# Application of Feature-Modeling on Stock Market

##### **Shengyi Liang, Mohao Yi, Chi Yu Yeh, Jie Yu, Haikuo Lu, Yuxuan Zhang**

Boston University

…description

Submitted on …

**ABSTRACT**

With the purpose of providing suggestions to hedge fund managers, an efficient platform which can deal with lots of financial data and give out specific advice is needed. To solve it, we build up a self-running system which enables autmantical data download, cleaning, stock selection (including quantitative strategies / feature selection, training and prediction) and backtesting. It downloads the stock price and volume data ranging from 2010 to 2015 of the companies listed in S&P 500 from Yahoo Finance automatically. Since daily data contain much noise, we only use monthly data in the model. To deal with missing value, we replace NA value with the data of the previous day. When adjusting our model, we use different standards to classify the data. After cleaning the data, we can select the well-performed quantitative strategies / feature / alpha based on these data by neural network. Finally, through backtesting with selected features, eight most worthing buying companies' stocks will be picked out and given out as the suggestions to the fund managers. The whole process can be implemented by clicking a button.

***Index Terms-*** neural network, automantical system, finance, stock market

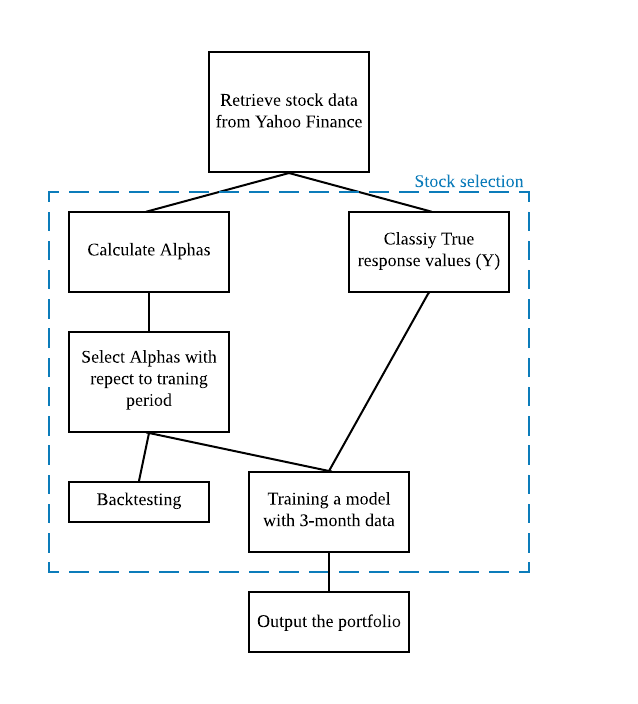
1. **INTRODUCTION**

It is really heavy workload for people to search for their target companies' stocks among a large number of data painstakingly. Faced with such a situation, we always hope to deal with these data in a clear-cut way once and for all, just like pressing a button. Actually, not just "like", we are able to have such a magic button now. What hides behind the magic button is a self-running system including data download and cleaning, stock selection (including quantitative strategies) and backtesting. We believe that such a structure enables a highly efficient, automantical and easily improved system. For data download and cleaning part, we use data of S&P 500 from yahoo finance, clean it by replacing NA values with the data of the previous day and make certain transformation of our data. For the stock selection and backtesting processes, neural network, one of common machine learning models in stock price prediction, is used for stock classification in our project. At first, financial technicles (alphas) with good performance are selected by neural network. And then, we do backtesting based on the selected financial technicals and screen out eight most worthing buying companies' stocks for each month. Besides the system itself, to encapsulate the whole system and create a user-friendly interface, we build up a GUI with the help of kivy package in Python.

1. **DATASET**

What we originally planned to work on is the data of S&P 500, such as the stock price on each day and other relative data of each company. Yahoo Finance provides detailed stock price data including the open and close price, the high and low price, the adjusted-close price and the trading volume on each day for the 500 companies in the list of S&P 500 [1]. Thanks to yfinance package in Python, we can obtain daily, weekly and monthly stock price of each company in any given period, making our automantical system run more conveniently. In our project, all of the financial technicles (alphas) that we have tested are on account of the data from Yahoo Finance.

