A Factor-Based Alpha Investment Strategy Using Machine Learning

Yiyang Zhang Zhiqi Zhou Ziyi Gao

April 2022

Abstract

With the development of the finance market and technology, the quantitative trading plays an increasingly important role in financial markets around the world. Among various trading strategies, factor-based alpha investment structures are the mostly used medium or low frequency trading strategies in stock markets. In this project, we experiment with different machine learning techniques to find an alpha factor of the stock market and evaluate them from different aspects. Moreover, we build a factor-based alpha investment strategy based on the alpha factor we obtained. The stock market we chose is the SSE 50 Index, a representative stock market index of the Shanghai Stock Exchange in China.

The final results show that Lasso Regression and Random Forest models achieve the best performance. Both models return us a factor that has mean Information Coefficient (IC) equal to 0.036 and 0.216, and Information Ratio (IR) equal to 0.232 and 1.276. Therefore, both alpha factors based on these two models have significant power in earning excess return and beating the market. In the out-of-sample back-testing experiment, based on the trading strategy built upon our models, we obtain an excess return of 10.72% and 9.15% over the simple buy-and-hold strategy on SSE 50 Index. This shows that our factor building methods through machine learning and our factor-based trading strategies have the significant power in beating the market and getting excess return.

1 Introduction

With the growth of the world economy and the development of the finance industry, the stock market investment starts to play a much more important role in people's life. The blooming in technology also leads to the appearance and progress of quantitative trading. For the stock market, after Fama and French proposed the three-factor model and five-factor model that captured the important features of stock returns[1], the factor-based alpha trading strategies started to be widely utilized in the field of quantitative trading. In this case, alpha is defined as the excess return on an investment after adjusting for market-related volatility and random fluctuations.

Nowadays, using the prediction of the stock market to outperform the market index has always been challenging. This is mainly due to the fluctuating pattern of the stock price. Factor-based alpha investment strategies, designed to manage risks within a portfolio while also delivering market-beating returns, are widely used in the quantitative trading of stocks [2] and commodities and futures market [3]. A good construction of a portfolio will help generate significant profits in stock trading. Therefore, in this project, we expect to apply various machine learning models on the stock data to build an effective alpha factor for the stocks, which can be used in the valid factor-based alpha trading strategies. Our prediction goal for the machine learning models is the return rate after the next five days (5 day's return) of the stock. Based on the trained models and the predicted return rate, we can further build an effective alpha-factor for the stocks and use our trading strategies to construct a good portfolio that can give us an excess return over the market.

2 Data

2.1 Data Description

We use the daily data of individual component stocks of the Shanghai Stock Exchange (SSE) index from Jan 5th, 2015 to Mar 2nd, 2022, which represents the Top 50 companies by "float-adjusted" capitalization and other criteria. The data comes from the Wind-Financial Terminal [4] - one of the most authoritative and leading provider of financial information services in China. The dataset originally has 175,417 rows of data combined from 111 individual stock datasets (because of the adjustments of the index components in history) with 13 variables in total, including time, open price, close price, highest price, lowest price, pre-close price, the change amount of the close price, the change rate of the close price, the turnover rate, trading volume, trading amount, market value and the liquid market value of the stocks.

Figure 1 shows the price trend of the SSE 50 Index in our selected period.

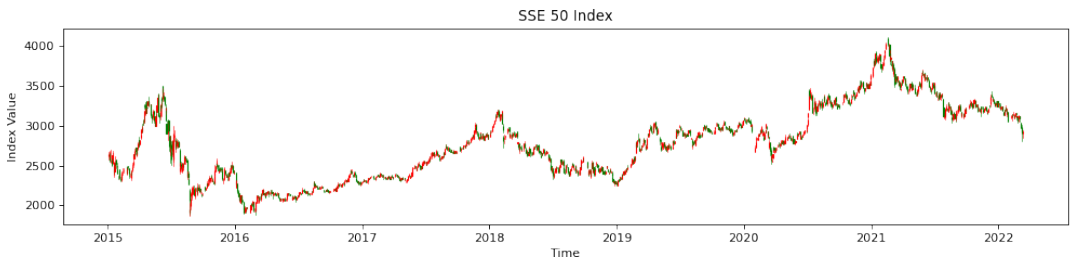

Figure 1: Trend of SSE 50 Index

2.2 Exploratory Data Analysis

Since it is hard to conduct exploratory data analysis on the entire stock market, we decide to select two representative stocks as our objects of exploratory data analysis. In this section, we decide to use stocks 600000.SH and 600519.SH as our main focus.

As shown in Figure 2, we first plot the price trend of these two stocks. According to Figure 2, we find

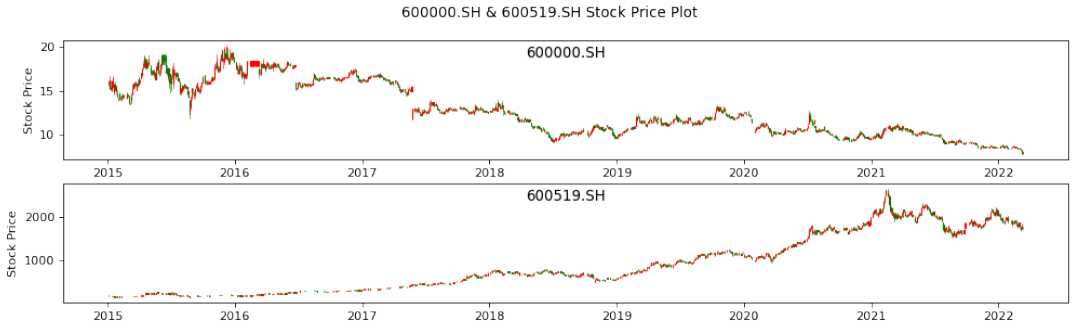

Figure 2: Example Stock Price Plot

that stock 600000.SH followed a downward trend and stock 600519.SH followed a upward trend between Jan 5th, 2015 and Mar 2nd, 2022. These two stocks typically show types of stocks that we want to exclude (600000.SH) and include (600519.SH) in our constructed portfolio. In our portfolio, we want to have the proportion of stocks with upward trend as large as possible to guarantee our portfolio having an excess return over the market.

Next, we build histogram plots for our prediction target - future 5 day's return rate. From Figure 3 we can see that, for stock 600000.SH, the mean of the distribution of target is smaller than 0. Its distribution approximately follows a slightly left-skewed distribution, which is consistent with the trend of it. For stock 600519.SH, the mean of the distribution of target is larger than 0, but it seems to be symmetrically distributed. This is reasonable as compared to the trend of stock 600519.SH. However, there might be some hidden information behind the distribution of the stock.

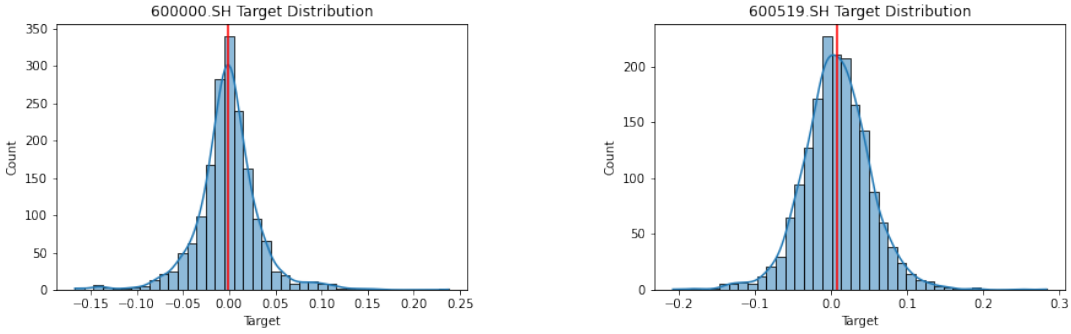

Figure 3: Distributions of Future 5 days returns

In order to dig up more hidden information behind the stocks, we check the correlation between different features of these two stocks. Correlation heat-maps for the two stocks are presented in Figure 4.

