QRT Data Challenge: Stock Prediction

Samy Kobbite

Friday, October 18th

Introduction

In this project, my primary goal was to analyze and predict stock market behavior by employing machine learning techniques alongside various statistical analyses. To achieve this, I focused on extracting valuable insights from stock returns and volumes, while also integrating financial indicators such as exponential moving averages and volatility. The project's framework relies on dividing stocks into clusters based on their behavior and performance similarities. Ultimately, I aim to create predictive models for forecasting stock returns, ensuring that the methodology used is both sound and avoids overfitting.

The following report will lead you through the steps I followed and present the results of my analysis. However, for a more detailed understanding of the project, I recommend also reviewing the jupyter notebook.

Data Loading and Preprocessing

To start with, I conducted an initial exploration of the dataframes, reorganizing the entries by both stock and date. A dedicated function was implemented to ensure that no data from the test set overlapped with the training set. Furthermore, I verified that all dates were spaced by at least 20 days, ensuring that no temporal information was accessible, which was the case. After this verification, it was confirmed that there was no equivalent of "temporal" inside information for us to capitalize on.

Next, I addressed the issue of missing values. Drawing on the results from the paper by Chen and McCoy (2022) [1], *Missing Values Handling for Machine Learning Portfolios*, I adopted the cross-sectional mean method to handle missing data. This technique, as highlighted in the paper, is highly effective because it prevents look-ahead bias and generally yields results comparable to, if not better than, more sophisticated approaches. Specifically, for each date, we replaced the missing value of a stock in a particular column with the mean of the other stocks' values on that same date, thereby avoiding any temporal bias.

Following this step, we successfully filled all missing values in the test set, but encountered residual NaNs in the training set. This was due to the fact that for certain dates, entire columns were missing across all stocks. Upon identifying the affected dates, we opted to remove them rather than fill the gaps with zeros to avoid introducing bias into the model. While this may result in a slight loss of information, it ensures the integrity of the dataset and minimizes the risk of bias during model training. With this approach, we

were able to get two fully completed dataframes, ready for model training, testing, and further statistical analysis.

Adding features

After I had two filled dataframes, I decided to enrich them with some basic features that may prove useful for the models later on. The following features have been added:

- Cumulative returns
- Volatility
- Exponential Moving Average (EMA)
- Rolling volatility
- Mean of the returns and volumes
- EMA of cumulative returns
- Number of positive returns/volumes over the last 20 days

1 Data Visualization and Initial Exploration

Now that I had a well-structured dataframe, free of missing values and enhanced with new features, my next objective was to visualize the data and perform some basic statistics. I believe that data visualization is a crucial step in the process, as it allows us to start identifying potential relationships within the data, which can be explored more rigorously later on. While it may not always be methodologically strict, visualizing the data can inspire valuable insights worth investigating in depth.

To begin, I created a simple plot showing the relative returns and volumes for a few selected stocks over time. In some of these graphs, I noticed a certain correlation between the two variables, which prompted me to investigate further. This led me to consider the potential predictive power of volume data: although we won't have access to the volume on a specific date when making future predictions, we will have information about the volumes from the previous 20 days. If there is a consistent correlation between volumes and returns, analyzing the 20-day volume series might provide valuable signals that can be used to infer future returns.

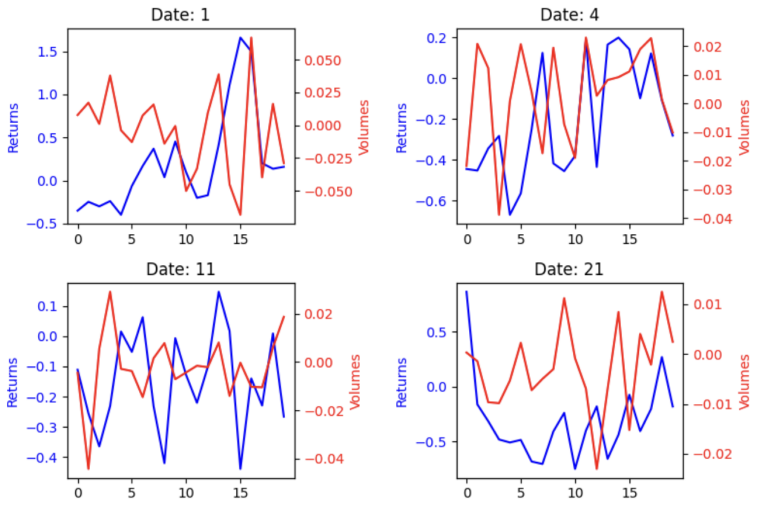

Figure 1: Returns and relative volumes

Following this idea, I calculated the correlation between relative volumes and returns for each stock at each date. Although I found a general trend for certain stocks, the correlation between volumes and returns varied significantly depending on the date.