1. **MODEL DESCRIPTION**
2. **General Design: Self-Running, Automation, theories**

Figure 1: the structure of general design

For the general design, the structure contains three sections: data download and cleaning (retrieve stock data from Yahoo Finance), stock selection (calculate alphas, select alphas with respect to training period, classify true response values, training and backtesting) and the output (output the portfolio), as shown in Figure 1. In the data download and cleaning section, data are obtained from Yahoo Finance by *yfinance* package in Python. We use stock price (close/open price, high/low price and adjusted close price) and volume data of each company listed in S&P 300. And, only monthly data are taken as our input in the project. To deal with the NA values, we use the data of previous day to replace the NA values. If all the values are missing, we just ignore the company. For stock selection, five parts work for the process, consisting of calculate alphas, select alphas with respect to training period, classify true response values, training and backtesting. Based on the selection process, we present the portfolio output for users. More details will be depicted in the later.

During the construction, we use functional programming to build up the structure. Functional programming allows us to build up our entire process by writing functions with basic functions. As we write complex functions, we are confident about certain lower functions as we have tested them in advance. As the platform is offered to hedge fund managers, we create the platform in functional programming style while keeping the process automated that any user could get the suggestions that they want by clicking one button.

1. **Data Processing: Classification, dataset composition**

As we have retrieved stock market data from Yahoo Finance, the stock prices and volumes are transferred into the values of “technicals” which we call “Alphas” and treat them as features extracted from the stock market. These Alphas come from the website Zura Kakushadze’s research paper “101 Formulaic Alphas”, including “101 real-life quantitative trading alphas” which have low correlation [2]. After this step, 82 feature values are extracted from each company.

For response variable (Y), since we use neural network as classifier, the type of response variable should be categorical. To begin with, we pay attention to the percentage returns of each company on each specific date. However, the percentage return is still continuous variable. Considering the fact that we only focus on whether the stock price will go up or down in the next month, we can just transform the response variable into binary one. Consequently, a classifier () is introduced: if the percentage of return is higher than or equal to, the response value will be 1; if the percentage of return is less than , the response value will be 0. Finally, we take 0.5 as our classifier in this transformation. For the upper half of companies with higher percentage return, their y values are 1; then, the lower half have 0 as their Y values.

1. **Feature Selection**

After the transformation of the data, we will our neural network model with the feature vectors and binary response values. With these data, we pack all companies’ data into one dataset so that we are able to have a relatively large sample size for training. We use 80% of the data for training and 20% of the data as the testing dataset. Through our training, however, we notice that not all Alphas are efficient and reliable as time passes. So, we devise a single alpha machine: for each Alpha, it performs five times cross-validation and stores the accuracy. Then, we select the 15 Alphas with the highest accuracy for future prediction because they are still viable now. In this process, we remove some of Alphas not suitable for monthly data which causes the accuracy as NA output.

1. **Training and Prediction**

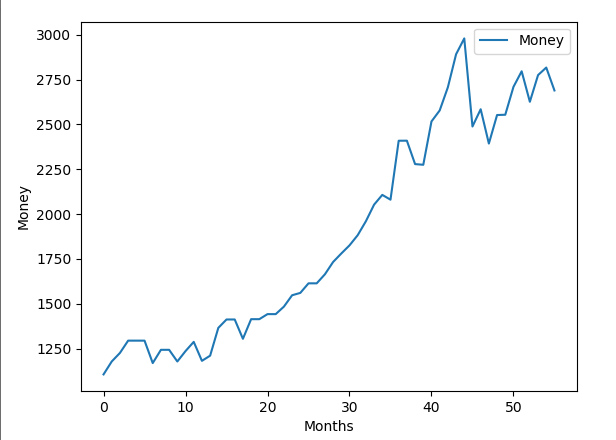
By using the 15 selected features, we train the model for prediction on rolling basis of three months. For example, in order to get a prediction for Aug 2nd, our code would train a neural network with data from May to July and make a prediction based on feature values of Aug 1st. At first, we make predictions for each company in this way and collect the predicted Alpha values. And then, we select the eight companies whose stock prices have the highest possibility of going up (the higher the values of the Alphas are, the more likely the company’s stock price is to go up), and obtain a portfolio value with these eight companies. As a result, our suggestion is to invest the eight companies. To avoid the noise like unusual news report, we ignore the combination of which the return percentage is higher than 20% in the process. The reason why we choose 20% as the boundary is that most of the companies have 20 percentage of return in our model.

1. **ANALYSIS OF THE MODEL**
2. **Comparison between Pairs of Training and Testing Datasets**

For simulation, we allow user to choose training period and testing period such that overfitting could be avoided. Also, since the economic crisis lasted from 2007-2009. We train and test the model with our code on multiple periods from 2010 to 2019.