From Figure 4, we find that for both stocks, the features which belong to the same type are highly

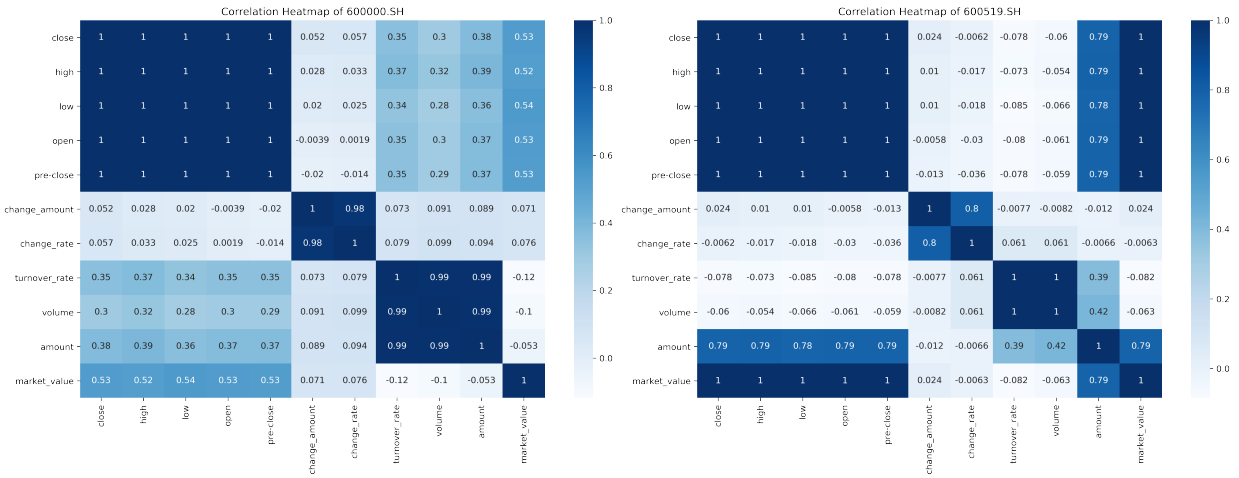

Figure 4: Correlation of Basic Features of Two Selected Stocks

correlated. For example, the correlations between close price and open price of the stock are 1, since open price and close price are very similar to each other for each stock. There are indeed some different correlation patterns for the features of these two stocks, which could be explained by the discrepancy of their trends.

2.3 Data Preparation

2.3.1 Adding Features

We first remove all null values in the dataset. After data cleaning, we have 152,789 rows of data. Then, we manually calculate and add some features to our data. By adding features, we want to insert some experimentally useful information into the dataset, which can help us build better models for more accurate predictions. We add 11 variables in total, including future 5 day's return rate, 5, 10, and 25-day period momentum, 5-day Volume-Weighted Average Price (VWAP), 10-day Relative Strength Index (RSI10), and some other technical analysis features of the stock. Detailed formulas for how we calculate each variable are explained in Appendix A.

Now, we conduct an exploratory data analysis for our added features by using violin plots as shown in Figure 5. From Figure 5, we find that our added features have different patterns of distribution. However, the distributions of features in the same category, like Moving Average and Momentum categories, are very similar. Overall, we hope these added features with different distributions can help our models capture more useful information during the process of model training. Based on the former researcher's experience [2], such kind of feature engineering is helpful.

2.3.2 K-means Clustering

Since our data is time-series data, we split the dataset into training and test set by time. The training set starts from Jan 5th, 2015 to Feb 26, 2021. The test set starts from Mar 1st, 2021 to Mar 2nd, 2022. In the following clustering and model training processes, we only use the data from the training set.

Before we start to train our models, we use a clustering algorithm to categorize the stocks into different

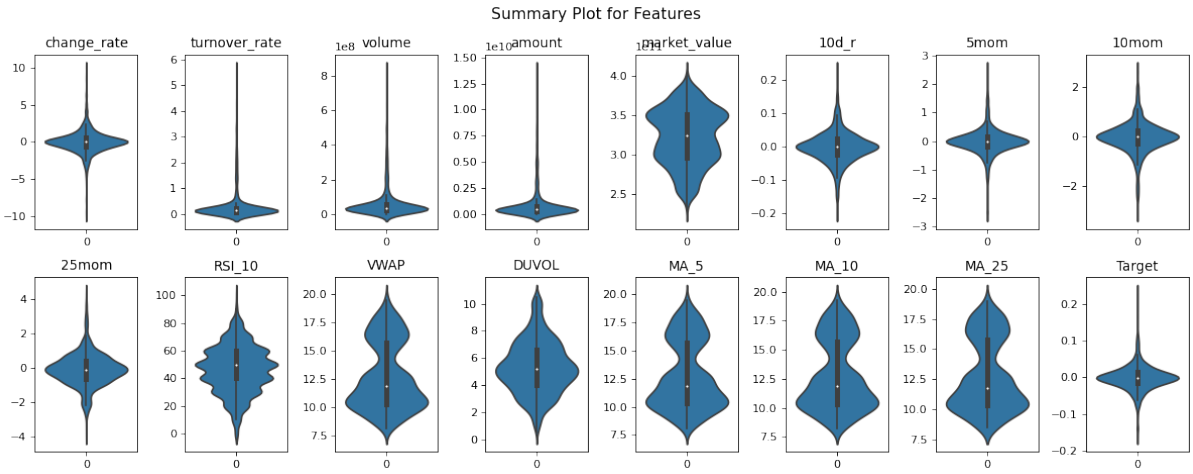

Figure 5: Exploratory Data Analysis for Added Features

groups. In some time series forecasting studies, clustering has usually been combined with the prediction [5] [6]. By clustering these 111 stocks, we expect to group stocks with similar trends of return into one cluster. Through clustering, we hope that a more suitable model for each cluster of stocks will help us achieve better prediction performances. Therefore, we choose to use the K-means algorithm to cluster the stocks based on the correlations of their 5 day's return. K-means algorithm is a widely known and easily implemented unsupervised learning algorithm. We choose the number of clusters to be 5, since we will build a portfolio that contain 5 stocks from the SSE 50 Index market. As a result, we finally combine all individual stock data from the same cluster, and the detailed information about each cluster is provided in Table 1. In the future model training part, we will use these combined datasets for each cluster rather than the original individual stock data.

	Cluster 0	Cluster 1	Cluster 2	Cluster 3	Cluster 4
Number of Stocks	16	25	25	10	35
Number of Samples	22,177	34,145	37,425	9,401	49,641

Table 1: K-means Clustering Result 1

According to Figure 6, we can view our K-means clustering results in 3D-plot. Three axes x, y, z are three columns of randomly chosen correlations between samples. The scatter points show the correlation among the data, and different colors denote different clusters. We observe that the clusters are almost well-separated with a little overlapping in three dimensions.

Therefore, we could assume that our different clusters represent stocks with different patterns of return rate. Next, we plot the price trend of the stocks in each cluster to see if their return patterns are indeed different from each other.

