Multi-stock portfolio analysis and forecasting

Dong Huanchen	Tian Yuxin	Kong Deran	Zhao Yipan
1930026024	1930013031	1930026061	1930026171
p930026024@mail.uic.edu.cn	p930013031@mail.uic.edu.cn	p930026061@mail.uic.edu.cn	p930026171@mail.uic.edu.cn

21 May 2022

Abstract

At present, the financial market is always unpredictable, full of uncertainties, and it is a market with many investment risks. Therefore, investors always hope to find a way to reduce investment risks and increase profits. So, our project aims to produce the maximum return by portfolio analysis and time series analysis.
We use Monte Carlo simulation to fit a Markowitz model such that get the maximum profit of the different portfolio then compare the results by Sharpe ratio. And also train a model which is construct by Neural network in deep learning. Then we do time series analysis based on the weights we get from the Markowitz and Neural network model to have predictions. At last, we generalize our model on tushare.

Key-words: Portfolio analysis, Neural network, Deep learning, Time series analysis

1 Introduction

Among the college students, the investment we are concerned about mainly focus on stocks and funds, then we choose stock portfolios for analysis. The core problem of investment optimization is how to reasonably allocate existing wealth among investable risk assets in order to achieve investment goals such as maximizing profits under given risks or maximizing cumulative returns.
First, we prepare multiple stock data, including the stock's opening price, closing price, and date. But the original data has total 14 stocks which is complexity and unrealizable to analysis, so we choose 3 stocks by calculating the average annualized return and dispersion coefficient. In this way, we calculate the returns of different investment portfolios according to different weight combinations, such as equal weight and random weight.
Then, descriptive statistical analysis is carried out, and the Markowitz model is simulated by Monte Carlo through data such as mean and variance. In addition, we can also explore portfolios under different conditions, such as portfolios with minimal

investment risk. We also need to use the Sharpe ratio to calculate yield and rate of profit.

And the deep learning we also used to train a model to calculate the weights of portfolio such that we can compared all model we simulated to conclude a model with best fitness.

2 Data Preprocessing

We collect stock data on Kaggle first. In our original data, we have a total of 14 stocks. We select 3 of these 14 stocks based on their annualized returns and the dispersion coefficients to perform the calculations.

Where the annual return is defined as the ratio of the total return earned from investing in the stock to the original investment amount.

Calculation formula:

Daily rate of return:

$$R_t = \frac{P_t - P_{t-1}}{P_{t-1}} \quad (1)$$

(P_t denotes the closing price of the stock at moment t and P_{t-1} denotes the closing price of the stock at moment $t-1$)

Annual return: (assuming 252 trading days in a year)

$$(1+\mu)^{252} - 1 \quad (2)$$

The coefficient of dispersion of a stock is the coefficient of variation, which is the ratio of the standard deviation to the mean of a set of data. For the standard deviation and variance of different data samples, the results are naturally not directly comparable due to the different units of measurement of the data, so the calculation of the dispersion coefficient is performed in order to produce an identical measure.

Calculation formula:

$$\frac{Standard\ deviation\ of\ daily\ return}{Average\ of\ daily\ yields} \quad (3)$$

	Average annualized return	Dispersion coefficient
NYA	0.08674391	23.8301031
000001.SS	0.07053724	50.7401712
399001.SZ	0.05933182	71.5666889
GDAXI	0.11880133	24.3875737
GSPTSE	0.04896199	35.5572638
HSI	0.06786267	40.7232776
IXIC	0.17900581	14.7815725
J203.JO	0.07797617	30.3939939
KS11	0.03221267	62.8841420
N100	0.09489260	27.2921811
N225	0.16322731	21.2701295
NSEI	0.14059923	17.4020269
SSMI	0.08386610	27.4312866
TWII	0.08012641	26.8225086

Table 1 Average annualized return and dispersion coefficient

Therefore, we believe that the annualized return is more influential, so we have selected three stocks with higher annualized returns, namely IXIC, N225, NSEI.

3 Portfolio Analysis

3.1 Correlation analysis of portfolio

Correlation matrix: the linear relationship between stocks.

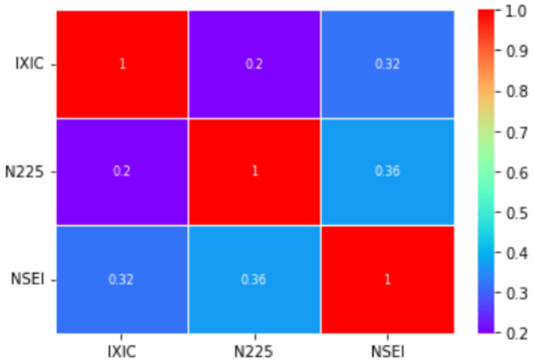

Figure 1 Correlation matrix

Covariance matrix: the fluctuation between stocks.

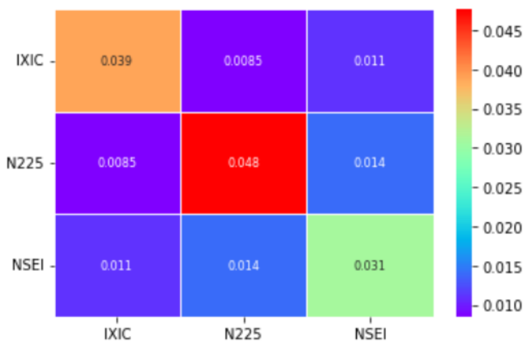

Figure 2 Covariance matrix

Standard deviation: portfolio risk

$$\sigma = \sqrt{w^T \cdot \Sigma \cdot w} \tag{4}$$

where w is the weight of portfolio, Σ is the covariance matrix.

3.2 Random weighted portfolio

Set a random set of weights, such as:

$$w = [0.1,0.75,0.15]$$

3.3 Monte Carlo simulation

We use Monte Carlo simulation [7] for analysis, that is, randomly generate a set of weights, calculate the return and standard deviation under the combination, repeat this process many 10000 times, and calculate the return and standard deviation of each combination.

