compute various rolling features across different window lengths. We perform this operation taking the mean values for all FRED macroeconomic features with lookback periods of 7, 30, 60 and 90-days. Similarly, we perform this operation taking the minimum, maximum, mean and standard deviation for the percentage daily return with lookback periods of 5, 10, 20 and 60-days. The differing windows are to reflect a weekly, monthly and quarterly economic cycle for macroeconomic data and a business cycle of 5 trading days per week for stock data.

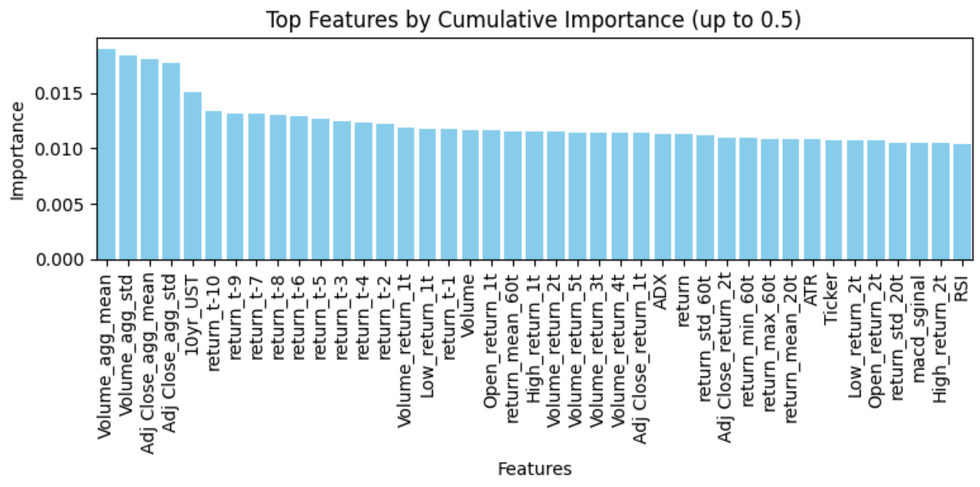
Target variable: In order to calculate our target variable, we first calculate the percentage daily returns of the adjusted close. Returns are chosen due to their stationary, mean-reverting properties and hence predisposition to modeling. Next, we binarize the returns where all returns above zero are the positive class (1) and all else (0). Next, because we want to forecast tomorrow's binary return, as well as avoid introducing any lookahead data leakage, we backward shift all binary returns by one timestep. In other words, we now have all of yesterday's data available to forecast today's return.

3.3 Feature Selection

We employed a RandomForest-based feature selection approach. Using RandomForest's feature importance is advantageous in that it identifies and retains the most relevant variables, while it keeps interpretability. Given that the feature importances are a function of the models' performance, we followed a hyperparameter tuning process and identified the best parameters before extracting top features. The steps we took are as follows:

1. Data preprocessing: Used train + valid dataset, with preprocessing to encode categorical data using OrdinalEncoder.
2. Model pipeline: A RandomForestClassifier was incorporated into a pipeline with hyperparameter tuning using RandomizedSearchCV and TimeSeriesSplit for cross-validation, optimizing for precision.
3. Feature importance extraction: Post training, extracted feature importances from the best model to identify the most impactful predictors.

4. Feature selection: Implemented a cumulative importance threshold of 50% to select key features, improving model efficiency and focus on relevant predictors.



3.4.1 Individual Models: Overview

RandomForest: An ensemble learning method that builds multiple decision trees. Our expectation of the model is capturing complex, non-linear relationships in financial data, while lowering the risk of overfitting as averaging multiple trees is expected to reduce the risk of overfitting.

LightGBM: A gradient boosting framework that uses tree-based learning algorithms. It is designed for distributed and efficient training, particularly on large datasets. LightGBM uses Gradient-based One-Side Sampling and Exclusive Feature Bundling, which increases efficiency and reduces memory usage. It can handle large datasets efficiently, crucial for the extensive amount of datasets we have.

Deep neural network: We define a feedforward neural network with two hidden layers using the pytorch library. The network uses a ReLU activation function in both hidden layers and a softmax activation in the output layer to perform binary classification. We also apply dropout in the first hidden layer as a regularization technique.