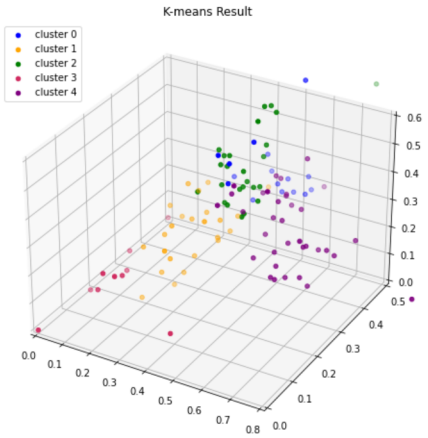

Figure 6: K-means Clustering Result 2

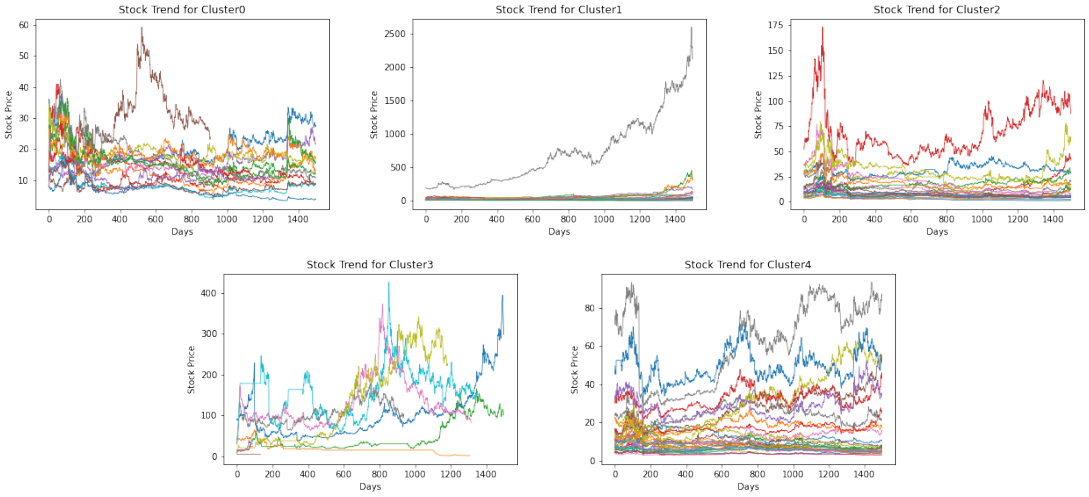

Figure 7: Stock Trends in Different Clusters

From Figure 7, we find that the trend of stock prices and return patterns in each cluster seem to be significantly different from each other. These different return patterns captured by the K-means algorithm could be very helpful for our models to improve accuracy in the prediction tasks.

3 Experiment

3.1 Model Training

We fit different machine learning models to each cluster based on the clustering result in Section 2.2. Our candidate models are Lasso Regression, Random Forest, XGBoost, and Neural Network. Details and reasons why we choose these four models are explained as the following:

(a) Lasso Regression: Since we have 25 variables and some of variables are based on the calculation of other variables, then the data may exist multicollinearity issue. By using Lasso Regression, it allows us to shrink some coefficients to avoid overfitting and do the feature selection automatically.

(b) Random Forest: Random Forest Regression is an ensemble learning method built upon the idea of bagging [7]. It combines predictions from multiple machine learning algorithms to help us achieve better predictions than just a single model. Besides, it controls overfitting by randomly selecting variables to build a tree [7].

(c) XGBoost: XGBoost is also an tree-based ensemble learning algorithm, which is an implementation of gradient boosting. We choose XGBoost over AdaBoost since it provides parallel tree boosting [8], and its computational speed is fast when compared to other implementations of gradient boosting [9]. Furthermore, it is commonly used on large datasets.

(d) Neural Network: Neural Network is a very powerful model reflecting the human brain. It can recognize hidden patterns and relationships in the data, which is broadly used in financial operations and trading [10]. In this project, we build a Multilayer Perceptron as our prediction model.

We use grid search 5-fold cross validation methods to tune the hyperparameters in each model except Neural Network. Due to the computational complexity of the Neural Network and our limited computation resources, we only randomly input some parameter values and use 5-fold cross-validation to determine the best model.

3.2 Factor Building and Performance Analysis

Using the trained XGBoost, Random Forest, and Lasso Regression Models in Section 3.1, we obtain the predicted 5 day's return for the stock on each day in the training set. In this section, we incorporate industry information by adding another variable "Industry" to our training data. Subsequently, we fit a linear regression model on the predicted 5 day's return for each cluster to do the neutralization on industry factor and market value factor. This will increase the generality of our models to different types of stock, particularly to stocks with different market values and industry types. The linear regression model is:

$$\text{Predicted 5 day's Return} = \beta_0 + \beta_1 \text{I(Industry)} + \beta_2 \text{Market Value}.$$

Denote the predicted 5 day's return from our machine learning model as y_i, and the predicted 5 day's return from our linear regression model as $\hat{y_i}$. As $\hat{y_i}$ can be interpreted as a part included in y_i that can be explained by the industry factor and market value factor, the proportion of these parts can be explained by using R^2. Therefore, the residual $\epsilon_i = y_i - \hat{y_i}$ stands for the part in y_i that cannot be illustrated by industry factor and market value factor. We will treat these residuals ϵ_is as our final factors.

After we get the final factors, we use Information Coeffecient (IC) and Information Ratio (IR) to evaluate their performances.

- Information Coefficient (IC): IC $\in [-1, 1]$ represents the correlation between predicted and realized values and is usually used to evaluate forecasting ability (i.e., stock picking ability) [2][3]. The greater the absolute value of IC, the better the prediction ability.

  $$\text{IC}_t = \text{correlation}(\vec{f_t}, \vec{r}_{t+1})$$

  Denote t is our position adjustment period. And we have that:

  - IC_t is the factor's IC value at period t,
  - $\vec{f_t}$ is the predicted factor vector of t period on $t+1$ period return rate
  - $\vec{r}_{t+1}$ is the real return vectors for stocks at period $t+1$.

- Information Ratio (IR): IR represents the factor's ability to obtain stable alpha [2][3].

  $$IR = \text{mean}(IC_t)/\text{std}(IC_t), \text{during the chosen multiple periods of time.}$$

Usually, $|\text{IC}| \geq 0.03$ indicates that the factor is effective, and $\text{IR} \geq 0.5$ indicates that factor has a stable power in earning excess alpha return.

3.3 Out-of-sample Back-testing

Our back-testing period starts from Mar 1st, 2021 to Mar 2nd 2022. Our trading strategy is defined as stated in Algorithm 1. In our algorithm, the trading cycle is each 5 days. We also ignore the transaction

costs of each trading.

Algorithm 1 Alpha Factor Based Trading Strategy Back-testing Algorithm

Input: Initial Money M; Machine Learning Model for each Cluster ML_i; Regression Model for each Cluster R_i; Individual Stock data S_j. $i = 0, 1, 2, 3, 4$, and $j = 0, 1, ..., 110$.

Output: Final Value of Portfolio W;

1: Select stocks S_js that in cluster i;
2: At time t, use $F_j = ML_i(S_j) - R_i(S_j)$ to get the final neutralized factor F_j for all stocks S_js
3: Sort the F_js we got at time t;
4: If IC of this factor greater than 0 in training period, we will buy the stocks with biggest five F_js as our portfolio that holds for period t to $t+5$. Else, we will buy the stocks with smallest five F_js as our portfolio that holds for period t to $t+5$.
5: Divide M equally to buy the 5 chosen stocks at day t and sell them at the day $t+5$. Each stock has an return rate $r_k, k = 0, 1, 2, 3, 4$;
6: Now, present portfolio value at time $t+5$ is $M^* = M \times (1 + \frac{\sum r_k}{5})$
7: Update $M = M^*$ and $t = t+5$, continue the process 2 to 6 till the end of our back-testing period
8: **return** The final value of our portfolio $W = M$;

4 Results

4.1 Model Selection

We use Mean Squared Error (MSE) as the metric to evaluate our training models. As shown in Table 2, we find that the best XGBoost candidate model after parameter tuning has the lowest MSE on all clusters. However, Random Forest models have comparable and similar performances as XGBoost. Besides, Lasso Regression performs slightly worse than Random Forest and XGBoost models. Neural Network models have the highest MSE among these four models. Therefore, in the process of factor-building and out-of-sample back-testing, we will use Lasso Regression, Random Forest, and XGBoost models.