According to Markowitz portfolio theory, rational investors always maximize the expected return at a given level of risk, or minimize the expected risk at a given level of return. All points that satisfy this condition form the Markowitz [8] efficient frontier. And only the points on the efficient frontier are the most efficient portfolios.

In this case, we choose the portfolio with the least risk, and have the highest return in that risk which is minimum risk portfolio.

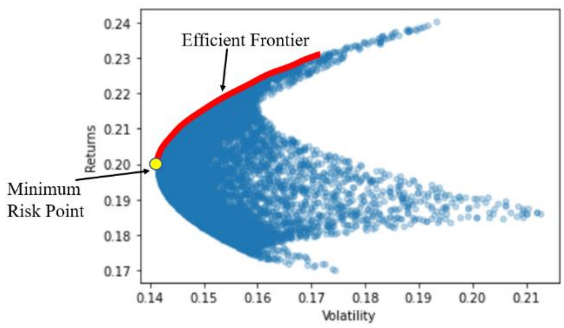

Figure 3 Monte Carlo simulation

And the weight in this case is:

$$w = [0.34599492,0.23494931,0.41905577]$$

3.4 Sharpe ratio

The Sharpe ratio [9] measures the performance of an investment such as a security or portfolio compared to a risk-free asset, after adjusting for its risk. It represents the additional amount of return that an investor receives per unit of increase in risk.

$$R_S = \frac{E(R_i - r_f)}{\sqrt{var(R_i - r_f)}} = \frac{E(R_p)}{\sqrt{var(R_p)}} \quad (5)$$

where $E(R_p)$ and $var(R_p)$ are the estimates of the mean and variance of portfolio returns.

And R_p is the realized portfolio return over n stocks denoted as:

$$R_p = \sum_{i=1}^{n} w_i r_i \quad (6)$$

where r_i is the return of stock i. And the weight of each stock i is $w_i, w_i \in [0,1]$ and $\sum_{i=1}^{n} w_i = 1$

We calculate Sharpe ratio of the portfolios from Monte Carlo simulation, and return the weight of the maximum Sharpe ratio.

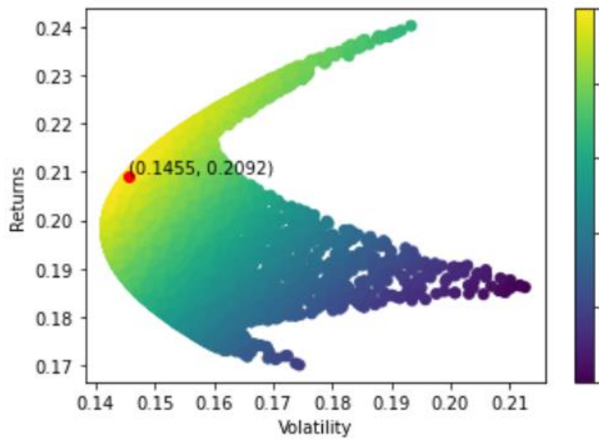

Figure 4 Sharpe ratio

The weight of maximum Sharpe ratio:

$w = [0.50319022, 0.22285832, 0.27395146]$

3.5 Neural network in deep learning

The network architecture is depicted in Figure 5. Our model consists of three main building blocks: input layer, neural layer, and output layer. The idea of this design is to use neural networks to extract cross-sectional features from the input assets. Once the features are extracted, the model outputs portfolio weights and we get realized returns to maximize Sharpe ratios.

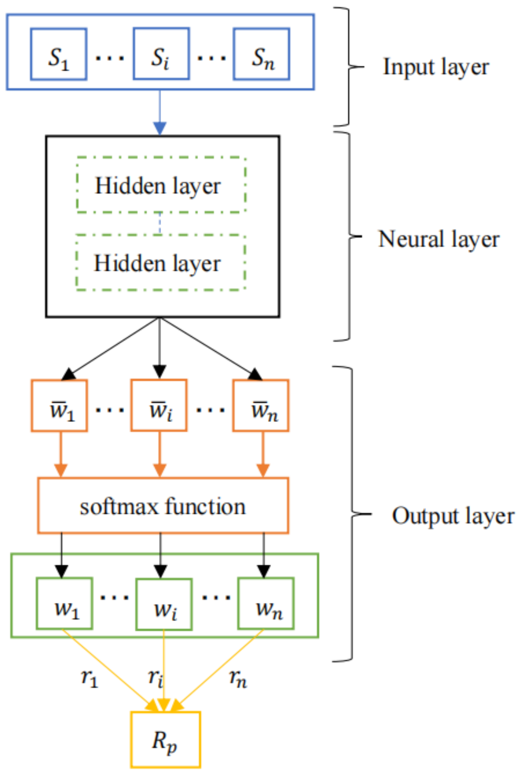

Figure 5 Deep learning process

Input layer: We denote each stock as S_i, and our portfolio consists of n stocks. Each input is the price of the stock over a period of time, so the dimension of input will be (k, n). We can then feed this input into the network layer and expect to extract nonlinear features.

Network layer: The network layer consists of multiple hidden layers. A number of works [3,4,5] have shown that LSTM has the best performance in daily financial data modeling. The cell structure of LSTM has a gate mechanism that can summarize and filter information from its long history, so the model ends up with fewer trainable parameters and better generalization results.

As a result, we used the LSTM in our model.

Output layer: Because the weight is non-negative and the sum is one, we used softmax [6] as the activation equation to normalize the output of neural layer $(\overline{w}_1, \overline{w}_2, \ldots, \overline{w}_n)$, We get the final portfolio weights $(w_1, w_2, \ldots, w_n)$ and multiply them times the return rate of each stock $(r_1, r_2, \ldots, r_n)$ to obtain the return rate of the portfolio R_p. By formula (7), we can obtain the Sharpe ratio, take the negative Sharpe ratio as the loss function, and use 'Adam' as the optimizer to continuously update the parameters through gradient rise:

$$\theta_{new} \coloneqq \theta_{old} + \alpha \frac{\partial R_S}{\partial \theta} \quad (7)$$

where α is the learning rate and the process can be repeated for many epochs until the convergence of Sharpe ratio.

