Udacity MLND Capstone Project Report

Ruoyu Lin

Started Jan. 25 2022

1 Project Definition

1.1 Project Overview

Quantitative finance/financial engineering is the practice of approaching financial analytics through extensively technical methods, such as dynamical stochastic models and machine learning approaches. Inspired by Burton Malkiel's investment thesis in *A Random Walk Down Wall Street*, where he argue that the vast majority of investment fund managers cannot outperform the market, we aim to using machine learning to design a trading strategy that minimizes reallocation of portfolio. In other words, should an agent trust that she cannot outperform the efficient market but still can harvest some information from the market data, she will put in her money in the stock market, and only adjust the position when she's adequately certain that the market will go down the following day.

Hence, in this project, we attempt to build and train a stock price forecaster using XG-Boost algorithm ingested with historical stock market data at daily frequency, that tries to detect market down days to simulate the adjustment of portfolio positions before the predicted market down day. The stock market data, which is free for public use, will be extracted from yahoo finance using Python library `yfinance`, at the frequency of one day. All data that will be used are from 2017-1-1 to the most recent date when the project report is written down (approximately 2022-1-25).

1.2 Problem Statement

The problem in this project then, seems self-explanatory – can the raw, public market data of stock prices be used to make predictions of the next day returns, and enable an investing agent to sell before the market goes down? Or, is the market efficient enough, such that all information contained in the price that could have enable arbitrage had already been traded away?

In other words, does the public market data contains any digestible information for a machine learning algorithm? Does the data contain predictive power? If so, our trained model should be able to produce adequately accurate predictions, reflected either through prediction precision or high returns in a simulated trading experiment in past out-of-sample financial data.

1.3 Metric

We will measure the performance of our model through two means: first of, like all binary classification problems, measures such as prediction accuracy or precision can be employed to evaluate the classifier; however, on the other hand, since the application field of this project is in quantitative finance, whose final goal is to maximize net portfolio value, the portfolio return yielded by the predictions made by our model can also be used to evaluate the model. In this study, we will employ both to evaluate our model.

2 Analysis

2.1 Techniques and algorithms

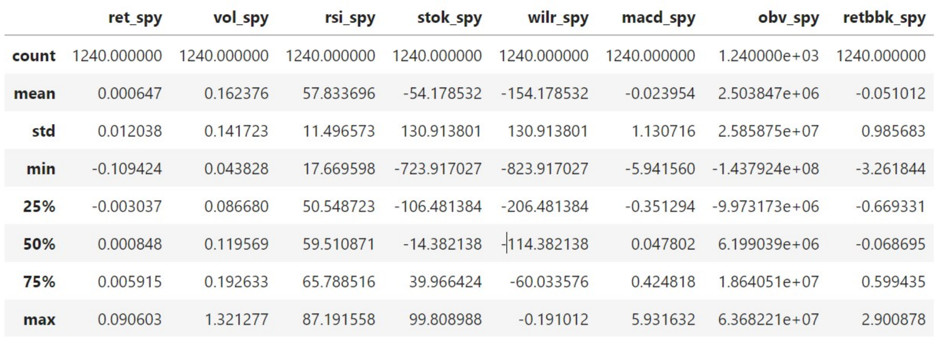
We will make use of known financial technical indicators to analyze the probability of a market down day. In finance, this is called technical analysis, which is a practice by financial analyst/traders to extract information from the raw market data. In the context of machine learning, this is equivalent to feature engineering under supervised learning environment.

The usage of technical indicator are usually combined with some thresholds. For example, it's a common practice to interpret RSI > 70 to be a sell signal, and RSI< 30 to be a buy signal. What we will do different in our project than technical analysis is, we will leave the reading of these indicators to the algorithm, instead of setting arbitrary thresholds beforehand. To achieve this, we will make use of XGBoost, the extreme gradient boosting system designed by Tianqi Chen in 2016, to read technical indicator as a more traditional ensemble classifier like random forest would do. In both theory and anecdotal practices, XGBoost generally performs vastly better than a traditional random forest model, as each estimator is further boosted with computed gradient.