	Training Mean Squared Error (MSE) of Best Model				
	Cluster 0	Cluster 1	Cluster 2	Cluster 3	Cluster 4
Lasso	0.004165	0.004770	0.004093	0.008465	0.002199
Random Forest	0.003611	0.004643	0.003836	0.005529	0.002020
XGBoost	0.003428	0.003396	0.003371	0.005020	0.001949
Neural Network	0.017270	0.010680	0.010677	0.021523	0.006145

Table 2: Best Model Performance Obtained From Cross-Validation

4.2 Factor Building and Performance Analysis

Table 3 shows the regression R^2 and Adjusted R^2 for different machine learning models under each cluster. We observe that the prediction results obtained from XGBoost and Random Forest have low R^2 and Adjusted R^2. This implies that both models use few information from the industry factor and market value factor, which is what we prefer. By contrast, the model from Lasso Regression has very high R^2 and Adjusted R^2 in some clusters. Therefore, the neutralization process of the industry factor and market value factor is very necessary for such model in order to make it suitable for other new stocks.

From Table 4, we find that, based on the IC and IR evaluation criteria explained at the end of Section

		Linear Regression Results				
		Cluster 0	Cluster 1	Cluster 2	Cluster 3	Cluster 4
XGBoost	R^2	0.0411	0.0059	0.0161	0.0232	0.0056
	Adjusted R^2	0.0410	0.0055	0.0157	0.0227	0.0054
Random Forest	R^2	0.0478	0.0942	0.0396	0.0375	0.0214
	Adjusted R^2	0.0476	0.0938	0.0392	0.0370	0.0212
Lasso	R^2	0.6831	0.1254	0.4968	0.6275	0.1995
	Adjusted R^2	0.6831	0.1251	0.4966	0.6273	0.1993

Table 3: Regression Result for Industry and Market Value

3.2, all factors from these model are effective stock selection factors. Nevertheless, the stability of factor we got from Lasso Regression is not good compared to other models. This could be explained by the reason that after the neutralization process of industry factor and market value factor, some important information is lost since we have a very high R^2 result for Lasso Regression as shown in Table 3.

	Mean IC	**IR**
Lasso Regression	0.036	0.232
Random Forest	0.216	1.276
XGBoost	0.283	1.200

Table 4: Mean IC and IR for Different Models

Moreover, Figure 8 presents the distribution of IC values of factors from different models during the training period. We find that each of them has a positive mean and thus these factors are all positively correlated with the return of the stocks. Besides, they seem to be symmetrically distributed except XGBoost which is heavily left skewed. This left skewness might lead to the loss of efficiency of the factor, since the positive correlation between this factor and future return might not as good as the estimated in the out-of-sample test.

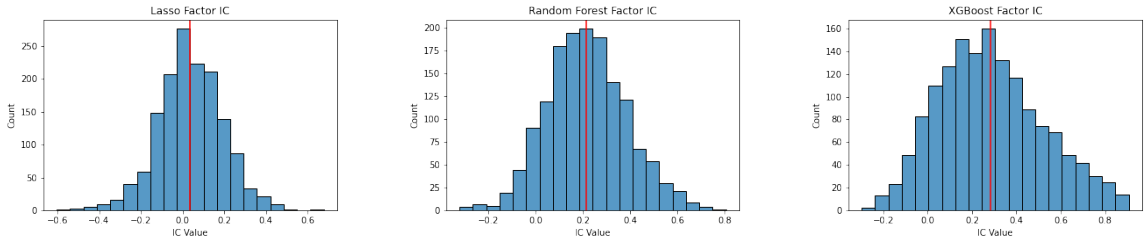

Figure 8: Factor IC Distributions for Different Models

4.3 Out-of-sample Back-testing Performance

During the back-testing period, our money starts from $M = \text{RMB } 1,000,000$, the benchmark "Index" is just use all the money M to buy the SSE 50 Index ETF and hold it until the end of the back-testing period.

For our own constructed portfolios based on the trading Algorithm 1, we obtain the following results in Table 5. From Table 5, we observe that our portfolios built upon Lasso Regression factor and Random Forest factor have significant excess return over the SSE 50 Index. However, the portfolio built based on XGBoost is not performing well, which could be explained by the left-skewness we discussed previously or the issue of overfitting.

	Index	Lasso	XGBoost	Random Forest
Annualized Return	-18.09%	-7.37%	-17.11%	-8.93%
Annualized Excess Return	0	10.72%	0.98%	9.16%

Table 5: Back-Testing Results

The portfolio values can be visualized in Figure 9. Interestingly, we notice that the excess return are mainly obtained during the period when the SSE 50 Index does not have a downward trend. Therefore, our factor-based alpha investment strategies based on machine learning algorithms might be more suitable to get a excess return over the market index in a bull market.

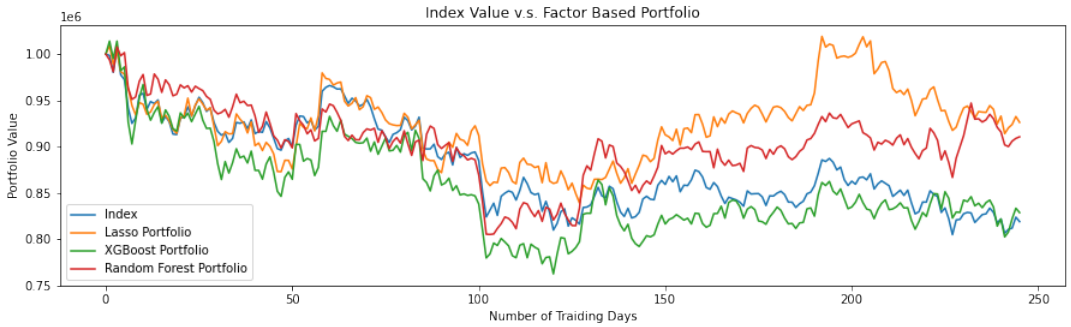

Figure 9: Back-testing Results for Different Portfolios

5 Conclusion

From the results of out-of-sample back-testing in Section 4.3, we conclude that Lasso Regression and Random Forest models achieve the best performances among these four portfolios. Even though XGBoost models have the best results on the training set, it performs much worse than Lasso Regression and Random Forest during the back-testing period. This might suggests the overfitting issue of XGBoost models.

For future work, we can consider updating our models on a rolling basis. For example, we can split the back-testing period into several 3-month periods. We can update our training data to the most recent 1-year data every three months to train new models and obtain newly predicted return rates. Then, we can recalculate IC and reselect the stocks for our portfolio. Moreover, we can apply principal component analysis (PCA) for dimension reduction since we have 25 variables in total, and based on the EDA, some of them are highly correlated. We can compare the results if we use PCA with those if we do not use PCA and verify if the use of dimension reduction can help us achieve better results. Last but not least, during model training, we can try some other machine learning algorithms, such as Support Vector Machine and Long Short-term Memory Networks.

6 Contribution by Group Members

All the information searching, coding, experiments, and report writing are done by discussion of 3 group members. Since some of work are done by us together, we will only list the greatest contributions of each group member:

- Yiyang Zhang: Coding of Data Preprocessing, EDA, Factor Building, and Out-of-sample Backtesting. Writing of Section 2.2, 2.3.1, 3.2, 3.3, 4.2, 4.3
- Zhiqi Zhou: Coding of Model Training, final code inspections and revisions, presentation slides making, and project presenting. Writing of Section 1, 2.1, 2.3.2, 3.1, 3.2, 4.1.
- Ziyi Gao: Coding of Modeling Training and EDA as well as presentation slides making. Writing of Abstract and Section 5.