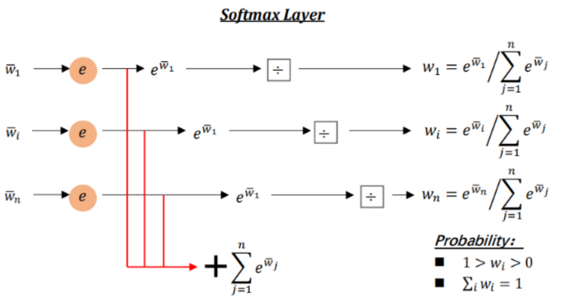

Figure 6 Softmax layer

Result of deep learning:

$w = [0.4945516, 0.33219865, 0.17324968]$

3.6 Results of different portfolio

$w = [0.1, 0.75, 0.15]$

$w\_MR = [0.34599492, 0.23494931, 0.41905577]$

$w\_MSR = [0.50319022, 0.22285832, 0.27395146]$

$w\_DL = [0.4945516, 0.33219865, 0.17324968]$

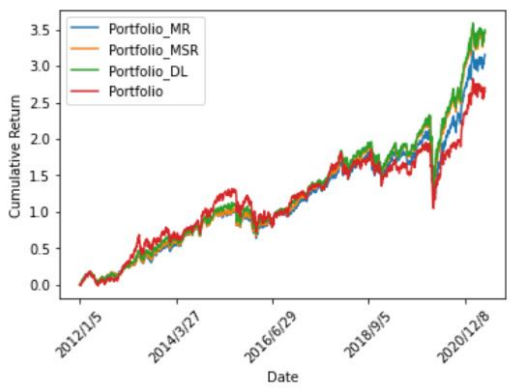

Figure 7 Cumulative return of portfolio

4 Time Series Analysis

In the previous work, we obtained the weight corresponding to the maximum return calculated by Monte Carlo and the weight obtained by the neural network of machine learning, and can obtain the cumulative return under each weight. Based on these two weights, time series analysis of corresponding income was carried out.

Here we use the ARIMA(p,d,q) function for time series forecasting.

4.1 Time Series on Markowitz Weights

Since the original cumulative return data fluctuates greatly, we perform first-order difference on the data for smooth processing. And get its corresponding ACF (Auto correlation function) graph and PACF (Partial auto correlation function) graph.

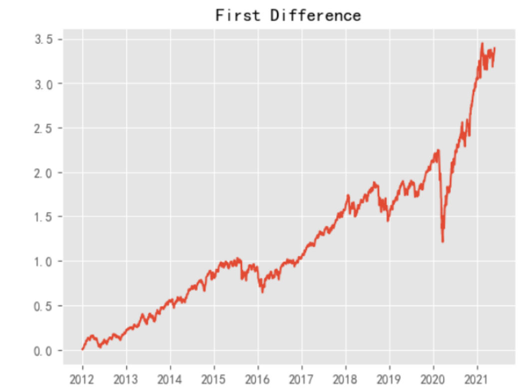

Figure 8 First difference of Monte Carlo

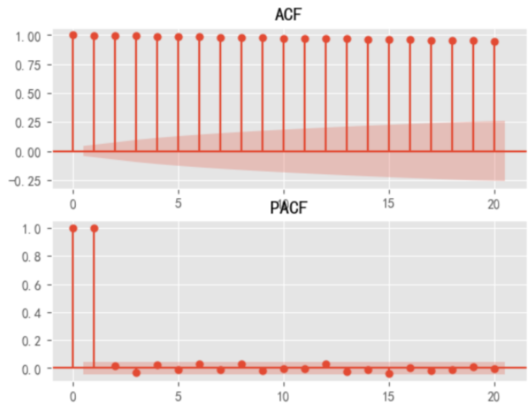

Figure 9 ACF&PACF of first difference of MC

However, we can see that the first-order difference map still has large fluctuation, but the ACF is tailed, and all points do not fall within the range of double standard deviation, so we do the second-order difference. The second order difference map and its ACF and PACF are as follows.

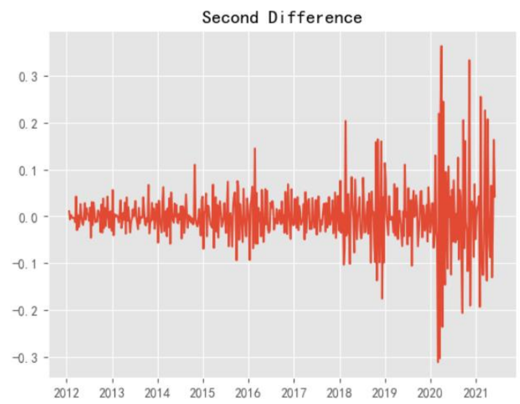

Figure 10 Second difference of Monte Carlo

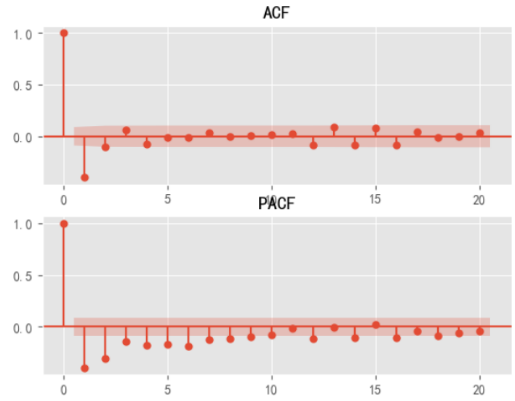

Figure 11 ACF&PACF of second difference of MC

The second-order difference maps are almost stationary, and both ACF and PACF are truncated that tend to 0. And because p=9, q=2 can be obtained from ACF and PACF, and the second-order difference corresponds to d=2, we use ARIMA (9, 2, 2) to model the earnings after September 2020, which can be get the graph below, where the blue line is the original cumulative return, and the red line is the forecast data. [2]

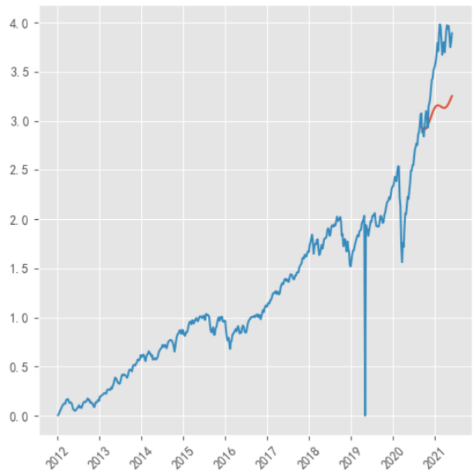

Figure 12 Time series prediction of MC

4.2 Time Series on Deep Learning Weights

Same as the calculation method in 3.1, first we can get the first order difference and its ACF and PACF.