3.4.2 Individual Models: Hyperparameter Tuning

We took consistent steps of hyperparameter tuning and cross-validations across the models to fairly compare their performance. Specifically:

1. Preprocessing: We used consistent preprocessing. For numerical variables, we used median imputation combined with scaling to normalize their distribution, ensuring robust handling of missing values and reducing the influence of outliers. Categorical variables

were imputed using the most frequent category and encoded ordinally, allowing the model to interpret these as numerical data efficiently.

2. Hyperparameter tuning: Across the models, RandomizedSearchCV with TimeSeriesSplit was used to tune hyperparameters while avoiding data leakage in time-series data. Precision was selected as the scoring metric to focus on the accuracy of positive predictions.
3. Cross-validation: After identifying the best parameters, we utilized TimeSeriesSplit for cross-validation with three splits and a test size of 126,000 (252 business days / year x 500 stocks) observations per split to make each split roughly annual data.

RandomForest

The Random Forest model is integrated with preprocessing steps in a pipeline to maintain consistency. Key hyperparameters optimized include:

- n_estimators: Controls the number of trees in the forest. This affects model robustness, by controlling the number of trees.
- max_depth: Limits the depth of each tree to prevent overfitting.
- min_samples_split & min_samples_leaf: Regulate the growth of trees to ensure they generalize well.
- max_features: Determines the number of features for splitting nodes, balancing model complexity and variance.

The selection of hyperparameters for tuning was strategically aimed at balancing model performance and preventing overfitting.

LightGBM

We integrated LightGBM into our pipeline post-preprocessing to maintain consistency and reproducibility in model training. The LightGBM model was tuned for the following key hyperparameters:

- num_leaves: Controls the complexity of the model by limiting the number of leaves in each tree.
- learning_rate: Affects the rate at which the model learns, smaller values potentially leading to better generalization.
- n_estimators: Specifies the number of boosting stages the model will go through, akin to the number of trees in Random Forest.
- lambda_l1 & lambda_l2: L1 and L2 regularization terms on weights, encouraging model simplicity and avoiding overfitting.

We chose hyperparameters focusing on balancing complexity and overfitting.

Deep Neural Network

Deep learning models are known to have many tunable hyperparameters and we have limited our selection to those most commonly explored when optimizing model performance:

- Learning rate: Controls the size of the steps taken during gradient descent, impacting how quickly and accurately the model learns.
- Maximum number of epochs: The number of complete passes through the training data, controlling the stopping point of the training phase.
- Batch size: The number of data samples processed before updating model parameters during each iteration of training, thus influencing the granularity of parameter updates.
- Number of units: The number of neurons in the first and second hidden layers, affecting the model's dimensionality and capacity to learn complex patterns.
- Dropout rate: The proportion of neurons randomly removed during each training iteration, mitigating against overfitting by promoting generalization.

In order to pass our pytorch model through sklearn's RandomizedSearchCV cross-validation step, we first use a wrapper provided by the skorch library. This provides compatibility between pytorch and the native sklearn API.

3.5 Ensemble Models

Ensembling is a machine learning technique where multiple individual models are combined with the aim to improve predictive performance as individual model errors are averaged out to produce more robust forecasts. We employ the two most common forms of ensembling, both of which are available in the sklearn library.

VotingClassifier
We use soft voting, where the output probabilities of all three individual models are weighted equally and averaged in order to make forecasts.

StackingClassifier
This ensemble builds a meta-model which takes as its three input features the output probabilities of each individual model. We use the default LogisticRegression as our meta-model, which learns how much importance to assign to each of the individual models and thus apply a weighted averaging to make forecasts.

3.6 Classification Threshold Tuning

After evaluating all individual and ensemble models on the validation set, we choose the model with the highest precision score. We then use the Optuna library to further optimize its classification threshold (aka decision threshold) such that it maximizes cumulative gross equity returns for the trading strategy built upon the model's forecasts.

3.7 Trading Strategy

The strategy aims to maximize intraday trading returns by optimizing the decision threshold for converting predictive probabilities into binary trading signals. This threshold is crucial for determining when to buy or sell based on model predictions.