2.2 EDA and Visual Analysis

We split the data into train set (trading days in 2017 to 2020, $n = 968$)and test set (trading days from 2021 Jan to now, $n = 270$). First, we describe the generated features. See below for summary statistics of generated indicators:

Figure 1: Indicator Summary



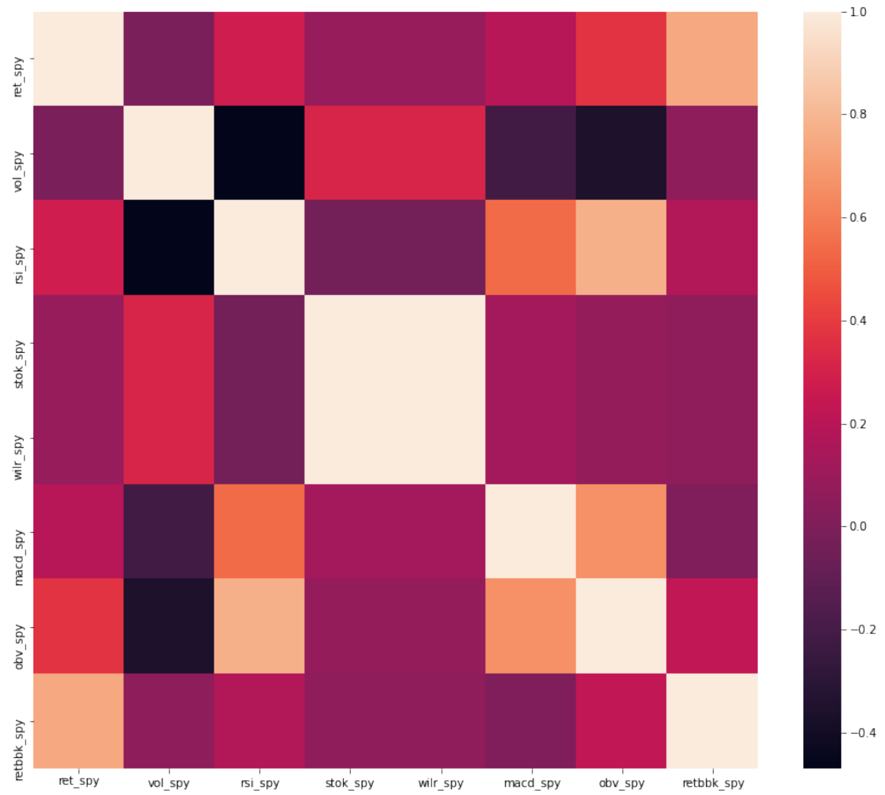
	ret_spy	vol_spy	rsi_spy	stok_spy	wilr_spy	macd_spy	obv_spy	retbbk_spy
count	1240.000000	1240.000000	1240.000000	1240.000000	1240.000000	1240.000000	1.240000e+03	1240.000000
mean	0.000647	0.162376	57.833696	-54.178532	-154.178532	-0.023954	2.503847e+06	-0.051012
std	0.012038	0.141723	11.496573	130.913801	130.913801	1.130716	2.585875e+07	0.985683
min	-0.109424	0.043828	17.669598	-723.917027	-823.917027	-5.941560	-1.437924e+08	-3.261844
25%	-0.003037	0.086680	50.548723	-106.481384	-206.481384	-0.351294	-9.973173e+06	-0.669331
50%	0.000848	0.119569	59.510871	-14.382138	-114.382138	0.047802	6.199039e+06	-0.068695
75%	0.005915	0.192633	65.788516	39.966424	-60.033576	0.424818	1.864051e+07	0.599435
max	0.090603	1.321277	87.191558	99.808988	-0.191012	5.931632	6.368221e+07	2.900878

We aim to create features such that they do not overly correlated. To visualize all the features we created, we utilize a heatmap:

as we can see that except for William's R and Stochastic Oscillator, whose definitions are similar, most of the features generated are not overly correlated. Finally, we visually examine our target variable – SPY returns. See below for a figure capturing both the time series of the returns and its distribution:

While asymptotic normality assumption is adequate, as shown by the distribution plot, we notice that there's more weight centered in the downside tail, while the mean is slightly above 0 (this is normal as stock market follows a geometric Brownian motion with positive drift, thus always scales upward). The event realized with probability measures in this tail