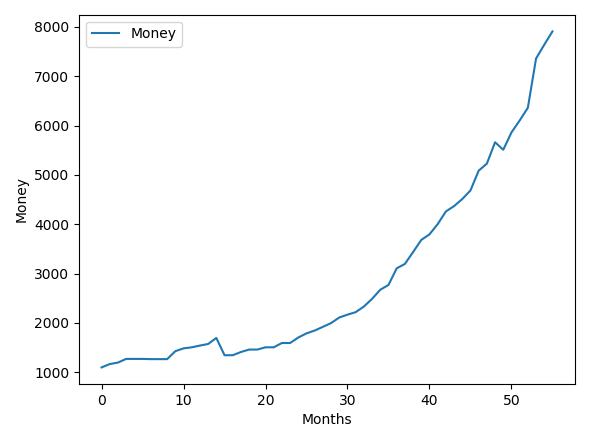
Multiple graphs here with results:

Train: 2010-2015 Test: 2014-2019



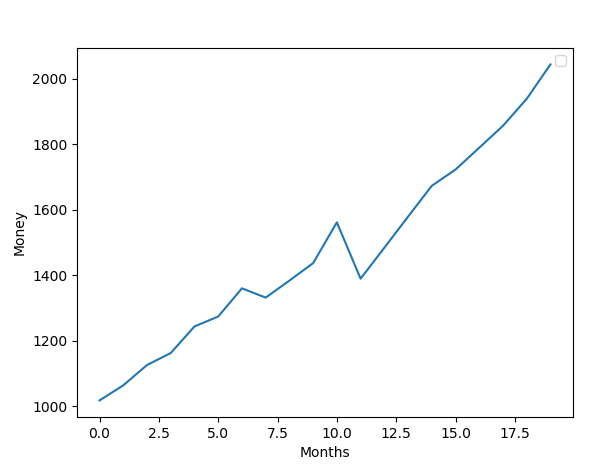
For this pair of data set, Alphas that work during period of 2010-2015 lose their power gradually especially after 2017.

Train: 2014-2019 Test: 2014-2019



For this pair of data set, selected Alphas correspond with the test period so that these Alphas perform well and should maintain their efficiency for 1-2 more years.

Train: 2017-2019 Test: 2017-2019

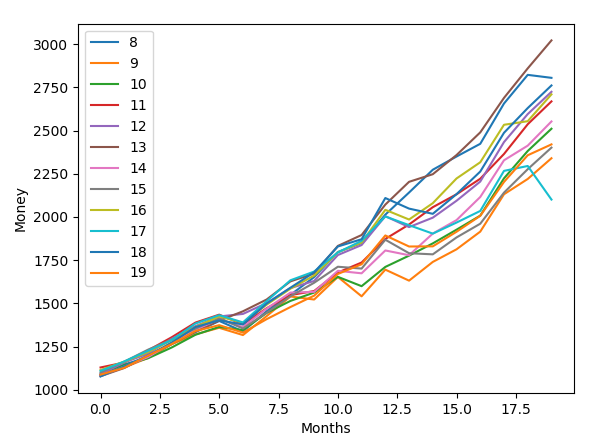


Moreover, as we change the length of period from 5 years to 2 years, outside factors become more influential that from Sep, 2018 to Dec, 2018, S&P 500 dropped about 10% whereas total of our simulated testing almost stayed unchanged.

Thus, Alphas selected more recently tend to perform more efficient than Alphas selected from earlier periods.

1. **the Graphs of a Range of Parameters for Each Parameter**

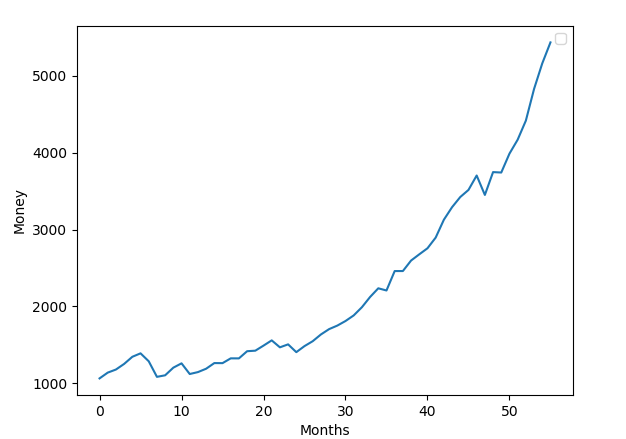
Number of selected Alphas:



From the graph, it is obvious that 13 is the most optimal number for selected alphas for period of 2 years. Moreover, including less or more may include Alphas that are losing their efficacy.