References

[1] Eugene F. Fama and Kenneth R. French. A five-factor asset pricing model. *Journal of Financial Economics*, 116:1–22, 4 2015.

[2] Huatai securities artificial intelligence series no. 23: Revisiting stock selection factor mining based on genetic programming. 2019.

[3] Hwabao securities: Special report on financial engineering - commodities + cta + multi-factor model construction and backtesting. 2020.

[4] Wind financial terminal. `https://www.wind.com.cn/en/wft.html`.

[5] Man Li, Ye Zhu, Yuxin Shen, and Maia Angelova. Clustering-enhanced stock price prediction using deep learning. *World Wide Web*, 2022.

[6] Hongxing He, Jie Chen, Huidong Jin, and Shu-Heng Chen. Trading strategies based on k-means clustering and regression models. *Computational Intelligence in Economics and Finance*, page 123–134.

[7] Prof. Liza Levina. Lecture notes in ensemble classifiers, February 2022.

[8] What is xgboost? `https://www.nvidia.com/en-us/glossary/data-science/xgboost/`.

[9] Jason Brownlee. A gentle introduction to xgboost for applied machine learning. `https://machinelearningmastery.com/gentle-introduction-xgboost-applied-machine-learning/`, Feb 2021.

[10] James Chen. Neural network definition. `https://www.investopedia.com/terms/n/neuralnetwork.asp`, Feb 2022.

Appendix A Added Features

- Target: return of the stock in future 5 days. Target = $(\text{Close}_{t+5} - \text{Close}_t)/\text{Close}_t$
- 5, 10 and 25 day period momentum:
  - $adj\_\text{close} = (\text{High} + \text{Low} + \text{Close})/3$
  - $nday-\text{momentum} = adj\_close_{\ t} - adj\_close_{t-n}$
- 5 day Volume-Weighted Average Price(VWAP):
  - Typical price = (High + Low + Close)/3
  - VWAP = $\sum$(Typical Price x Volume) / $\sum$(Volume)
- Stock 10 days' return: $r_{10} = (\text{Close}_t - \text{Close}_{t-10})/\text{Close}_{t-10}$
- 10 DAY Relative Strength Index (RSI10): $RSI_{10} = \frac{\text{days with positive return in recent 10 days}}{10} \times 100$
- 25 day Down to up volatility(DUVOL):
  - $\text{DUVOL}_i = \log\left(\frac{(n_u-1)\sum_d(r_{it}-\bar{r}_l)^2}{(n_d-1)\sum_u(r_{it}-\bar{r}_l)^2}\right)$
  - n_u is the number of days greater than the average compound rate of return
  - n_d is the number of days less than the average compound rate of return
  - r_{it} Is the daily rate of return of stocks
  - $\bar{r}_l$ Is the 60-day average return rate of a stock
- 5, 10, 25-day Moving Average of Close Price: find the average of close prices for a 5/10/25-day period.