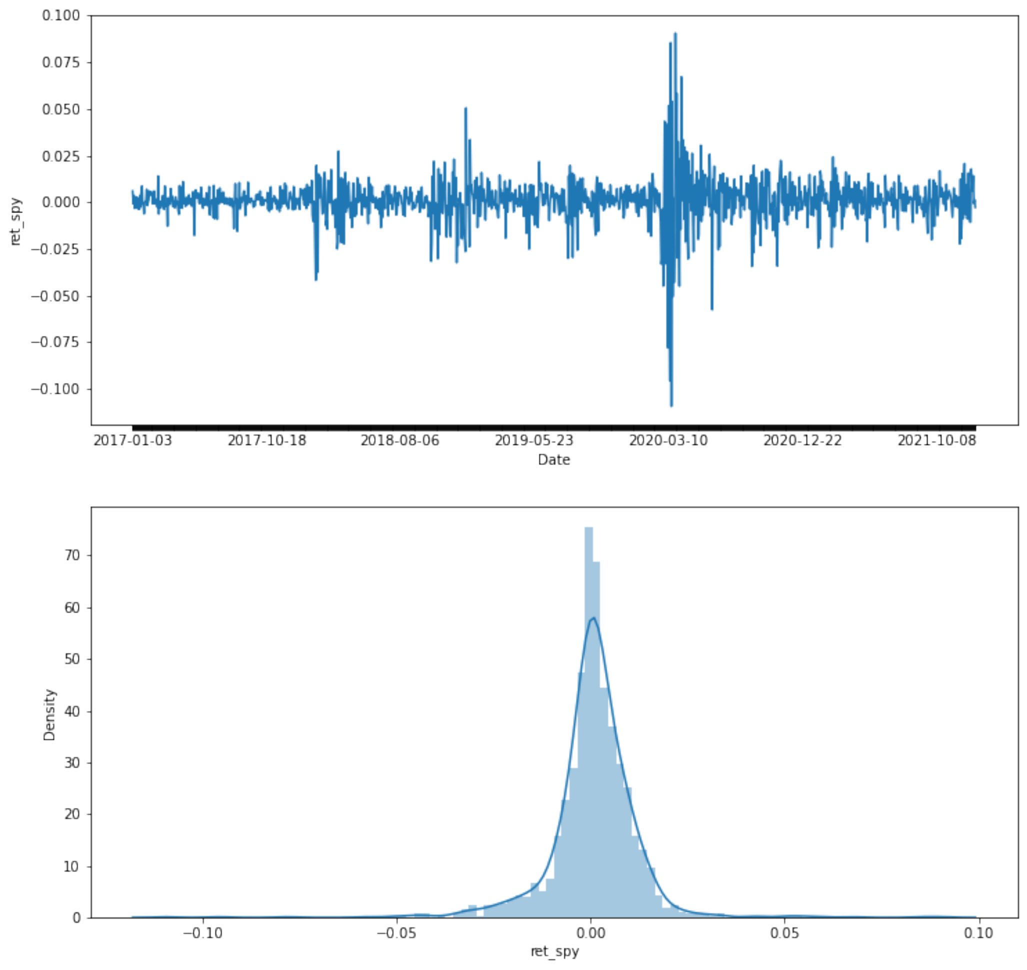
Figure 2: Feature Correlations



is what we want to capture in our project.

At last, since our project uses time series data, it's worth showing the autocorrelation plot of spy: Note that the one day correlation quickly dissipates as the lags increase. While this is not favoring our analysis, this is expected, as SPY index is an highly efficient index, and hence we do expect that almost all arbitrage opportunities are traded away.