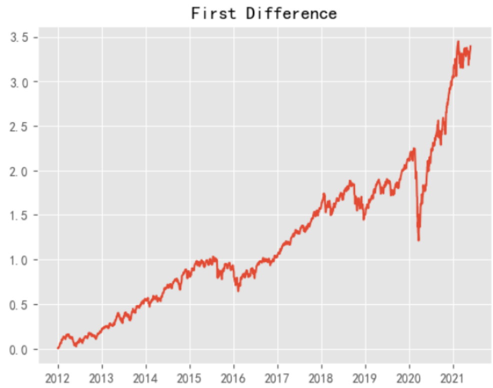

Figure 13 First difference of deep learning

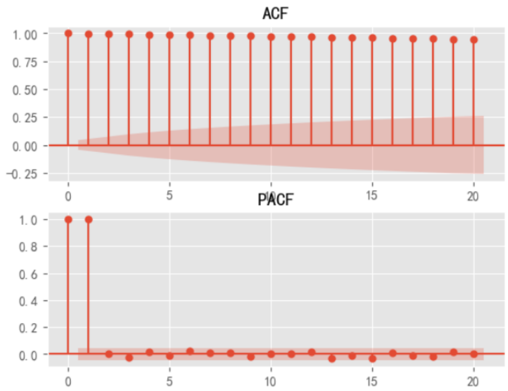

Figure 14 ACF&PACF of first difference of deep learning

Still not getting the desired stationarity, then calculating the second difference.

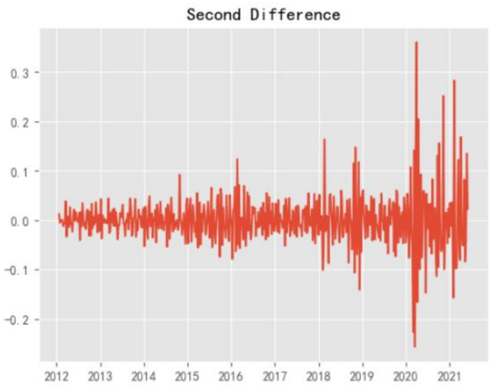

Figure 15 Second difference of deep learning

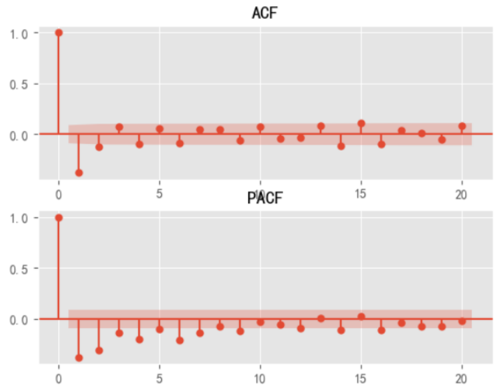

Figure 16 ACF&PACF of second difference of deep learning

Similarly, we get ARIMA (9, 2, 2) to model the earnings after September 2020, and we can get the following figure. Comparing the predictions made by the weights calculated by Monte Carlo, we can get the accuracy of the predictions made by machine learning is higher, and the model trained by machine learning has a better fitness.

Figure 17 Time series prediction of deep learning

5 Generalize the Model

In order to make our model more practical, we connected to the Tushare platform, read all the stocks currently trading normally, and obtained the weight of the stocks we wanted to buy through the model. There are two ways to select stocks.

The first way is to input the number of stocks(n) you want to invest in the portfolio, and the model will randomly select n stocks from all stocks.

The second way is to input the stock codes you want to invest in the portfolio one by one. Then the model will simulate the weight of the list of stocks, get the optimal portfolio weight, and draw the cumulative return chart of the portfolio.

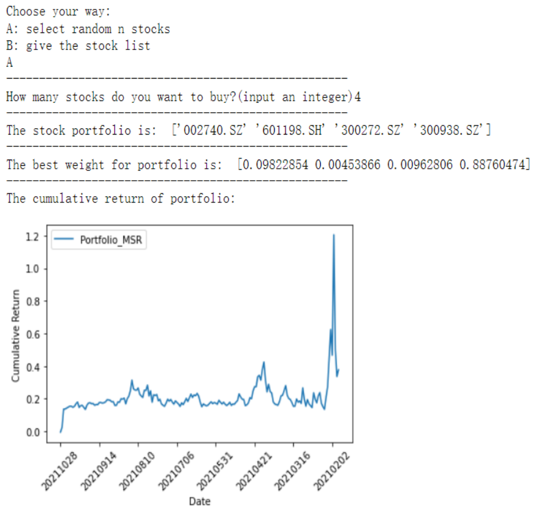

Figure 18 Demo of choice A

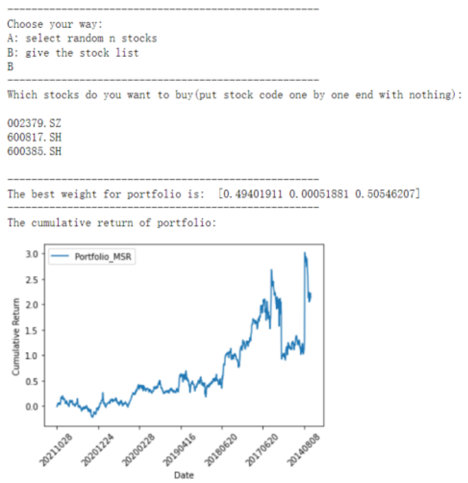

Figure 19 Demo of choice B

6 Conclusion

Our aim is to maximize returns by finding an optimal portfolio of three stocks, for which we use 4 different methods: Random weights, Monte Carlo simulation, Sharpe ratio, and Deep learning. Through these four methods, the maximum value of cumulative income and the corresponding weight are calculated respectively. And through the comparison chart of the cumulative returns of these four methods, we can clearly see that the weights obtained by deep learning have the greatest returns, so we can conclude that deep learning has the best fit for the optimal portfolio of stocks.