Appendix B Code

For full codes and datasets, you can access them at https://github.com/YiyangZhang0201/STATS503-FINAL.

```
# # DATA PREPROCESSING
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import math
import cufflinks as cf
from tqdm import tqdm
import chart_studio
import re
import urllib.request
import os
chart_studio.tools.set_credentials_file(username='zhyiyang', api_key='bwSqRbNvLD1oZ8xYIiQF')
from plotly.offline import iplot, init_notebook_mode
init_notebook_mode()

# ## Individual Stock Data
index_component_start = pd.read_excel("data/20150101-20150520.xlsx", sheet_name="")
```

```
stocks = index_component_start["  "].to_list()
index_component_change = pd.read_excel("data/index_adjustment.xlsx")
stocks_ch = index_component_change["  "].to_list()
index_component = list(set(stocks + stocks_ch))
len(index_component)

# Now, we can get the data of each stock and do the data cleaning.
def data_cleaning(data, filename):
    # drop useless columns
    data.drop(["    "], axis=1, inplace=True)
    data.drop("Unnamed: 0",axis=1, inplace=True)
    # change column names
    data.columns = ["time", "code", "close", "high", "low", "open", "pre-close",
                "change_amount", "change_rate", "turnover_rate", "volume",
                "amount", "market_value", "l_market_value"]
    # deal with the missing value because of the stop trading of individual stocks
    # the close, open, high, low during the stop trading is same as the previous
    for i in ["close", "open", "high", "low"]:
        data[i].replace(0, np.nan, inplace=True)
        data[i] = data[i].fillna(method='ffill')
    # change amount and change rate during the stop trading time is 0
    data["turnover_rate"].replace(np.nan, 0, inplace=True)
    data['change_amount'].replace(np.nan, 0, inplace=True)
    data['change_rate'].replace(np.nan, 0, inplace=True)
    # other variables remains 0
    data.to_csv(f"{filename}.csv", index=False)

for i in tqdm(index_component):
    data = pd.read_csv(f"stocks/sh{i[:6]}.csv")
    data_cleaning(data, i[:6])
```

```
# # DATA PREPROCESSING Part 2
# ## Add New Features to data
def add_features(data):
    # Target
    data["Target"] = (data["close"] - data["close"].shift(5))/data["close"].shift(5)
    data["Target"] = data["Target"].shift(-5)

    # 10 day's return
    data["10d_r"] = (data["close"] - data["close"].shift(10))/data["close"].shift(10)

    # typical/adjusted close price
    data["typical_P"] = (data["high"] + data["low"] + data["close"])/3

    # 5   10 and 25 day period momentum
    data["5mom"] = data["typical_P"] - data["typical_P"].shift(5)
    data["10mom"] = data["typical_P"] - data["typical_P"].shift(10)
    data["25mom"] = data["typical_P"] - data["typical_P"].shift(25)

    # RSI10
```

```
data["RSI_10"] = np.NaN
for i in range(len(data)-10):
    count = 0
    for k in range(10):
        if data["close"].loc[i+k] > data["pre-close"].loc[i+k]:
            count += 1
    data["RSI_10"].loc[i+10] = count/10*100

# 5 day VWAP
data["mid_V"] = data["typical_P"] * data["volume"]
data["upvwap"] = data["mid_V"].rolling(5).sum()
data["VWAP"] = data["upvwap"]/data["volume"].rolling(5).sum()

# 5 day DUVOL
data["count"] = np.NaN
data["change_r_p"] = np.NaN
data["change_r_d"] = np.NaN
data["mr_60"] = data["change_rate"].rolling(60).mean()

for i in range(len(data)):
    if data["close"].loc[i] >= data["pre-close"].loc[i]:
        data["change_r_p"].loc[i] = data["change_rate"].loc[i]
        data["change_r_d"].loc[i] = data["mr_60"].loc[i]
    else:
        data["change_r_p"].loc[i] = data["mr_60"].loc[i]
        data["change_r_d"].loc[i] = data["change_rate"].loc[i]

data["mr_sq_p"] = (data["change_r_p"] - data["mr_60"])**2
data["mr_sq_d"] = (data["change_r_d"] - data["mr_60"])**2

for i in range(len(data)-25):
    count = 0
    for k in range(25):
        if data["close"].loc[i+k] > data["pre-close"].loc[i+k]:
            count += 1
    data["count"].loc[i+25] = count
data["DUVOL"] =
    np.log((data["count"]-1)*data["mr_sq_d"].rolling(25).sum()/(25-data["count"]-1)*data["mr_sq_p"].rolling(25
data["DUVOL"].replace(-float('inf'), -100)
data["DUVOL"].replace(float('inf'), 100)
data["DUVOL"].replace(np.NaN, 0)

# 5, 10, 25 Moving average
data["MA_5"] = data["close"].rolling(5).mean()
data["MA_10"] = data["close"].rolling(10).mean()
data["MA_25"] = data["close"].rolling(25).mean()

# drop useless columns for future steps
data.drop(["typical_P", "mid_V", "upvwap", "mr_60", "mr_sq_p", "mr_sq_d",
          "change_r_p", "change_r_d", "count"], axis=1, inplace=True)
return data
```

```
index_component_start = pd.read_excel("data/20150101-20150520.xlsx", sheet_name="")
stocks = index_component_start["  "].to_list()
index_component_change = pd.read_excel("data/index_adjustment.xlsx")
stocks_ch = index_component_change["  "].to_list()
index_component = list(set(stocks + stocks_ch))

# Deal with NaN and Inf
def delete_inf(data):
    data.replace([np.inf, -np.inf], np.nan, inplace = True)
    data.replace(np.nan, 0, inplace = True)
    return data

for com in tqdm(index_component):
    data = pd.read_csv(f"cleaned_stock/{com[:6]}.csv")
    data_f = add_features(data)
    data_f = delete_inf(data_f)
    data_f.to_csv(f"featured_stock/{com[:6]}.csv", index=False)
```

```
# # Training Data Preparation (With K-means)
#
# Now we prepare the training data for model.
# ## Use K-means to cluster the stocks
# We first use K-means to cluster the stocks based on their return type. This can help us to
    build more suitable model for each clustered category of stocks.

cluster_data = pd.DataFrame()
for com in tqdm(index_component):
    data = pd.read_csv(f"featured_stock/{com[:6]}.csv")[["time", "code", "Target"]][:1497]
    data['time'] = pd.to_datetime(data['time'])
    cluster_data = pd.concat([cluster_data, data], axis=0)
cluster_data.reset_index(drop=True)

# Now we start the K-means clustering, and cluster the stocks into 5 categories.
from sklearn.cluster import KMeans
cluster_p = cluster_data.pivot(index='time', columns='code', values='Target')
cluster_p = cluster_p.sort_index()

corr = cluster_p.corr()
corr = corr.replace(np.NaN, 0)
ids = corr.index

kmeans = KMeans(n_clusters=5, random_state=615).fit(corr.values)
print(kmeans.labels_)

from mpl_toolkits.mplot3d import Axes3D
plt.rcParams['axes.facecolor']='white'

fig = plt.figure(figsize=(5,5))
ax = Axes3D(fig)
```

```
ax.axes.set_xlim3d(left=0.0, right=0.8)
ax.axes.set_ylim3d(bottom=0.0, top=0.5)
ax.axes.set_zlim3d(bottom=0.0, top=0.6)
ax.scatter3D(corr.iloc[0], corr.iloc[10], corr.iloc[100], c = kmeans.labels_) plt.title("K-means
    Result")
plt.savefig("plot1")

## add each stock to its corresponding category
stock_list1 = []
stock_list2 = []
stock_list3 = []
stock_list4 = []
stock_list5 = []
for i in range(len(kmeans.labels_)):
    if kmeans.labels_[i] == 0:
        stock_list1.append(ids[i])
    elif kmeans.labels_[i] == 1:
        stock_list2.append(ids[i])
    elif kmeans.labels_[i] == 2:
        stock_list3.append(ids[i])
    elif kmeans.labels_[i] == 3:
        stock_list4.append(ids[i])
    elif kmeans.labels_[i] == 4:
        stock_list5.append(ids[i])
stock_type = [stock_list1, stock_list2, stock_list3, stock_list4, stock_list5]


# Now, we start to prepare training data for model building based on the stock types.
def training_data_type(type):
    """
    merge the training dataset based on stock types
    """
    training_data = pd.DataFrame()
    for i in stock_type[type]:
        data = pd.read_csv(f"featured_stock/{i}.csv").drop(["time", "code"], axis=1)[:1497]
        data = data.dropna(axis=0)
        training_data = pd.concat([training_data, data], axis=0)
    training_data.reset_index(drop=True)
    return training_data

# Store the training data.
for t in [0,1,2,3,4]:
    train = training_data_type(t)
    train.to_csv(f"TrainingData/StockType{t}.csv", index=False)

Store the type information got by cluster.
for i in [0,1,2,3,4]:
    np.savetxt(f'TrainingData/stocktype{i}.txt',stock_type[i],delimiter="\n", fmt="%s")
```

```
## Model Training
```

```
import pickle
import torch
import torch.nn as nn
import xgboost as xgb
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import GridSearchCV
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error
from xgboost.sklearn import XGBRegressor
from sklearn.linear_model import Lasso
from sklearn.preprocessing import MinMaxScaler