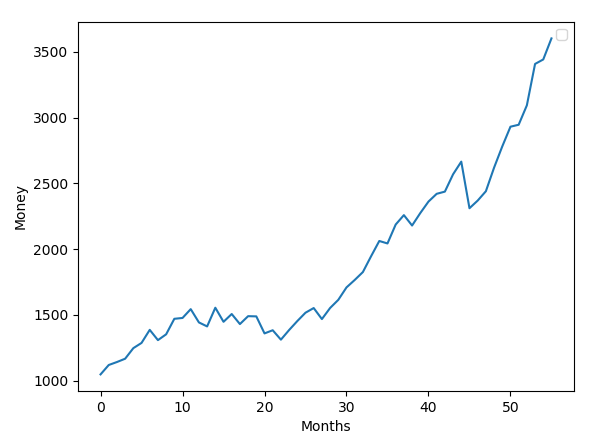
Classify Method:

50%:



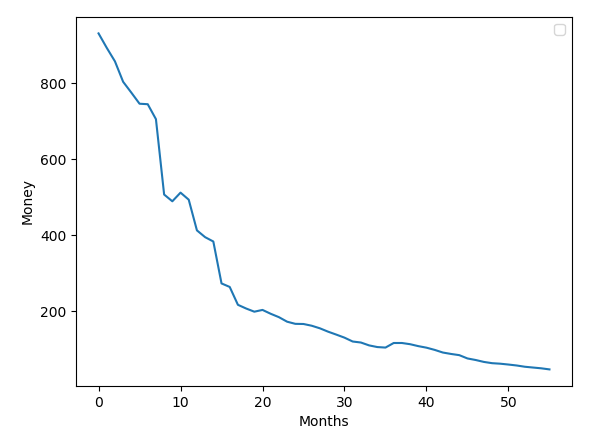
For each month, for the top 50% of percentage return, their responsive values are classified as 1 where the others are classified as 0. This method is the most stable and efficient one with highest total return on average.

Increase -> 1, Decrease -> 0:



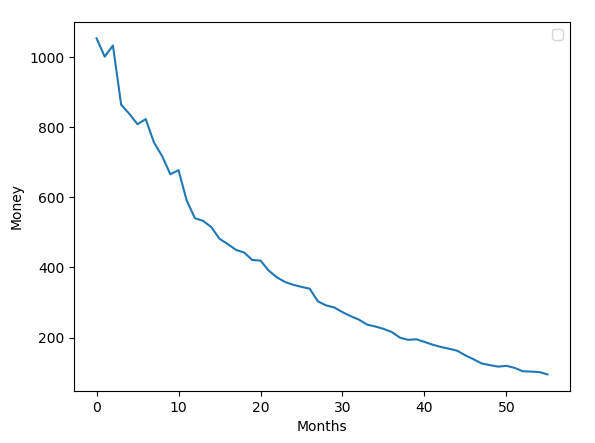
This method classifies increases in price as 1 and decreases in price as 0. It is not objective and efficient when most companies increase or decrease and generates unstable and biased results.

Rolling S&P 500:



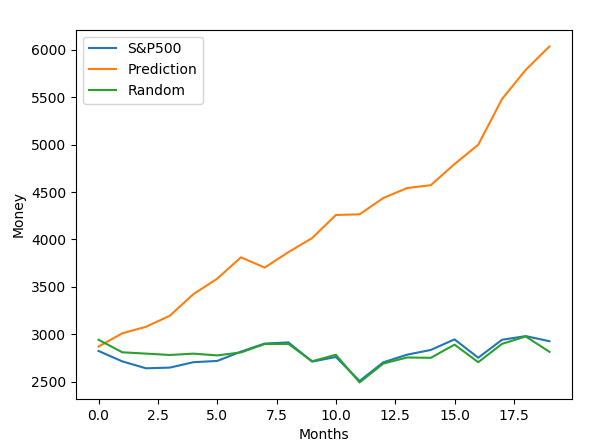
This method first calculates the average monthly increase of S&P 500 and classifies companies with percentage return higher than that as 1 and the others as 0. However, this classification seems to work well as a predictor for buying the shorts.

Actual S&P 500:



This method classifies companies with percentage returns higher than S&P 500 at each month as 1 and the others as 0. Similarly, this classification works well as indicator for buying shorts. In addition, it is more stable than Rolling S&P 500 as it shows a smoother curve.

1. **Performance Benchmarked with S&P 500**



Obviously, our continuous prediction has better performance than both random machine and S&P 500. Also, random selected companies follow the trend of S&P 500.