In addition, based on the better two of these four methods which are Monte Carlo and Deep learning, we conducted time series analysis. By taking the part of the return data as the training set, we explored the corresponding ARIMA models under different weights and carried out fitting, and finally predicting. And then we compare the predicted income and the actual income, we can conclude that the accuracy of Deep learning is higher than that of Monte Carlo, and it can be proved once again that the fitting effect of deep learning is better than other methods.

Finally, we generalize our project so that it can connect to the tushare website and get stock information and enter the information into our system. Then let the system train itself and find the optimal return and weight of the corresponding stock portfolio.

Reference

[1] Zhang, Z., Zohren, S., & Roberts, S. (2020). Deep learning for portfolio optimization. *The Journal of Financial Data Science*, *2*(4), 8-20.

[2] Brownlee, J. (2020, December 9). 11 classical time series forecasting methods in python (cheat sheet). Machine Learning Mastery. Retrieved May 21, 2022, from https://machinelearningmastery.com/time-series-forecasting-methods-in-python-cheat-sheet/

[3] Lim, B., Zohren, S., & Roberts, S. (2019). Enhancing time-series momentum strategies using deep neural networks. *The Journal of Financial Data Science*, *1*(4), 19-38.

[4] Tsantekidis, A., Passalis, N., Tefas, A., Kanniainen, J., Gabbouj, M., & Iosifidis, A. (2017, August). Using deep learning to detect price change indications in financial markets. In *2017 25th European Signal Processing Conference (EUSIPCO)* (pp. 2511-2515). IEEE.

[5] Zhang, Z., Zohren, S., & Roberts, S. (2020). Deep reinforcement learning for trading. *The Journal of Financial Data Science*, *2*(2), 25-40.

[6] Liang, X., Wang, X., Lei, Z., Liao, S., & Li, S. Z. (2017, November). Soft-margin softmax for deep classification. In *International Conference on Neural Information Processing* (pp. 413-421). Springer, Cham.

[7] Rubinstein, R. Y., & Kroese, D. P. (2016). *Simulation and the Monte Carlo method*. John Wiley & Sons.

[8] Brodie, J., Daubechies, I., De Mol, C., Giannone, D., & Loris, I. (2009). Sparse and stable Markowitz portfolios. *Proceedings of the National Academy of Sciences*, *106*(30), 12267-12272.

[9] Sharpe, W. F. (1998). The sharpe ratio. *Streetwise–the Best of the Journal of Portfolio Management*, 169-185.

[10] [3] Saksham Mittal and Sujoy Bhattacharya and Satrajit Mandal. (2021). Characteristics analysis of behavioural portfolio theory in the Markowitz portfolio theory framework. *Managerial Finance*, 48(2), pp. 277-288.