Using Bayesian hyperparameter optimization via Optuna's TPE sampler, the strategy's objective is to maximize the cumulative return by:

1. Converting probabilities to binary signals based on a dynamic threshold.
2. Shifting signals to simulate real-world trading lag.
3. Calculating daily profits by applying these signals to intraday price changes.
4. Maximizing the cumulative return at the end of the time series.

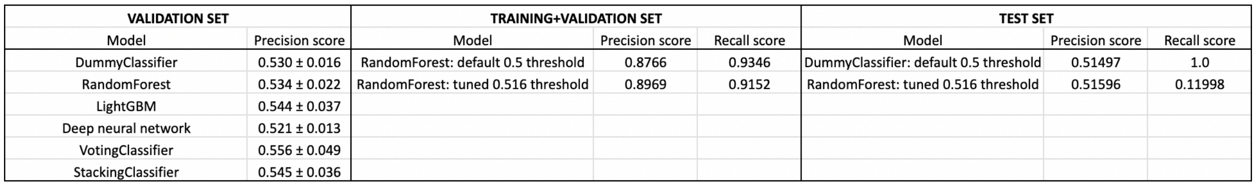
4. EVALUATION STRATEGY & RESULTS

Our project follows the following workflow:

- Training set: From 2000-01-01 up to and including 2017-12-31. Used to perform time series walk-forward 3-fold cross-validation hyperparameter tuning where we seek to maximize the precision score. Each test fold is 252 trading days which approximates one calendar year. We chose 3-folds to keep a consistent 3-year period, similar to the size of both the validation and test sets.
- Validation set: From 2018-01-01 up to and including 2020-12-31. Used to evaluate the performance of all individual and ensemble models. The model with the highest precision score is chosen for the trading strategy. Benchmark performance is established using sklearn's DummyClassifier which always predicts the positive (in this case also the majority) class – and this is equivalent to a buy-and-hold trading strategy.
- Training and validation set: Used to optimize the classification threshold of the best model such that gross cumulative equity returns are maximized.
- Test set: From 2021-01-01 up to and including 2023-12-31. Used to perform one definitive out-of-sample evaluation of the best model, which at this stage has both tuned hyperparameters and a tuned classification threshold. Benchmark performance is established using both a DummyClassifier and a naive buy-and-hold trading strategy.

4.1 Precision Score

The precision score is chosen as the metric to be maximized. It is defined as *TP / (TP+FP)*, where TP are true positives and FP are false positives. In order to maximize this score, we are therefore interested in increasing the true positives and/or decreasing false positives. In practice, this translates to classifying only those instances as positives where we have a high degree of confidence. From a trading perspective, this results in a higher percentage of winning trades, however the trade-off is fewer trades overall (i.e. the precision recall trade-off).