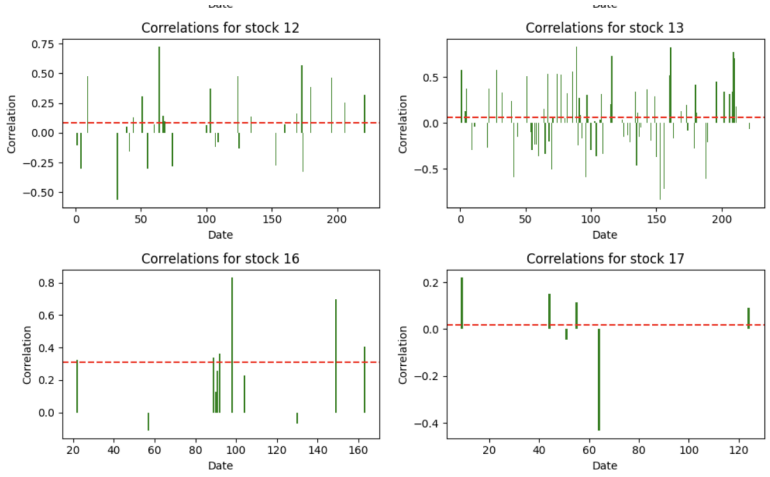

Figure 2: Correlation between the returns and the relative volumes of stocks during the last 20 days

In light of these results, I decided to create clusters of stocks that exhibited similar volume-return correlations for each specific date.

Now that I had those clusters, I moved on and tried to identify correlations within the regressor columns and the target column directly (RET).

To do this, I grouped the data based on various industry-related categories such as sector, industry, or sub-industry, and then computed the means of a specific feature, like volatility, and the means of the returns for each group. I then calculated correlations to identify any meaningful relationships. The results revealed significant correlations and anti-correlations between predicted returns and the features used, indicating that these features could enhance the model's predictive power. However, I also noticed that some features displayed similar correlation patterns with returns at the group level, meaning that I could reduce redundancy by selecting only one of these features for the final model.

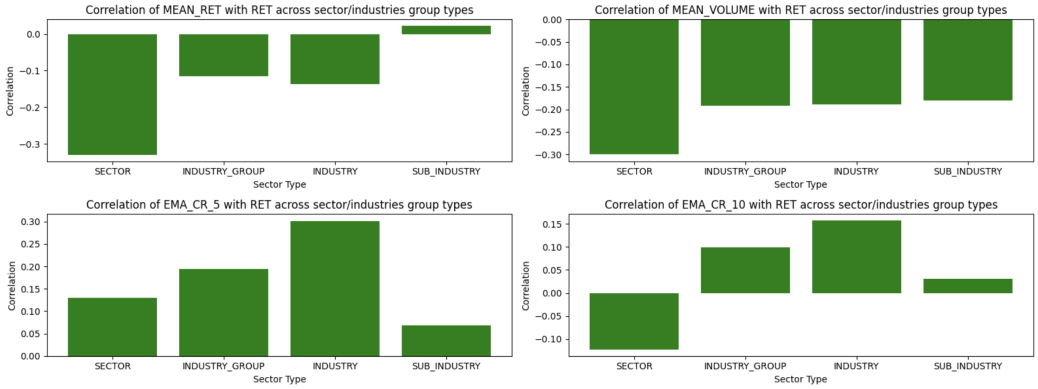

Figure 3: Correlations on aggregated levels between the final returns and different features

Also, I observed that the strength of the correlation between features and returns varied by sector and industry. I decided that I would take that into account when engineering new features. However, throughout this process, I remained mindful of the risks of introducing look-ahead bias. Indeed, I was able to identify correlations on groups of stocks (sectors/industires etc...) but I reminded myself that later, when I would aggregate data, I would have to do so on specific groups for each dates, in order to avoid any look-ahead bias.

So far, the fact that stocks within the same group, (such as those classified in the same sector or industry), tend to exhibit similar behavior had led me to consider engineering features based on these groupings, as it could prove useful for capturing common trends. However, I also recognize that stocks can be "similar" in many other ways beyond their sector/industry classification. Indeed, stocks from different sectors can exhibit similar behavioral patterns, while some stocks within the same sector may behave in

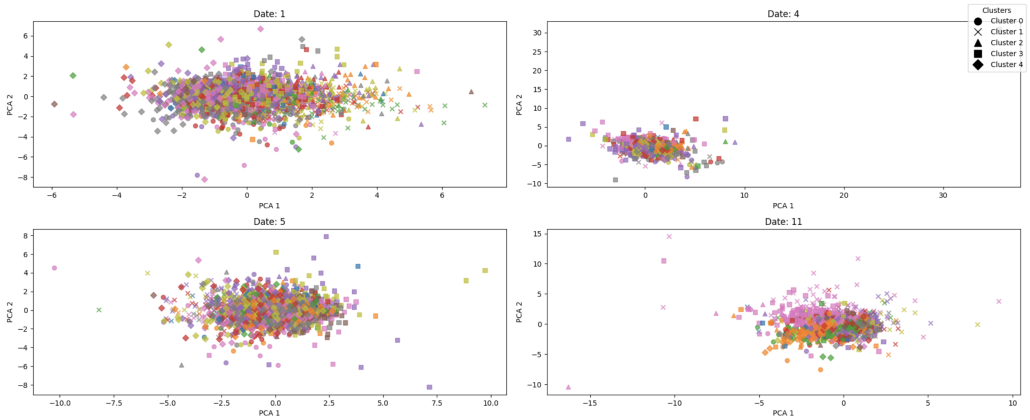

Figure 4: Comparison of classification: sectors vs returns clusters

contrasting ways. To account for this, I created dynamic clusters (based on K-means) for each date, using features such as returns, volumes, or other stock characteristics, allowing for a more flexible and data-driven approach to identifying similarities between stocks.