Github link

https://github.com/DominicDHC/Financial-Computing

Appendix

```
import pandas as pd
import numpy as np
```

```
import pandas_datareader
import datetime
from pandas import read_csv
import matplotlib.pylab as plt
import seaborn as sns
from matplotlib.pylab import style
from statsmodels.tsa.arima_model import ARIMA,ARMA
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf
import statsmodels.api as sm

## Part2: Data preprocessing
stock_data = pd.read_csv("stock.csv",index_col='Date')
stock_data = stock_data.dropna()
stock_IXIC = stock_IXIC['2012/1/3':]
stock_IXIC=stock_IXIC.dropna()
stock_IXIC['earn_rate']=np.log((stock_IXIC['Close']/stock_IXIC['Close'].shift(1)))
earn_rate_data=stock_IXIC['earn_rate'].dropna()
print(earn_rate_data.head())
earn_rate_data.plot(grid=True,color='green')
plt.title('2012-2019 earn rate of every day')
plt.ylabel('earn rate', fontsize='10')
plt.xlabel('date', fontsize='10')
plt.legend(loc='best')
plt.show()
earn_mean_daily = np.mean(earn_rate_data)
print("Average daily return of IXIC stock:", earn_mean_daily)
earn_rate_year=(1+np.mean(earn_rate_data))**252-1
print("IXIC Stock Average Annualized Return:",earn_rate_year)
earn_rate_coefficient=np.std(earn_rate_data)/np.mean(earn_rate_data)
print("IXIC Stock Daily Yield Dispersion Coefficient:",earn_rate_coefficient)

## Part3: Portfolio analysis
df = pd.read_csv("stock.csv")
ticker_list = df['Index'].unique()
df.index = df['Date']
df1 = df[df['Index']==ticker_list[0]]
df2 = df[df['Index']==ticker_list[1]]
df3 = df[df['Index']==ticker_list[2]]
df1 = df1[["Close"]]
```

```
df2 = df2[["Close"]]
df3 = df3[["Close"]]
df1.rename(columns={'Close':ticker_list[0]},inplace=True)
df2.rename(columns={'Close':ticker_list[1]},inplace=True)
df3.rename(columns={'Close':ticker_list[2]},inplace=True)
StockPrices = pd.concat([df1,df2,df3],axis='columns',names=['Date']).dropna()
StockReturns = StockPrices.pct_change().dropna()
print(StockReturns.head())
stock_return = StockReturns.copy()
# Calculate the daily rate of return and discard the missing value
StockReturns = StockPrices.pct_change().dropna()
print(StockReturns.head())
# Copy the yield data to the new variable stock_ Return, this is for the convenience of subsequent calls
stock_return = StockReturns.copy()

portfolio_weights = np.array([0.1,0.75,0.15])
#Calculate Weighted Stock Returns
WeightedReturns = stock_return.mul(portfolio_weights, axis=1)
# Calculate the return of the portfolio
StockReturns['Portfolio'] = WeightedReturns.sum(axis=1)

print(StockReturns.head())

StockReturns.Portfolio.plot()
plt.show()
# Define the drawing function of cumulative income curve
def cumulative_returns_plot(name_list):
    for name in name_list:
        CumulativeReturns = ((1+StockReturns[name]).cumprod()-1)
        CumulativeReturns.plot(label=name)
    plt.legend()
    plt.xticks(rotation=45)
    plt.xlabel("Date")
    plt.ylabel("Cumulative Return")
    plt.show()

cumulative_returns_plot(['Portfolio'])
numstocks = 3
portfolio_weights_ew = np.repeat(1/numstocks, numstocks)
```

```
StockReturns['Portfolio_EW'] = stock_return.mul(portfolio_weights_ew, axis=1).sum(axis=1)
print(StockReturns.head())
cumulative_returns_plot(['Portfolio', 'Portfolio_EW'])
# StockReturns = StockPrices.pct_change().dropna()

# stock_return = StockReturns.copy()
correlation_matrix = stock_return.corr()

print(correlation_matrix)
import seaborn as sns
#Create heat map
sns.heatmap(correlation_matrix,annot=True,cmap='rainbow',linewidths=1.0,annot_kws={'size':8})
plt.xticks(rotation=0)
plt.yticks(rotation=0)
plt.show()
cov_mat = stock_return.cov()
cov_mat_annual = cov_mat * 252
print(cov_mat_annual)
sns.heatmap(cov_mat_annual,annot=True,cmap='rainbow',linewidths=1.0,annot_kws={'size':8})
plt.xticks(rotation=0)
plt.yticks(rotation=0)
plt.show()
portfolio_weights = np.array([0.46,0.09,0.45])
portfolio_volatility = np.sqrt(np.dot(portfolio_weights.T, np.dot(cov_mat_annual, portfolio_weights)))
print(portfolio_volatility)
# Sets the number of simulations
number = 10000
# Set an empty numpy array to store the weight, yield and standard deviation obtained from each simulation
random_p = np.empty((number, 5))
# Set the seed of random number to make the result repeatable
np.random.seed(5)

for i in range(number):


    random5=np.random.random(3)
    random_weight=random5/np.sum(random5)

    #Calculate the average annual rate of return
    mean_return=stock_return.mul(random_weight,axis=1).sum(axis=1).mean()
```

```
        annual_return=(1+mean_return)**252-1

        #Calculate the annualized standard deviation, which is also called volatility
        random_volatility=np.sqrt(np.dot(random_weight.T,np.dot(cov_mat_annual,random_weight)))

        #Store the weight generated above, the calculated yield and standard deviation into the array random_
P medium
        random_p[i][:3]=random_weight
        random_p[i][3]=annual_return
        random_p[i][4]=random_volatility

RandomPortfolios=pd.DataFrame(random_p)

RandomPortfolios.columns=[ticker +'_weight' for ticker in ticker_list]+['Returns','Volatility']



RandomPortfolios.plot('Volatility','Returns',kind='scatter',alpha=0.3)
plt.show()
# Find the index value of the data with the smallest standard deviation
min_index = RandomPortfolios.Volatility.idxmin()

# Highlight the point with the lowest risk in the income risk scatter chart
RandomPortfolios.plot('Volatility', 'Returns', kind='scatter', alpha=0.3)
x = RandomPortfolios.loc[min_index,'Volatility']
y = RandomPortfolios.loc[min_index,'Returns']
plt.scatter(x, y, color='red')

plt.text(np.round(x,4),np.round(y,4),(np.round(x,4),np.round(y,4)),ha='left',va='bottom',fontsize=10)
plt.show()
# Extract the weight corresponding to the minimum fluctuation combination and convert it into numpy array
numstocks=3
MR_weights = np.array(RandomPortfolios.iloc[min_index, 0:numstocks])

StockReturns['Portfolio_MR'] = stock_return.mul(MR_weights, axis=1).sum(axis=1)

print(MR_weights)
# Set the risk-free return rate to 0
risk_free = 0
# Calculate the sharp ratio for each asset
```

```
RandomPortfolios['Sharpe'] = (RandomPortfolios.Returns - risk_free) / RandomPortfolios.Volatility

max_index = RandomPortfolios.Sharpe.idxmax()

RandomPortfolios.plot('Volatility', 'Returns', kind='scatter', alpha=0.3)
x = RandomPortfolios.loc[max_index,'Volatility']
y = RandomPortfolios.loc[max_index,'Returns']

# Plot the scatter plot of income standard deviation and color the sharp ratio
plt.scatter(RandomPortfolios.Volatility, RandomPortfolios.Returns, c=RandomPortfolios.Sharpe)
plt.scatter(x, y, color='r')
plt.colorbar(label='Sharpe Ratio')
plt.text(np.round(x,4),np.round(y,4),(np.round(x,4),np.round(y,4)),ha='left',va='bottom',fontsize=10)
plt.show()
MSR_weights = np.array(RandomPortfolios.iloc[max_index, 0:numstocks])