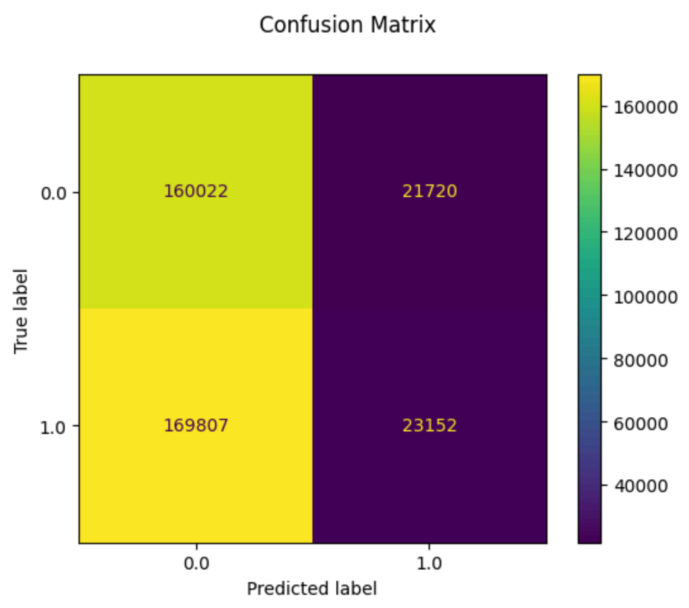


VALIDATION SET		TRAINING+VALIDATION SET			TEST SET		
Model	Precision score	Model	Precision score	Recall score	Model	Precision score	Recall score
DummyClassifier	0.530 ± 0.016	RandomForest: default 0.5 threshold	0.8766	0.9346	DummyClassifier: default 0.5 threshold	0.51497	1.0
RandomForest	0.534 ± 0.022	RandomForest: tuned 0.516 threshold	0.8969	0.9152	RandomForest: tuned 0.516 threshold	0.51596	0.11998
LightGBM	0.544 ± 0.037						
Deep neural network	0.521 ± 0.013						
VotingClassifier	0.556 ± 0.049						
StackingClassifier	0.545 ± 0.036						

The above table shows the performance metrics of our models:

- During training, the highest average precision score of 0.556 was achieved by the VotingClassifier, however when accounting for standard deviation it also meant that in one calendar year its precision score was only 0.507 – only slightly better than random guessing. In practice, this may result in worse trading performance due to larger drawdowns, which is not always well-received by investors. As such, we choose the model with the lowest absolute precision score, which is the RandomForest at 0.512 (i.e. 0.534 - 0.022).
- Over the training and validation stage, the RandomForest decision threshold is optimized and shifted from 0.5 to 0.516. Subsequently, this results in the precision score increasing from 0.8766 to 0.8969 (note the trade-off with recall, which decreases from 0.9346 to 0.9152).
- On the test set, the DummyClassifier has perfect recall as it is able to find all positive instances, however its precision score is only 0.51497. Our RandomForest outperforms with a score of 0.51596, but conversely finding many fewer of the positive instances and therefore making less trades overall.

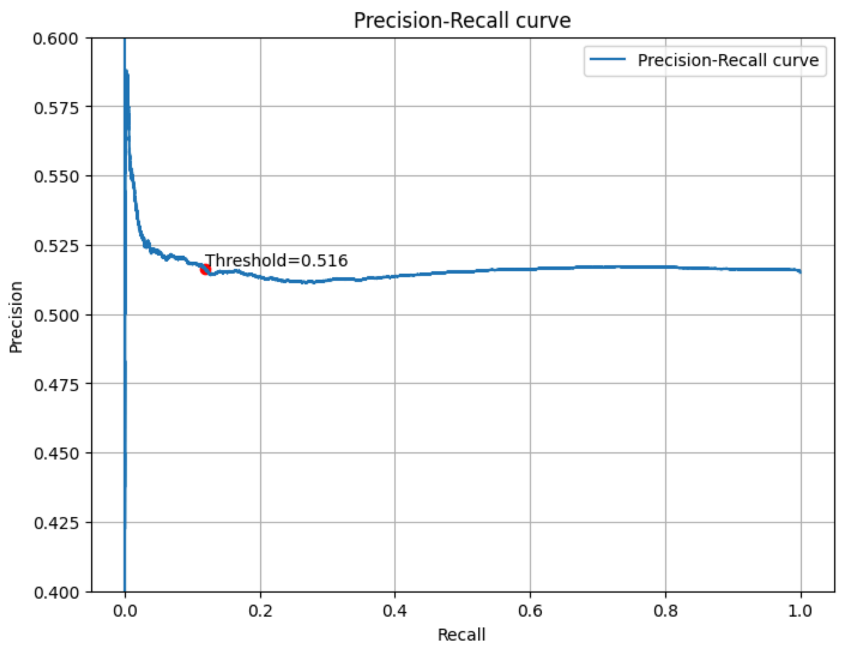
The following confusion matrix summarizes our best model performance on the test set. We can see that the precision score is 23152 / (23152 / 21720) = 0.51596. The key takeaway is that this aligns with the notion that financial time series contain a very low signal-to-noise ratio and despite the low absolute precision score, it may be regarded as reasonable having outperformed the DummyClassifier in both validation and test sets.



4.2 Precision Recall Curve

The precision-recall curve is a standard evaluation metric for binary classifiers. While it provides an essential perspective on the trade-off between precision (the ratio of true positives to combined true and false positives) and recall (the ratio of true positives to combined true positives and false negatives), it does not inherently account for the financial impact of the decisions made at each threshold.