From the graph, I observed a correlation between the shape (representing clusters) and the color (representing sectors), indicating that both sectors and clusters capture similar information to some extent. However, this correlation is far from perfect, as some shapes appear in multiple colors and form distinct areas in the graph, separate from those defined by sector. This suggests that the clusters are capturing additional information that the sector classification alone does not. From this, I infered that clustering stocks on specific dates could help enhance the performance of my algorithm.

Feature Engineering and selection

Now that I had a complete dataframe, enriched with additional data and aggregated features by sectors, industries, sub-industries, and various types of clusters, the next step was to select the most useful data for our model to predict returns. To minimize the risk of overfitting, I focused on using aggregated features as much as possible and avoided having too many features. I was also careful not to include features that were highly correlated with each other, as this tends to destabilize the algorithm.

To determine which features to use, I tested the model multiple times with different combinations and progressively retained the features that proved to be the most explanatory. Although I could have approached this in a more systematic way, such as testing several combinations randomly, I preferred

to do it manually, as it allowed me to interpret the results at each stage. Similarly, while a PCA could have been an option, I chose not to use it due to concerns about interpretability, as I wanted to maintain clarity regarding the influence of each feature.

Model and results

In terms of model selection, I decided to retain the Random Forest as the primary algorithm due to its strong ability to capture non-linear relationships between variables. While I considered using XGBoost, which often performs well on panel data and can handle complex interactions more effectively, I observed that, in this specific case, Random Forest provided better results.

Regarding hyperparameter tuning, I chose to slightly increase the depth of the trees in the Random Forest to a maximum depth of 9. This adjustment was intended to allow the model to capture more intricate patterns and relationships in the data, which might otherwise be missed with shallower trees. However, increasing tree depth can also increase the risk of overfitting. To counterbalance this, I concurrently raised the number of trees in the forest from 500 to 600. This helps stabilize the model by averaging over a larger number of trees, reducing variance and improving generalization.

By selecting these parameters, I aimed to optimize the model's predictive power while maintaining stability and robustness in the results.

Regarding the results, below one can find the graph with features used and their order of imprtance:

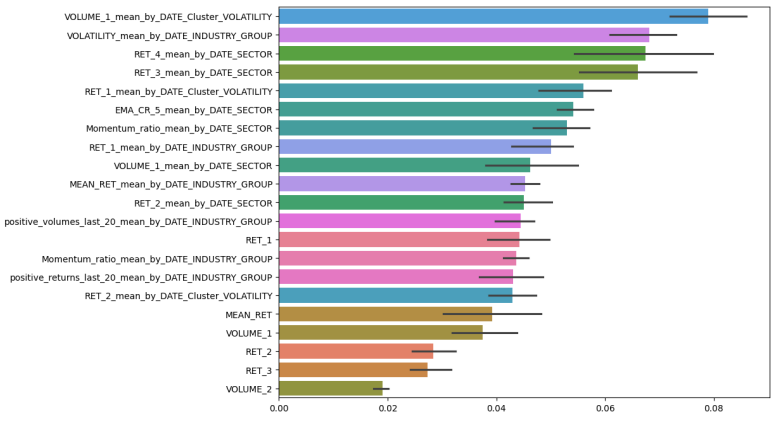

Figure 5: Features ordered by importance

Finally, in terms of accuracy, I achieved 51.93% [51.62; 52.24] ($\pm$ 0.31)

on the notebook, with a final score of 51.75% after submission, placing me 133rd out of 463 participants.

Ideas for improvement

To further improve the prediction, one potential direction would be to explore the use of a fixed-effects model. This approach could help capture unobserved heterogeneity across stocks or time periods by controlling for characteristics that remain constant over time within each entity. I tried similar local technique with XGBoost but overfitting was an issue.

Another avenue that could be interesting would be the use of a Panel Tree (P-Tree) model, on which I read an article recently. As introduced by Cong, Feng, He, and He (2022) [2]. The P-Tree model is specifically designed for handling panel data by splitting the cross-section of asset returns based on a global no-arbitrage condition, while also capturing nonlinear interactions. This model offers a balance between interpretability and predictive performance and can help reduce the risk of overfitting by leveraging a global splitting criterion. Such techniques could prove highly valuable in enhancing the robustness and accuracy of our predictive models.

References

[1] Chen, A. Y., & McCoy, J. (2022). *Missing Values Handling for Machine Learning Portfolios*. arXiv. https://doi.org/10.48550/arXiv.2207.13071

[2] Lin William Cong, Guanhao Feng, Jingyu He, and Xin He, *Asset Pricing with Panel Tree under Global Split Criteria*, SSRN, 2022. Available at SSRN: https://ssrn.com/abstract=3949463.