StockReturns['Portfolio_MSR'] = stock_return.mul(MSR_weights, axis=1).sum(axis=1)

print(MSR_weights)
import numpy as np

# setting the seed allows for reproducible results
np.random.seed(10)

import tensorflow as tf
from tensorflow.keras.layers import LSTM, Flatten, Dense
from tensorflow.keras.models import Sequential
import tensorflow.keras.backend as K

import pandas as pd

class Model:
    def __init__(self):
        self.data = None
        self.model = None

    def __build_model(self, input_shape, outputs):
        '''
        Builds and returns the Deep Neural Network that will compute the allocation ratios
```

```
        that optimize the Sharpe Ratio of the portfolio

        inputs: input_shape - tuple of the input shape, outputs - the number of assets
        returns: a Deep Neural Network model
        '''
        model = Sequential([
            LSTM(64, input_shape=input_shape),
            Flatten(),
            Dense(outputs, activation='softmax')
        ])

        def sharpe_loss(_, y_pred):
            # make all time-series start at 1
            data = tf.divide(self.data, self.data[0])

            # value of the portfolio after allocations applied
            portfolio_values = tf.reduce_sum(tf.multiply(data, y_pred), axis=1)

            portfolio_returns = (portfolio_values[1:] - portfolio_values[:-1]) / portfolio_values[:-1]    # %
change formula

            sharpe = K.mean(portfolio_returns) / K.std(portfolio_returns)

            # since we want to maximize Sharpe, while gradient descent minimizes the loss,
            #     we can negate Sharpe (the min of a negated function is its max)
            return -sharpe

        model.compile(loss=sharpe_loss, optimizer='adam')
        return model

    def get_allocations(self, data: pd.DataFrame):
        '''
        Computes and returns the allocation ratios that optimize the Sharpe over the given data

        input: data - DataFrame of historical closing prices of various assets

        return: the allocations ratios for each of the given assets
        '''
```

```
        # data with returns
        data_w_ret = np.concatenate([ data.values[1:], data.pct_change().values[1:] ], axis=1)

        data = data.iloc[1:]
        self.data = tf.cast(tf.constant(data), float)

        if self.model is None:
            self.model = self.__build_model(data_w_ret.shape, len(data.columns))

        fit_predict_data = data_w_ret[np.newaxis,:]
        self.model.fit(fit_predict_data, np.zeros((1, len(data.columns))), epochs=40, shuffle=False)
        return self.model.predict(fit_predict_data)[0]
model = Model()
w = model.get_allocations(pd.DataFrame(StockPrices))

print("The weight of portfolio is:",w)
StockReturns['Portfolio_DL'] = stock_return.mul(w, axis=1).sum(axis=1)
cumulative_returns_plot(['Portfolio_MR','Portfolio_MSR','Portfolio_DL','Portfolio'])


## Part 4: Time series
style.use('ggplot')
plt.rcParams['font.sans-serif'] = ['SimHei']
plt.rcParams['axes.unicode_minus'] = False
stockFile = './stock.csv'
stock = pd.read_csv(stockFile, index_col=0, parse_dates=[0]).dropna()
stock
df = pd.read_csv("stock.csv", parse_dates=[0])
sn = df['Index'].unique()
df.index = df['Date']
df1 = df[df['Index']==sn[0]]
df2 = df[df['Index']==sn[1]]
df3 = df[df['Index']==sn[2]]
df1 = df1[["Close"]]
df2 = df2[["Close"]]
df3 = df3[["Close"]]
df1.rename(columns={'Close':sn[0]},inplace=True)
df2.rename(columns={'Close':sn[1]},inplace=True)
df3.rename(columns={'Close':sn[2]},inplace=True)
```

```
stock_price = pd.concat([df1,df2,df3],axis='columns',names=['Date']).dropna()
stock_return = stock_price.pct_change().dropna()
w=[0.50319022, 0.22285832, 0.27395146]
Portfolio = stock_return.mul(w, axis=1).sum(axis=1)
w1=np.array([0.48009107, 0.19022909, 0.3296799,0])
Portfolio1 = (stock_return.mul(w1, axis=1)).sum(axis=1)
p0=cumulative_returns_plot(['Portfolio'])
p1=cumulative_returns_plot(['Portfolio1'])

portfolio = p0
portfolio_diff = portfolio.diff()
portfolio_diff = portfolio.dropna()

#plt.figure()
plt.plot(portfolio_diff)
plt.title('First Difference')
plt.show()
portfolio = pd.DataFrame(p0)
#portfolio.index=['portfolio']
portfolio
portfolio_week = portfolio['Portfolio'].resample('W-MON').mean()
portfolio_train = portfolio_week['2012':'2021']
fig=plt.figure()
ax1=fig.add_subplot(211)
ax2=fig.add_subplot(212)
acf = plot_acf(portfolio_diff, lags=20,ax=ax1,title="ACF")
pacf = plot_pacf(portfolio_diff, lags=20,ax=ax2,title="PACF")
plt.show()
# second difference
portfolio_diff2 = portfolio_train.diff(1)
portfolio_diff2 = portfolio_diff2.dropna()
for i in range(1):
    portfolio_diff2 = portfolio_diff2.diff(1)
    portfolio_diff2 = portfolio_diff2.dropna()
plt.figure()
plt.plot(portfolio_diff2)
plt.title('Second Difference')
plt.show()
fig=plt.figure()
```

```
ax1=fig.add_subplot(211)
ax2=fig.add_subplot(212)
acf = plot_acf(portfolio_diff2, lags=20,ax=ax1,title="ACF")
pacf = plot_pacf(portfolio_diff2, lags=20,ax=ax2,title="PACF")
plt.show()
portfolio_train[np.isnan(portfolio_train)] = 0
portfolio_train[np.isinf(portfolio_train)] = 0
# train
modelp = ARIMA(portfolio_train, order=(9, 2, 2),freq='W-MON')
resultp = modelp.fit()
pred_portfolio = resultp.predict('2020/09', dynamic=True, typ='levels')

print (pred_portfolio)

plt.figure(figsize=(6, 6))
plt.xticks(rotation=45)
plt.plot(pred_portfolio)
plt.plot(portfolio_train)
plt.show()
portfolio1 = p1
portfolio1_diff = portfolio1.diff()
portfolio1_diff = portfolio1.dropna()

#plt.figure()
plt.plot(portfolio1_diff)
plt.title('First Difference')
plt.show()
portfolio1 = pd.DataFrame(p1)
#portfolio.index=['portfolio']
portfolio1
portfolio1_week = portfolio1['Portfolio1'].resample('W-MON').mean()
portfolio1_train = portfolio1_week['2012':'2021']
fig=plt.figure()
ax1=fig.add_subplot(211)
ax2=fig.add_subplot(212)
acf = plot_acf(portfolio1_diff, lags=20,ax=ax1,title="ACF")
pacf = plot_pacf(portfolio1_diff, lags=20,ax=ax2,title="PACF")
plt.show()
# second difference
```

```
portfolio1_diff2 = portfolio1_train.diff(1)
portfolio1_diff2 = portfolio1_diff2.dropna()
for i in range(1):
    portfolio1_diff2 = portfolio1_diff2.diff(1)
    portfolio1_diff2 = portfolio1_diff2.dropna()
plt.figure()
plt.plot(portfolio1_diff2)
plt.title('Second Difference')
plt.show()
fig=plt.figure()
ax1=fig.add_subplot(211)
ax2=fig.add_subplot(212)
acf = plot_acf(portfolio1_diff2, lags=20,ax=ax1,title="ACF")
pacf = plot_pacf(portfolio1_diff2, lags=20,ax=ax2,title="PACF")
plt.show()
portfolio1_train[np.isnan(portfolio1_train)] = 0
portfolio1_train[np.isinf(portfolio1_train)] = 0
# train
modelp1 = ARIMA(portfolio1_train, order=(9, 1, 2),freq='W-MON')
resultp1 = modelp1.fit()
pred_portfolio1 = resultp1.predict('2020/09', dynamic=True, typ='levels')

print (pred_portfolio1)

plt.figure(figsize=(6, 6))
plt.xticks(rotation=45)
plt.plot(pred_portfolio1)
plt.plot(portfolio1_train)
plt.show()
```