def prepare(data):
    x_train = data.loc[:, data.columns != 'Target']
    y_train = data["Target"]
    return x_train, y_train

## Lasso Regression
# regularization parameter for Lasso
param_grid = {
    'alpha': [0.1, 0.2, 0.3, 0.5, 0.6, 0.7, 0.8]
}

def fit_lasso(x_train, y_train):
    lasso = Lasso()
    CV_lasso = GridSearchCV(estimator = lasso, param_grid = param_grid,
                      scoring = "neg_mean_squared_error",
                      cv = 5, verbose = 2)
    CV_lasso.fit(x_train, y_train)
    best_grid = CV_lasso.best_estimator_
    return best_grid

clusters = list(range(5))
for t in tqdm(clusters):
    data = pd.read_csv(f"TrainingData/StockType{t}.csv")
    #data = delete_inf(data)
    x_train, y_train = prepare(data)
    lasso_model = fit_lasso(x_train, y_train)
    with open(f'lasso_cluster{t}', 'wb') as files:
        pickle.dump(lasso_model, files) ## store the best model for each cluster

# ### Best Lasso model Performance
for i in range(5):
    data = pd.read_csv(f"TrainingData/StockType{i}.csv")
    #data = delete_inf(data)
    x_train, y_train = prepare(data)
    loaded_model = pickle.load(open(f'lasso_cluster{i}', 'rb'))
    print(loaded_model)
    y_pred = loaded_model.predict(x_train)
    print(f"Mean squared error (Cluster {i}): {mean_squared_error(y_train, y_pred)}")
```

```
# ## RANDOM FOREST
## a parameter grid for hyperparemeters in RF
param_grid = {
    'min_samples_leaf': [0.005, 0.01, 0.02],
    'min_samples_split': [0.01, 0.02, 0.03],
    "max_features": ["sqrt", 11, 18]
}

def fit_rf(x_train, y_train):
    rfr = RandomForestRegressor(random_state = 1)
    CV_rf = GridSearchCV(estimator = rfr, param_grid = param_grid,
                         scoring = "neg_mean_squared_error",
                         cv = 5, verbose = 2)
    CV_rf.fit(x_train, y_train)
    best_grid = CV_rf.best_estimator_
    return best_grid

clusters = list(range(5))
for t in tqdm(clusters):
    data = pd.read_csv(f"TrainingData/StockType{t}.csv")
    x_train, y_train = prepare(data)
    rf_model = fit_rf(x_train, y_train)
    with open(f'model_rf_cluster{t}', 'wb') as files:
        pickle.dump(rf_model, files)

# ### Best RF model Performance
for i in range(5):
    data = pd.read_csv(f"TrainingData/StockType{i}.csv")
    x_train, y_train = prepare(data)
    loaded_model = pickle.load(open(f'model_rf_cluster{i}', 'rb'))
    y_pred = loaded_model.predict(x_train)
    print(f"Mean squared error (Cluster {i}): {mean_squared_error(y_train, y_pred)}")

# ## XGBOOST
# A parameter grid for XGBoost
params = {
       'gamma': [0.005, 0.01, 0.1],
       'learning_rate': [0.005, 0.01, 0.1],
       'max_depth': [6, 10, 20]
       }

def fit_xgb(x_train, y_train):
    xgb_mod = xgb.XGBRegressor(random_state = 1)
    CV_xgb = GridSearchCV(estimator = xgb_mod, param_grid = params,
                         scoring = "neg_mean_squared_error",
                         cv = 5, verbose = 2)
    CV_xgb.fit(x_train, y_train)
    best_xgb = CV_xgb.best_estimator_
    return best_xgb

clusters = list(range(5))
```

```
for t in tqdm(clusters):
    data = pd.read_csv(f"TrainingData/StockType{t}.csv")
    x_train, y_train = prepare(data)
    xgb_model = fit_xgb(x_train, y_train)
    with open(f'model_xgb_cluster{t}', 'wb') as files:
        pickle.dump(xgb_model, files)

# ### Best XGBoost model Performance
for i in range(5):
    data = pd.read_csv(f"TrainingData/StockType{i}.csv")
    x_train, y_train = prepare(data)
    loaded_model = pickle.load(open(f'model_xgb_cluster{i}', 'rb'))
    y_pred = loaded_model.predict(x_train)
    print(f"Mean squared error (Cluster {i}): {mean_squared_error(y_train, y_pred)}")

### Neural Network
def mse(net, features, labels):
    with torch.no_grad():
        mse = criterion(net(features), labels)
    return mse.item()

def train(net, train_features, train_labels, test_features, test_labels,
          num_epochs, learning_rate, weight_decay, batch_size):
    train_ls, test_ls = [], []
    dataset = torch.utils.data.TensorDataset(train_features, train_labels)
    train_iter = torch.utils.data.DataLoader(dataset, batch_size, shuffle=True)
    optimizer = torch.optim.Adam(params=net.parameters(), lr=learning_rate,
         weight_decay=weight_decay)
    net = net.float()
    for epoch in range(num_epochs):
        for X, y in train_iter:
            l = criterion(net(X.float()), y.float())
            optimizer.zero_grad()
            l.backward()
            optimizer.step()
        train_ls.append(mse(net, train_features, train_labels))
        if test_labels is not None:
            test_ls.append(mse(net, test_features, test_labels))
    return train_ls, test_ls

def get_k_fold_data(k, i, X, y):
    assert k > 1
    fold_size = X.shape[0] // k
    X_train, y_train = None, None
    for j in range(k):
        idx = slice(j * fold_size, (j + 1) * fold_size)
        X_part, y_part = X[idx, :], y[idx]
        if j == i:
            X_valid, y_valid = X_part, y_part
        elif X_train is None:
            X_train, y_train = X_part, y_part
```

```
        else:
            X_train = torch.cat((X_train, X_part), dim=0)
            y_train = torch.cat((y_train, y_part), dim=0)
    return X_train, y_train, X_valid, y_valid

def k_fold(k, X_train, y_train, num_epochs,
           learning_rate, weight_decay, batch_size):
    train_l_sum, valid_l_sum = 0, 0
    for i in range(k):
        data = get_k_fold_data(k, i, X_train, y_train)
        net = model
        train_ls, valid_ls = train(net, *data, num_epochs, learning_rate,
                                   weight_decay, batch_size)
        train_l_sum += train_ls[-1]
        valid_l_sum += valid_ls[-1]
        print('fold %d, train mse %f, valid mse %f' % (i, train_ls[-1], valid_ls[-1]))
    return train_l_sum / k, valid_l_sum / k

for i in tqdm(range(5)):
    data0 = pd.read_csv(f"TrainingData/StockType{i}.csv")
    features = data0.drop(["Target"], axis=1)
    fl = features.columns.to_list()
    # standardize the data
    target = data0[['Target']]
    features = features.apply(lambda x: (x - x.mean()) / (x.std()))
    scaler = MinMaxScaler(feature_range=(-1, 1))
    target["Target"] = scaler.fit_transform(data0['Target'].values.reshape(-1,1))
    # prepare the data
    train_features = torch.tensor(features.values, dtype=torch.float)
    train_labels = torch.tensor(target.Target.values, dtype=torch.float)
    model_sequential = nn.Sequential(
         nn.Linear(train_features.shape[1], 128),
         nn.ReLU(),
         nn.Linear(128, 256),
         nn.ReLU(),
         nn.Linear(256, 256),
         nn.ReLU(),
         nn.Linear(256, 256),
         nn.ReLU(),
         nn.Linear(256, 1)
        )
    class Net(nn.Module):
        def __init__(self, features):
            super(Net, self).__init__()

            self.linear_relu1 = nn.Linear(features, 128)
            self.linear_relu2 = nn.Linear(128, 256)
            self.linear_relu3 = nn.Linear(256, 256)
            self.linear_relu4 = nn.Linear(256, 256)
            self.linear5 = nn.Linear(256, 1)
```

```
        def forward(self, x):

            y_pred = self.linear_relu1(x)
            y_pred = nn.functional.relu(y_pred)

            y_pred = self.linear_relu2(y_pred)
            y_pred = nn.functional.relu(y_pred)

            y_pred = self.linear_relu3(y_pred)
            y_pred = nn.functional.relu(y_pred)

            y_pred = self.linear_relu4(y_pred)
            y_pred = nn.functional.relu(y_pred)

            y_pred = self.linear5(y_pred)
            return y_pred
    model = Net(features=train_features.shape[1])

    criterion = nn.MSELoss(reduction='mean')
    k, num_epochs, lr, weight_decay, batch_size = 5, 50, 0.0005, 0, 64

    train_l, valid_l = k_fold(k, train_features, train_labels, num_epochs, lr, weight_decay,
        batch_size)
    print(f"For cluster {i}")
    print('%d-fold validation: avg train mse %f, avg valid mse %f' % (k, train_l, valid_l))
```

```
# # Build Factor
# In this section, we will build the factor for each stocks using XGBoost/Random Forest/Lasso
    Regression models.
### Note that for the sake of brevity, we only put the code for one model. For other models, we
    use the same codes and just simply replace the "loaded_model" with the model we want.

# ## Standardize the Factor on Market value and Industry types using regression.
filenames=os.listdir(r'data\individual_stocks')
stock_list = []
Ind_list = []