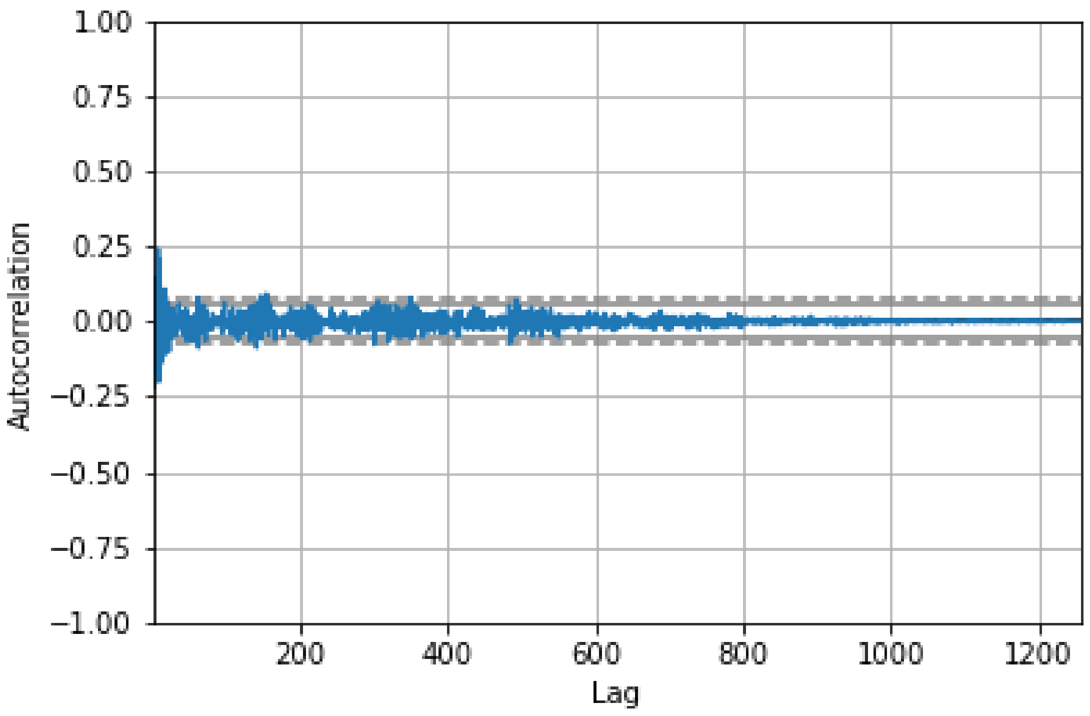
Figure 3: SPY returns



2.3 Benchmark

The benchmark model I consulted is actually not a XGBoost model like I used, but a random forest ensemble performed on a similar set of indicators [2]. However, this still serves as a valid benchmark to our studies, as classifier based on extreme gradient boosting can be viewed as a modified ensemble model, with number of boosting rounds homomorphic to

Figure 4: SPY Autocorrelation



that of the number of decision trees in a random forest.

For technical analysis, the smaller the validation window gets, the lower the prediction accuracy will get, as there exists more variance attributed to noise instead of structural patterned detected by the model. This is not true when there's abundant data following some underlying pattern, but this is unfortunately the case when it comes to time series analysis with limited data (as even if there exists a underlying distribution/structural pattern, there's only one possibility realized in the world – our observation). This theoretical phenomenon is conveyed verily in the benchmark model study, and for one day forecasting, the benchmark model only achieved about 0.6 accuracy when it comes to one day validation window forecasting. Thus, we do not expect this project to be performing better than the benchmark.

Finally, there exists another change worth mentioning on top of the benchmark model. While the original study aims to forecast the absolute direction of the stock price (positive return or negative return), our study inspired by Burton Malkiel aims to "cut loss" in a dynamic trading strategy. Therefore, instead of defining the classes to be positive and negative return days, we divide the observations into ones with lower than -0.5% returns and those with more than -0.5 returns.

3 Methods

3.1 Processing and Feature Engineering

According to EDA and visual examinations, we did not detect outstanding abnormalities that require our attention and treatment. Hence, we proceed to the feature engineering that occupies most of the data transformations As mentioned in algorithm section, the generation of technical indicator in financial forecast is essentially feature engineering in a supervised learning setting. We produce some classical technical indicators as follows:

- *Returns.* The stock returns, daily frequency, characterized by

$$\text{ret} = \frac{P_t - P_{t-1}}{P_{t-1}}$$

  where P_t is the price of stock at period t.

- *OHLC Volatility.* A function of stock price high, low, close and open that describes how volatile the stock price is, characterized by

$$\sigma_{gk}^2 = \frac{1}{T}\sum_{t=0}^{T}(0.511\log(\frac{H_t}{L_t}))^2 - 0.019\log(\frac{C_t}{O_t})\log(\frac{H_tL_t}{O_t^2}) - 2\log(\frac{H_t}{O_t})\log(\frac{L_t}{O_t})$$

where O, H, L, C denote open, high, low, close respectively.

- *Relative Strength Index.* RSI is a momentum indicator which aims to indicate whether a stock is overbought or oversold. It is characterized by

$$\text{RSI} = 100 - \frac{100}{1+\text{RS}}$$
$$\text{where RS} = \frac{\text{Avg. gain in past 14 days}}{\text{Avg. loss in past 14 days}}$$

- *Stochastic Oscillator.* The stochastic $\%K$ traces the momentum of the adjusted close price, measured by the level of price relative fo the low-high spread of a period. This is characterized by

$$K = \frac{C - L_{14}}{H_{14} - L_{14}}$$

where H, L, C denotes high, low, close price respectively.