Part 5: Generalize the Model

Portfolio_monte.py:

```
#!/usr/bin/env python
# coding: utf-8

# In[4]:
```

```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import datetime


# In[5]:



import tushare as ts
TOKEN = "e371c4002c53bc022bb788d62cbd234d74d28eb1a1c8f20f008a46e1"
ts.set_token(TOKEN)
pro = ts.pro_api()
data = pro.query('stock_basic', exchange='', list_status='L', fields='ts_code,symbol,name,area,industry,list_date')
data = data.dropna().reset_index()


# In[3]:

def cumulative_returns_plot(StockReturns,name_list):
    for name in name_list:
        CumulativeReturns = ((1+StockReturns[name]).cumprod()-1)
        CumulativeReturns.plot(label=name)
    plt.legend()
    plt.xticks(rotation=45)
    plt.xlabel("Date")
    plt.ylabel("Cumulative Return")
    plt.show()


# In[7]:

def get_mcw(stock_list):
    stocks = []
    for stock in stock_list:
        if (stock in data['ts_code'].values) == False:
            print("Wrong stock code!")
```

```
            return get_w()
        st = pro.daily(ts_code=stock, start_date="20010101",
                                        end_date="20211030",fields=["ts_code","trade_date", "close"])
        st.index = pd.to_datetime(st['trade_date'], format = "%Y/%m/%d")
        st = st[["close"]]
        st.rename(columns={'close':stock},inplace=True)
        stocks.append(st)
    StockPrices = pd.concat([stocks[i] for i in range(len(stocks))],axis='columns',names=['trade_date']).dropna()
    if StockPrices.empty == True:
        print("These stocks have not common trade data in tushare, select random stocks again.")
        return w_for_rand(len(stocks))
    StockReturns = StockPrices.pct_change().dropna()
    stock_return = StockReturns.copy()

    cov_mat = stock_return.cov()
    cov_mat_annual = cov_mat * 252

    # Sets the number of simulations
    number = 10000
    # Set an empty numpy array to store the weight, yield and standard deviation obtained from each simulation
    random_p = np.empty((number, len(stock_list)+2))
    # Set the seed of random number to make the result repeatable
    np.random.seed(5)

    for i in range(number):

        random5=np.random.random(len(stock_list))
        random_weight=random5/np.sum(random5)

        #Calculate the average annual rate of return
        mean_return=stock_return.mul(random_weight,axis=1).sum(axis=1).mean()
        annual_return=(1+mean_return)**252-1

        #Calculate the annualized standard deviation, which is also called volatility
        random_volatility=np.sqrt(np.dot(random_weight.T,np.dot(cov_mat_annual,random_weight)))

    #Store the weight generated above, the calculated yield and standard deviation into the array random_
```

P medium

```
            random_p[i][:len(stock_list)]=random_weight
            random_p[i][len(stock_list)]=annual_return
            random_p[i][len(stock_list)+1]=random_volatility

    RandomPortfolios=pd.DataFrame(random_p)
    RandomPortfolios.columns=[s +'_weight' for s in stock_list]+['Returns','Volatility']
    # Set the risk-free return rate to 0
    risk_free = 0
    # Calculate the sharp ratio for each asset
    RandomPortfolios['Sharpe'] = (RandomPortfolios.Returns - risk_free) / RandomPortfolios.Volatility
    max_index = RandomPortfolios.Sharpe.idxmax()
    numstocks=len(stock_list)
    MSR_weights = np.array(RandomPortfolios.iloc[max_index, 0:numstocks])

    StockReturns['Portfolio_MSR'] = stock_return.mul(MSR_weights, axis=1).sum(axis=1)
    print("---------------------------------------------------")
    print("The best weight for portfolio is: ",MSR_weights)
    print("---------------------------------------------------")
    print("The cumulative return of portfolio:")
    cumulative_returns_plot(StockReturns,['Portfolio_MSR'])
    return


def w_for_rand(n):
    stock_list = data['ts_code'][np.random.randint(0,len(data.index)-1,n)].values
    print("---------------------------------------------------")
    print("The stock portfolio is: ",stock_list)
    get_mcw(stock_list)
    return

# In[ ]:

def get_w():
    print("---------------------------------------------------")
    way = input("Choose your way:\nA: select random n stocks\nB: give the stock list\n")
    if way == "A":
        print("---------------------------------------------------")
        n = int(input("How many stocks do you want to buy?(input an integer)"))
```

```
        w_for_rand(n)
        return
    if way == "B":
        stock_list = []
        print("--------------------------------------------------")
        print('Which stocks do you want to buy(put stock code one by one end with nothing):\n')
        for stock in iter(input, ''):
            if stock == '':
                break
            else:
                stock_list.append(stock)
        get_mcw(stock_list)
        return
    else:
        print("--------------------------------------------------")
        print("Wrong value!")
        get_w()
    return
```

main.ipynb:

```
import Portfolio_monte
Portfolio_monte.get_w()
```