for file in tqdm(filenames):
    if file != "20181217-20190616.xlsx":
        data = pd.read_excel(f"data//individual_stocks//{file}" , sheet_name="Sheet1")
        stock_list += data["    "].tolist()
        Ind_list += data["              "].tolist()

industry_information = pd.DataFrame(columns=["code", "Industry"])
industry_information["code"] = stock_list
industry_information["Industry"] = Ind_list
industry_information.drop_duplicates(inplace=True)

# ### ADD Industry infromation into featured data.
from statsmodels.formula.api import ols
for clusters in tqdm(range(5)):
```

```
    cluster_stocks = loadtxt(f'TrainingData/stocktype{clusters}.txt')
    R2 = []
    AR2 = []
    stock_datac = pd.DataFrame()
    for stock in cluster_stocks.tolist():
        stock = int(stock)
        stock_data = pd.read_csv(f"featured_stock/{stock}.csv")
        index = industry_information.code[industry_information.code ==
            str(stock)+".SH"].index.tolist()
        stock_data["Industry"] = [industry_information["Industry"].loc[index[0]]]*len(stock_data)
        x = stock_data.drop(["time", "code", "Target", "Industry"], axis=1)
        # calculate unstandardized factor
        loaded_model = pickle.load(open(f'model_xgb_cluster{clusters}', 'rb'))
        y_pred = loaded_model.predict(x)
        stock_data["factor_ori"] = y_pred
        # aggregate data into cluster
        stock_datac = pd.concat([stock_datac, stock_data[:1497]], axis=0)
        stock_data.to_csv(f"data_with_factor/{stock}.csv", index=False)
        # train the regression model
    fit = ols('factor_ori ~ C(Industry) + market_value', data=stock_datac).fit()
    R2.append(fit.rsquared)
    AR2.append(fit.rsquared_adj)
    print(f"Average R^2 for cluster {clusters} is {np.mean(R2)}")
    print(f"Average Adjusted R^2 for cluster {clusters} is {np.mean(AR2)}")

    for stock in cluster_stocks.tolist():
        stock = int(stock)
        data = pd.read_csv(f"data_with_factor/{int(stock)}.csv")
        data["FACTOR"] = data["factor_ori"] - fit.predict(data)
        data.to_csv(f"data_with_factor/{stock}.csv", index=False)


# ## Factor Analysis
#
# ### calculate IC
files=os.listdir(r'data_with_factor')
drop_files = []
for f in files:
    data = pd.read_csv(f'data_with_factor/{f}')
    if len(data) < 1749:
        drop_files.append(f)
len(drop_files)

IC_list = []
for i in tqdm(range(1497)):
    # here all data has length 1749, and test data start from 1497
    factor_list = []
    return_list = []
    for f in files:
        if f not in drop_files:
            data_c = pd.read_csv(f'data_with_factor/{f}')
```

```
            factor_list.append(data_c["FACTOR"].iloc[i])
            return_list.append(data_c["Target"].iloc[i])
    IC_i = np.corrcoef(factor_list, return_list)
    IC_list.append(IC_i[0][1])

IR = stat.mean(IC_list)/stat.stdev(IC_list)
print(f"Our factor's IR is {IR}")
print(f"Our factor's mean IC is {stat.mean(IC_list)}")

sns.histplot(IC_list, bins=20)
plt.xlabel("IC Value")
plt.axvline(x=stat.mean(IC_list),color='red')
plt.savefig("ICdistributionxg")
count = 0
for ic in IC_list:
    if ic > 0:
        count += 1
print(f"positive proporation of IC is {count/len(IC_list)}")

sns.histplot(IC_list[-90:], bins=20)
plt.xlabel("IC Value")
plt.axvline(x=stat.mean(IC_list[-90:]),color='red')
plt.savefig("ICdistributionxg")
count = 0
for ic in IC_list[-90:]:
    if ic > 0:
        count += 1
print(f"positive proporation of IC is {count/len(IC_list[-90:])}")

# ## BACKTESTING
# Our backtesting period is 2021-03-01 to 2022-03-02. Index is from 1497 to 1742.
# In this period, we experienced 3 time index components adjustments
# Our portfolio starts with RMB 1,000,000. Transection costs are ignored.

Index = pd.read_excel("data/index.xlsx")
Index.columns = ["Time", "prior_close", "open", "high", "low", "close", "volume",
                 "amount", "change", "change_percent"]
plt.figure(figsize=(15,5))
plt.plot(Index["close"].iloc[1497:1742])
plt.show()

# period 2021-03-01 ~ 2021-06-14, index 1497:1568
stock_list1 =
    pd.read_excel("data//individual_stocks//20201214-20210614.xlsx",sheet_name="Sheet1")[""].tolist()
# period 2021-06-16 ~ 2021-12-12, index 1568:1690
stock_list2 =
    pd.read_excel("data//individual_stocks//20210616-20211212.xlsx",sheet_name="Sheet1")[""].tolist()
# period 2021-12-13 ~ 2022-03-02, index 1690:1742
stock_list3 =
    pd.read_excel("data//individual_stocks//20211213-.xlsx",sheet_name="Sheet1")[""].tolist()
```

```
def order_dict(dicts, n):
    result = []
    result1 = []
    p = sorted([(k, v) for k, v in dicts.items()], reverse=True)
    s = set()
    for i in p:
        s.add(i[1])
    for i in sorted(s, reverse=True)[:n]:
        for j in p:
            if j[1] == i:
                result.append(j)
    for r in result:
        result1.append(r[0])
    return result1

def backtesting(stock_list, start, end, startmoney, n):
    # get adjustment dates, 5 days per adjust
    money_list = []
    dates = []
    for i in range((end-start)//5):
        dates.append(start+5*i)
    # get selected stocks at adjustment dates
    for date in tqdm(dates):
        if date != dates[-1]:
            factor_dict = {}
            for stock in stock_list:
                data = pd.read_csv(f"data_with_factor/{stock[:6]}.csv")
                if len(data) == 1749:
                    factor_dict[stock] = data["FACTOR"].iloc[date]

            stock_selected = order_dict(factor_dict, n)#
            print(stock_selected)#
            # get return of selected stocks
            return_list=[]
            for ss in stock_selected:
                r_l = []
                ssdata = pd.read_csv(f"data_with_factor/{ss[:6]}.csv")
                for k in range(5):
                    r_l.append(ssdata["change_rate"].iloc[date+k])
                return_list.append(r_l)
            # calculate the total return
            total_return = []
            for j in range(5):
                return_ = 0
                for h in range(n):#
                    return_ += return_list[h][j]/100
                total_return.append(return_)
            for ret in total_return:
                startmoney = startmoney/n * (n+ret)#
                money_list.append(startmoney)
        # calculaate the finishing dates
```

```
        elif date == dates[-1]:
            factor_dict = {}
            for stock in stock_list:
                data = pd.read_csv(f"data_with_factor/{stock[:6]}.csv")
                if len(data) == 1749:
                    factor_dict[stock] = data["FACTOR"].iloc[date]
            stock_selected = order_dict(factor_dict, n)#
            # get return of selected stocks
            return_list=[]
            for ss in stock_selected:
                r_l = []
                ssdata = pd.read_csv(f"data_with_factor/{ss[:6]}.csv")
                for k in range(5+(end-start)%5):
                    r_l.append(ssdata["change_rate"].iloc[date+k])
                return_list.append(r_l)
            # calculate the total return
            total_return = []
            for j in range(5+(end-start)%5):
                return_ = 0
                for h in range(n):#
                    return_ += return_list[h][j]/100
                total_return.append(return_)
            for ret in total_return:
                startmoney = startmoney/n * (n+ret)#
                money_list.append(startmoney)
    return money_list

value1 = backtesting(stock_list1, 1497, 1568, 1000000, 5)
value2 = backtesting(stock_list2, 1568, 1690, value1[-1], 5)
value3 = backtesting(stock_list3, 1690, 1742, value2[-1], 5)
value1.extend(value2)
value1.extend(value3)
value = [1000000]
value.extend(value1)
coe = 1000000/Index["open"].iloc[1497]
Index["close"] = Index["close"]*coe
Index_list = Index["close"].iloc[1497:1742].values.tolist()
Index_value = [1000000]
Index_value.extend(Index_list)
plt.figure(figsize=(15,3))
plt.plot(Index_value, label="Index")
plt.plot(value, label="Portfolio")
plt.legend()
plt.title("Index Value v.s. Factor Based Portfolio (XGBoost)")
plt.savefig("backtestresultxg")
print(f"The yearly return of SSE 50 Index is {(Index_value[-1]-Index_value[0])/Index_value[0]}")
print(f"The yearly return of our factor based portfolio is {(value[-1]-value[0])/value[0]}")
print(f"The excess return of our factor based portfolio is {(value[-1]-value[0])/value[0] -
    (Index_value[-1]-Index_value[0])/Index_value[0]}")
```