- *William's R.* The William's R is a similar momentum indicator to stochastic oscillator, but instead uses the difference between the high and close as the momentum indicator:

$$R = \frac{H_{14} - C}{H_{14} - L_{14}} - 100$$

where H, L, C denotes high, low, close respectively.

- *Moving Average Convergence Divergence.* MACD is a very popular technical indicator the signals possible future price movement, with the assumption that the exponential moving average of different time windows should be convey the price convergence.

$$\text{MACD} = ema_{12}(C) - ema_{26}(C)$$
$$\text{Signal} = ema_9(MACD)$$

  When MACD − Signal crosses from negative to positive, it signals a sell signal, and vice versa.

- *On Balance Volume.* On balance volume indicates the buying/selling trend of a stock. The formula characterizing OBV is given recursively as follows:

$$\text{OBV}(t) = \begin{cases} \text{OBV}(t-1) + \text{Volume}(t) & \text{if } C(t) \geq C(t-1) \\ \text{OBV}(t-1) - \text{Volume}(t) & \text{if } C(t) < C(t-1) \end{cases}$$

  where C denotes close price. $\text{OBV}(t_0)$ is initialized as 0.

- *Bollinger Band Rolling Window Score.* A Bollinger Band is an objective created based on the assumption of mean reversion: given a rolling window, a Bollinger Band is composed of upper band and lower band:

$$\mathrm{BB}_U = ma_{14}(C) + k \cdot mstd_{14}(C)$$
$$\mathrm{BB}_L = ma_{14}(C) - k \cdot mstd_{14}(C)$$

where $ma_{14}(C)$ denotes 14-days moving average of close and $mstd_{14}(C)$ denotes 14-days moving standard deviation of close, and k is some arbitrary multiplier defined such that when the adjusted close price crosses the upper band or lower band, a buy/sell signal is indicated. Our modified BB score computes k, instead of defining the crossing of upper/lower band, is defined by:

$$\mathrm{BBK} = \frac{C - ma_{14}(C)}{mstd_{14}(C)}$$

Hence when k is "sufficiently" high/low as determined by the model, a buy/sell signal is signaled to the model.

The feature engineering process is carried out in a modular fashion. After generating these features, the data is again passed into a custom function to generate time period lags for all features. These steps will be discussed in more details in the next subsection "Implementation".

3.2 Implementation

3.2.1 Feature Engineering Implementation

For each feature selected, a feature-generating function is written down according to the rules specified above. For example, for technical indicator William's R:

```
def gen_wilr(df, tickers, periods = 14): # technical indicator: William's R%


    for ticker in tickers:

        ticker = ticker.lower()

        hh = df.loc[:,f"high_{ticker}"].rolling(periods).max()

        ll = df.loc[:,f"low_{ticker}"].rolling(periods).min()

        df.loc[:,f"wilr_{ticker}"] = (hh - df.loc[:,

            f"adj close_{ticker}"]

            ) / (hh - ll) * (-100)


    return df
```

Note that the feature-generating functions are carried out in a pipeline fashion: a pandas dataframe is passed into the function, and the output is as well a dataframe containing all columns in the input data, plus the feature that is to be generated by the function. Hence, combining all such feature-generating functions, we create a pipeline aggregation that generate all features we need:

```
def gen_features(df, tickers):
    df = gen_ret(df, tickers)
    df = gen_vol(df, tickers)
    df = gen_rsi(df, tickers)
    df = gen_stok(df, tickers)
    df = gen_wilr(df, tickers)
    df = gen_macd(df, tickers)
    df = gen_obv(df, tickers)
    df = gen_retbbk(df, tickers)

    return df
```

After feature generation, since we want not one day, but multiple days of training window, we implement another pipeline function to generate time lags. Essentially, we shift all features used for analysis for k time periods, for $k = 1, \ldots, M$ where M is the maximum number of shift periods. Then, after removing the days with no observations i.e. $\max(M, \text{max window for feature generation})$, we have, at each observation, the wanted feature in the past M periods. Through this fashion, we can include all features of the past rolling window in a single observed data point.