In the context of a trading strategy, the optimal threshold—marked on the curve at 0.516—was selected not just for its precision-recall balance, but also for its maximization of stock returns. The threshold that yields the highest precision and recall may not align with the threshold that maximizes financial performance. The marked threshold on the precision-recall curve represents a deeper level of strategy optimization that considers the monetary implications of the model's predictions.



4.3 Our Trading Strategy vs. Benchmark Strategy

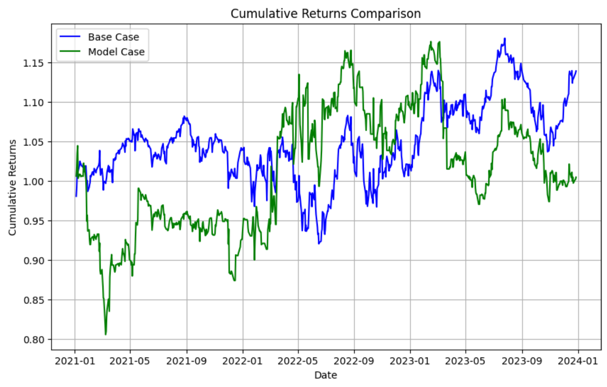
We finally compared the trading performance of the model with a base case with the following assumptions:

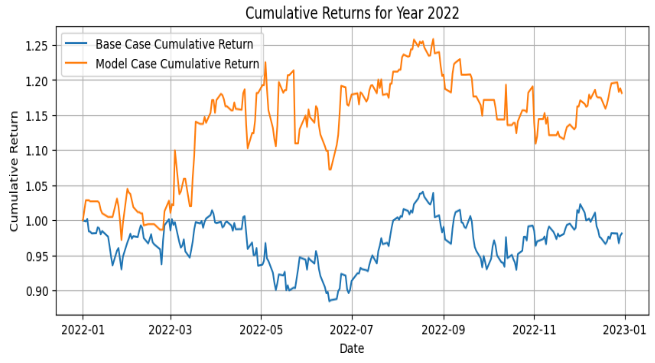
Base case definition**:** All stocks are bought and sold in equal amounts every trading day, without any selective signals from the model.

Model case definition**:** Conversely, the model case scenario involves buying only the stocks signaled by the model using the same total amount of money allocated in the base case. On days without a buy signal, no investments are made.

We applied cumulative return for the trading performance evaluation. It represents the total return achieved by consistently reinvesting the full amount daily. A cumulative return of 1.1 signifies a 10% gain on the investment, where, for example, $1 million would grow to $1.1 million. (The evaluation assumes a simplified scenario where no annual taxes, commissions, slippage, and other fees are considered in the calculation of returns).

The model's performance varied by year between 2021 and 2023, with overall results falling short of the base case over the full period. It excelled in the bear market of 2022, as its selective approach minimized losses by choosing stocks with growth potential. However, during the bull markets of 2021 and 2023, the model underperformed since its cautious strategy missed many opportunities presented by the generally rising market.





Finally, the Sharpe ratio, a measure of risk-adjusted return, was calculated for both the base and model cases. The ratio is calculated by dividing the annualized excess cumulative return over the risk-free rate (taken to be the 3-year US treasury yield) by the annualized standard deviation of the entire stocks in the same period. The Sharpe ratio provides insight into how much additional return was generated for each unit of risk taken on by the model compared to the base case.

The result was 0.2963 and -0.0009 for the base case scenario and model case scenario, respectively.

5. DISCUSSION

Impact of volatile years on model predictions: Incorporating volatile years like 2008, 2009, and 2020 in our training poses challenges due to significant market fluctuations. These conditions can skew predictions, potentially limiting the model's generalizability to more stable periods. Implementing a stratified sampling strategy might balance exposure and enhance robustness across diverse market conditions.

Reevaluation of trading strategy: Our simplistic fixed-capital, one-day holding period strategy was designed to test machine learning model efficacy but may not adequately reflect real-world trading complexities. Adapting the strategy to include variable holding periods and dynamic capital adjustments could provide more realistic insights and improve adaptability to market changes.

Future enhancements and integrations: Future enhancements will focus on collecting more variety of data and its manipulation such as integrating company-specific financial data and innovative feature engineering techniques.