It is worth mentioning in this section that principal component analysis as a data transformer is also considered and experimented in EDA stage, which yielded no evidence of improving model performance. Hence PCA transformation of data is excluded in the final implementation.

3.2.2 Data Ingestion Implementation

After creating the dataset, the data is split into training set and testing set, and upload onto an AWS S3 bucket. Note that it is intentional that a validation set is **not** created in this step, and the reason behind this, including a technical difficulty created by a mistake on AWS' end, will be further elaborated in the "Refinement" section, where I discuss my custom cross-validation and tuning method.

3.2.3 Training, tuning, and deployment Implementation

For the model training, simply create a XGBoost estimator that has been built in the `sagemaker.xgboost.estimator` class, and fit in some hyperparameters deemed adequate by the experiments conducted in EDA stage. Then, with the estimator returned by the finished sagemaker training job, a tuning script for the XGBoost framework `hpo.py` is created to tune the hyperparameters to enhance model performance (see next section for more details). Finally, after tuning, we deploy an estimator with the best hyperparameters onto an inference endpoint, which can be fed .csv data to forecast future market movements. Please see uploaded scripts and notebooks for details regarding the training and tuning code, which will not be included in this report for aesthetic and efficiency purposes.

3.3 Refinement

This step endured the most hardship in the progression of the project, due to a mistake on AWS' end and lack of help from Udacity (nor were there any existing resource online to treat this problem). However, this problem had been dealt with through an irregular solution.

At the very beginning, a tuning script `hpo.py` is created to attempt to tune the `XGBoost()` algorithm already-existing in the `sagemaker.xgboost.estimator` class. To implement this with respect to our classifier, we encounter two difficulties:

(1) Usually, for time series forecasting, the data is split into two folds: for some threshold time period t, $y_0, \dots, y_{t-1}$ are used for training, and y_t until the end of data is used for validation. However, we **cannot** split the training set into train and validation set for this time series forecasting project, as we aim to train a model to forecast the movement of the stock market in **one day**, and our classifier needs to work on every fold of the training data. This means that we need to split the data using similar method as the `sklearn.metric.TimeSeriesSplit`, such that the train-validation set is of structure as follows:

Train :	Validation :
$[1, 2, \dots, M]$	$[M+1]$
$[2, 3, \dots, M+1]$	$[M+2]$
$\vdots$	$\vdots$

Through time series split like this, each train-validation pair yields one classification score, and for each traverse through the dataset, the metric score is averaged through all pairs. Hence, there is no known method to define train and validation channels in S3 to my knowledge, hence the cross-validation through paired time series splits ought to happen after the data channel is passed into sagemaker.

(2) Because of aforementioned point, each traverse through the data must yield **one metric**, which no existing predefined metric can suffice. Hence, to train our model on such a structured group of train-validation pairs, we ought to define custom metrics. However, while XGBoost algorithm with framework version $\geq 1.0.0$ **should be** [7] considered as open sourced framework through which custom definitions of validation metrics are enabled, while attempting to define my own metric (cross-validation accuracy), sagemaker recognizes `sagemaker.xgboost.estimator.XGBoost` as a **built-in algorithm**, which does not accept custom definition for metrics. This would lead to the complete failure of hyperparameter tuning jobs, as the algorithm then could not rank all tuning jobs and find a "best tuning job".

The solution to this problem is as follows. Although redefinition does not work and thus "best tuning job" does not work, there exists predefined evaluation metrics with known regex expressions in the logger. For example, it is shown in sagemaker "training-job" section that metric `validation:auc` is logged in form

```
.*\[[0-9]+\].*#011validation-auc:([-+]?[0-9]*\.?[0-9]+(?:[eE][-+]?[0-9]+)?).*
```

therefore, should we manually log our own metric in strings following this regex form, we can make sagemaker record our own defined metric.

With solution to these two problems in mind, we create an entry point tuning script `hpo.py`, which accomplishes these steps:

- Import the training csv file (which contains training and validation observations) from the data channel.
- Conduct time-series split specified above. Store train-validation pairs.
- Define model. (This can only be done in XGBoost framework, since `xgboost` library is required for this step)
- Conduct cross-validation of the model.
- Log the custom metric under the regex form of another predefined metric.

To ameliorate the model, we tune hyperparameters such as gamma, minimum loss for further partition of a node, eta, a parameter controlling gradient boosting rate, and max depth, the maximum depth of a tree, which controls for overfitting problems. The parameter grids are specified in the Jupyter notebook, before fitting the tuner that utilizes aforementioned entry point script.

4 Results

4.1 Model

After tuning job, AWS Sagemaker successfully returned a model that seeks to predict market down days. The final hyperparameters we obtained are

$$\begin{aligned} \text{learning rate (eta)} &= 0.1747233886055575 \\ \text{min child weight} &= 7 \\ \text{alpha} &= 0.015533316376102427 \end{aligned}$$

which yields prediction precision of 0.764102564102564, and accuracy of 0.6457564575645757. This is extremely similar to the benchmark three days prediction accuracy measure for Microsoft stocks in the KSR paper [2]. However, since the application for this study rests in quantitative finance, this accuracy is not sufficient for test trading/industrial use. Since stock market data is a realization of time series with known distribution/pattern, unfortunately we cannot introduce random perturbations to the model like in a cross-sectional study. As mentioned in “refinement” section, the way we defined the evaluation metric (validation accuracy) has already excluded the possibility of overfitting, and has excluded the possibility of future data biases common in time series studies. Without considering the performance of the model, we have done an excellent job in ensuring that the model result is trustworthy. To sum up, although we are not satisfied with the prediction accuracy, this project study granted valuable insight into the stock market public data.

4.2 Justification and Discussion

The metric shown in the benchmark study that's closest to our purpose is a next 3 days validation forecast for MSFT, with an accuracy score of 0.645 [2]. Compared to the benchmark study by Khaidem, Saha, and Roy Dey, our accuracy is higher, but not due to a better model, but different study goals. The KSR paper tries to achieve a study (also a trading strategy) that is far more ambitious than that of ours, as predicting **both** the up and down days of the market (should that be possible at all), allows the investing agent to engage in a more frequently managed, thus riskier trading strategy which in turns has higher potential for profit – if a up day is coming, the agent buys the stock; and if a down day is coming, the agent sells/shorts the stock. If there exists no leverage limit, an agent can even keep profiting from this strategy when the market is consistently bearish.

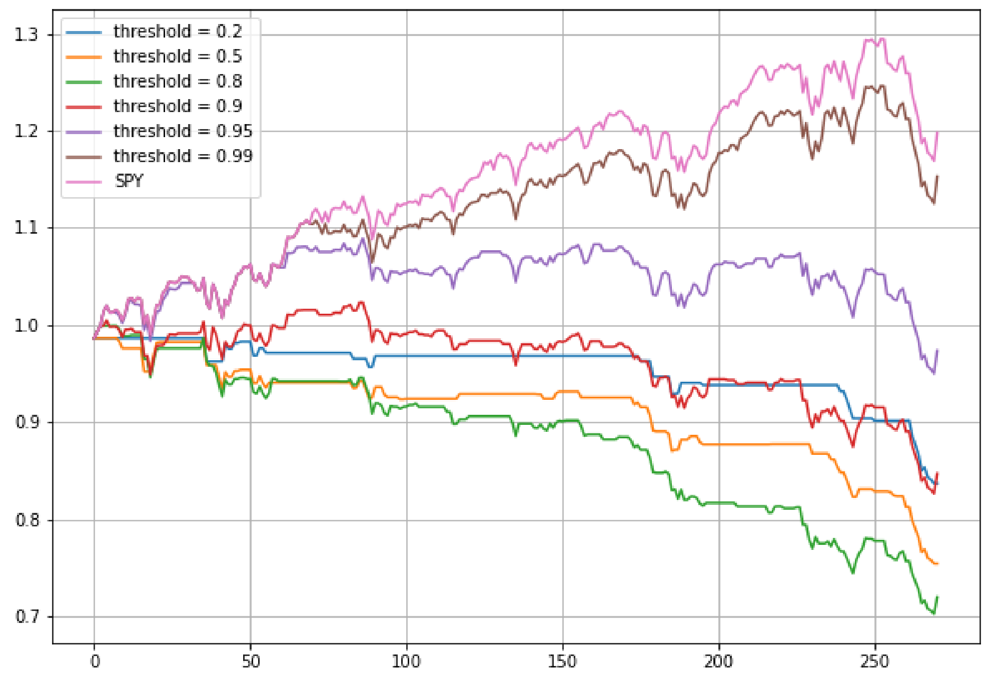
In fact, it is not sure that our model renders more information as the benchmark study did, since we are aiming for a less difficult goal. In grand view, both our study and what's mentioned in KSR's paper, are incredibly hard tasks, since the stock market is more efficiently than most market participants believe it to be. This result, while disappointing and fortunately expected, echoes well back to our inspiration – *A Random Walk Down Wall Street* by Burton Malkiel. The market is indeed very unpredictable, and the price itself contains more information than we think; and because of this efficiency, most of the arbitrage opportunities have already been traded away by other investing agents in the market.