6. BROADER IMPACTS

This work could affect investors, traders, and portfolio managers who rely on predictions to make informed financial decisions. Portfolio managers also stand to benefit by incorporating these predictions into broader investment strategies.

An ethical concern that arises from this work is the potential for misuse of predictive analytics to gain unfair market advantages, leading to issues like market manipulation.

7. CONCLUSION

Despite deriving our model features from just two – exhaustively mined – data sources, we were nevertheless able to build a machine learning model which outperformed the benchmark DummyClassifier in both the validation and test sets. Although more complex ensemble models did indeed result in higher precision scores, these were, somewhat counterintuitively, coupled with a higher standard deviation. As a result, the RandomForest was our best performing model.

However, our subsequent trading strategy was not able to outperform a simple buy-and-hold trading strategy, underperforming specifically during bull market years. Even though our trading strategy trailed its benchmark, it did manage to break even. As such, this can be considered a reasonable initial prototype trading model which may benefit from further refinements such as weighted capital allocation, scaling in and out of positions, and variable holding periods, among others.

Finally, ingesting data from additional sources, particularly non-traditional sources (e.g. satellite imagery, text, etc.), applying further feature engineering, and more nuanced feature selection, may all lead to additional performance gains and are potential areas for further work.

8. STATEMENT OF WORK

Takehisa Kanayama: data ingestion and feature engineering for FRED data; feature selection; LightGBM and RandomForest modeling and hyperparameter tuning; trading strategy development; precision-recall curve, equity curve and Sharpe ratio evaluation, discussion, broader impacts; poster

Vladimir Zinkovski: introduction; literature review; data ingestion and feature engineering for Yahoo! Finance data; deep neural network modeling and hyperparameter tuning; VotingClassifier and StackingClassifier ensemble modeling; decision threshold tuning; precision score and confusion matrix evaluation; conclusion; GitHub repository

9. REFERENCES

Chan, E. P. (2008). *Quantitative Trading: How to Build Your Own Algorithmic Trading Business*. John Wiley & Sons.

Coleman, M. (2024, March 18). Active Fund Managers vs. Indexes: Analyzing SPIVA Scorecards. *IFA*. https://www.ifa.com/articles/active-fund-managers-benchmark-analysis-sp

López de Prado, M. (2018). *Advances in Financial Machine Learning*. John Wiley & Sons.

APPENDIX A: Saving stock tickers from Wikipedia as a text file.

```
# get list of sp500 tickers from wikipedia
sp500_tickers =
pd.read_html('https://en.wikipedia.org/wiki/List_of_S%26P_500_companies')[
0]
sp500_tickers = sp500_tickers.Symbol.tolist()

# save sp500 tickers to text file in google drive
sp500_tickers_path = '/content/drive/MyDrive/Colab
Notebooks/Capstone/sp500_tickers.txt'
with open(sp500_tickers_path, 'w') as f:
   for symbol in sp500_tickers:
       f.write(symbol + '\n')
```

APPENDIX B: Saving stock data from Yahoo! Finance as a parquet file.