5 Conclusion

5.1 Interesting Visuals

As mentioned at the beginning of this report, other than machine learning prediction accuracy, we are as well interested in the market performance of our model, from a quantitative finance perspective. Hence, at the end of the training/deployment notebook, we as well included a trading simulation according to our predictions: which grants a better idea re-

Figure 5: SPY Trading Simulation: 2021-1-1 to date



garding the central thesis: market returns are extremely unpredictable. This has shown that although using different probability thresholds on our prediction does yield different interesting results (crossings occurred in the simulations), none of the probability thresholds outperforms the market. Even though on some of the market down days, the model successfully avoids going down with the market, but at the same time, the wrong patterns picked up by the market as well prevents us from maximizing portfolio value in the bull market.

5.2 Reflection

From a macroscopic view, our study did not create a profitable trading strategy with our XGBoost forecaster. However, in reality, one would not have had expected to be able to make accurate forecast of the stock market – as we mentioned for numerous times, the efficiency of the stock market is often underestimated. To evaluate details we planned, small modifications we chose in defining the technical indicators (features) excluded some biases in using traditional indicators with predefined thresholds. This is a good call, as this allows for more estimator space for the model to train; on the other hand, although encountering a plethora of difficulties caused by Sagemaker XGBoost algorithm, we achieve adequately the definition and implementation of the time-series cross-validation in sagemaker script mode.

However, retrospectively, I personally think that the model selection is somewhat rushed and arbitrary. Should we use another model for this sort of time series forecast, would the result be better? Is there any other technical indicators that could potentially ameliorate the performance of the model? These are the questions to think about should we produce an extended study.

5.3 Improvement

Since we have convinced ourselves that the stock market public data barely contains predictive power, one important improvement we could make to better our study is introducing more data from different sources. For example, could oil price index be related to the stock market data? Could FED interest rate change do? Could event indicators such as the March 2020 stock market crash have introduced more information into the study? These are all very possible cases to consider.

On top of data sources, we mentioned in the last section that we could attempt another model to build a better forecaster. For example, some other models to consider for this project are recurrent neural networks (LSTM included), and random forest, which is what was used in the original benchmark study. In all and all, this project granted invaluable experience in conducting a real-life study project with AWS Sagemaker, and granted previously unknown insights regarding the public stock market data.

References

[1] Malkiel, B. G. (2003). *A random walk down wall street.* W.W. Norton & Company.

[2] Khaidem, L., Saha, S., &; Roy Dey, S. (2016, April 29). *Predicting the direction of stock market prices using random forest.* https://arxiv.org/abs/1605.00003. Retrieved January 10, 2022, from https://arxiv.org/abs/1605.00003

[3] Tin Kam Ho. (1995). Random decision forests. *Proceedings of 3rd International*

Conference on Document Analysis and Recognition.
https://doi.org/10.1109/icdar.1995.598994

[4] Breiman, L. (2001). *Machine Learning, 45*(1), 5–32.
https://doi.org/10.1023/a:1010933404324

[5] Chen, T., &; Guestrin, C. (2016). XGBoost. *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining.*
https://doi.org/10.1145/2939672.2939785

[6] Dey, S., Kumar, Y., Saha, S., &; Basak, S. (2016). Forecasting to Classification: Predicting the direction of stock market price using Xtreme Gradient Boosting. researchgate . Retrieved January 15, 2022, from
https://www.researchgate.net/profile/Snehanshu-
Saha/publication/309492895_Forecasting_to_Classification_Predicting_the_direction_of_stock_market_pric
to-Classification-Predicting-the-direction-of-stock-market-price-using-Xtreme-Gradient-
Boosting.pdf

[7] Amazon AWS. *XGBoost Classes for Open Source Version*
https://sagemaker.readthedocs.io/en/stable/frameworks/xgboost/xgboost.html