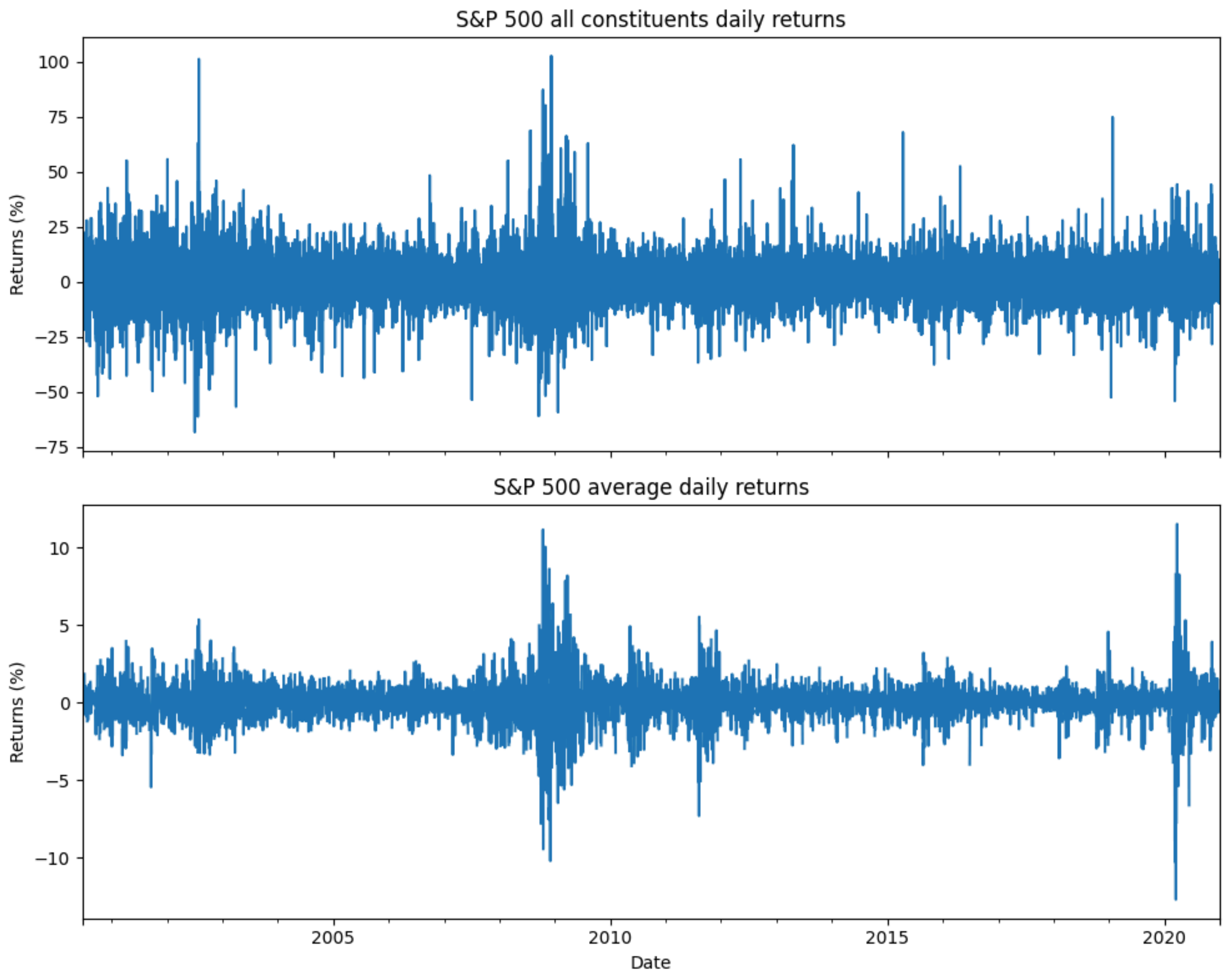
```
# download ohlc price and volume data
yf_df = yf.download(sp500_tickers, start=start_date, end=end_date)

# save sp500 data to parquet file in google drive
sp500_data_path = '/content/drive/MyDrive/Colab
Notebooks/Capstone/sp500_data.parquet'
yf_df.to_parquet(sp500_data_path, compression='gzip', index=True)
```

APPENDIX C: Saving the combined Yahoo! Finance and FRED dataset as a parquet file.

```
# save final feature engineered df containing sp500 and fred data
sp500_fred_data = '/content/drive/MyDrive/Colab
Notebooks/Capstone/sp500_fred_data.parquet'
main_df.to_parquet(sp500_fred_data, compression='gzip', index=True)
```

APPENDIX D: Stationarity of S&P 500 daily returns for both each constituent individually (top chart – all constituents overlaid on top of each other) and the average of all constituents (bottom chart).



APPENDIX E: Checking class imbalance over the training and validation data (excluding the test set to avoid lookahead bias). Classes are well balanced.

```
# check class imbalance
print(f"""
   Positive class:
{main_df['y'][:end_valid_date].value_counts(normalize=True)[1]:.4f}
   Negative class:
{main_df['y'][:end_valid_date].value_counts(normalize=True)[0]:.4f}
""")
   Positive class: 0.5147
   Negative class: 0.4853
```