1. **Most Recent Predictions**

**July:** **['EQIX', 'CHTR', 'ROP', 'LH', 'BA', 'TRV', 'TMO', 'SRE']**

**EQIX 0.1079**

**CHTR 0.0628**

**ROP 0.0085**

**LH 0.0002**

**BA 0.0671**

**TRV 0.0023**

**TMO 0.0337**

**SRE 0.0457**

**Avg Return: 0.0410**

**S&P 500 Return: -0.0184**

**August:** **['ORLY','AMZN','MKTX','CELG','EQIX','SBAC','DLTR','KLAC']**

**ORLY 0.00026**

**AMZN 0.00762**

**MKTX 0.01129**

**CELG 0.00061**

**EQIX 0.01731**

**SBAC 0.01310**

**DLTR 0.00453**

**KLAC 0.01372**

**Avg Return: 0.00855**

**S&P 500 Return: -0.0069**

In the most recent predictions, our machine still provides acceptable portfolio that have higher return than S&P 500.

1. **Reality Check: whether codes have no bugs and our prediction involves no future information**

Coding bugs: Test on basic functions first and then test on higher level functions. This way, all lower level functions have been checked and we do not have to check back.

Algorithm: By calculating and graphing on all portfolios that test data has generated, we first make sure that all data points are referred correctly. Moreover, in order to check if we make predictions based on future information, we cut data set correspondingly and check if there is any discrepancy between the original and modified graphs

1. **DISCUSSION**

With outside factors, stock market is not perfectly balanced that there are certain features that could be calculated from original stock data. Also, the stock market works like a mind game for everyone. Certain strategy or feature does not necessarily work forever that it usually works for one to two years. Nonetheless, some outdated Technicals such as momentum may come back on stage in the future as recently viable Technicals lose their potency in the future. Thus, identifying and transforming efficient and appropriate Technicals or Alphas allow users to locate trends within the stock market thereby making profits from the stock market. Moreover, Technicals or Alphas selected from most recent 2-year period tend to be more efficient.

Graphs on multiple testing periods that our strategy of selecting appropriate Technicals works well on S&P 500 that there is a 3% increase in total deposit each month. Since S&P 500 covers companies from most industries, our strategy should work on other general stock indexes as well such as Dowjones.

Nonetheless, as we use a regular neural network to score each company, its simple structural design may not be the best for the stock market. Implementing more complex and optimized structure may improve the performance even further. On the other hand, we believe that, in addition to stock data, financial data from financial report would further enhance our accuracy but we did not import those data due to our limited access to data sources.

1. **CONCLUSION**

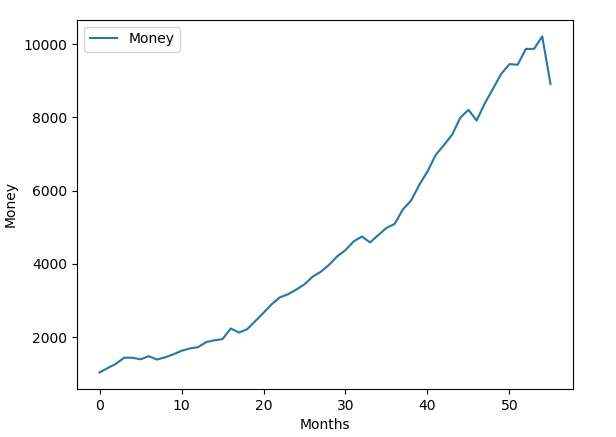
Conclusively, based on the fact that the stock market is tilted and not perfectly balanced, we believe there is a strategy to earn profit from the stock market. Then, by extracting features from stock prices and volumes, we can locate trends and indications from stock market thereby having positive profit in the stock market. Thus, based on those trends, we can score companies with respect to whether their prices would increase in the following month.

In the future, as long as users could invent new Technicals before the current ones lose their efficacy. Long term, or low frequency, trading allows them to make relatively stable profit from stock market that is not completely balanced.

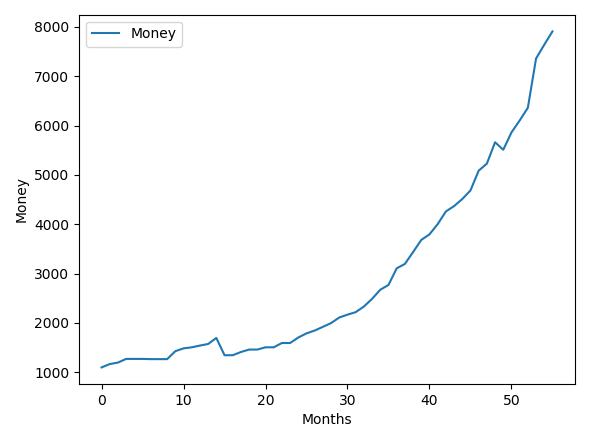
**APPENDIX**

1. Graphs

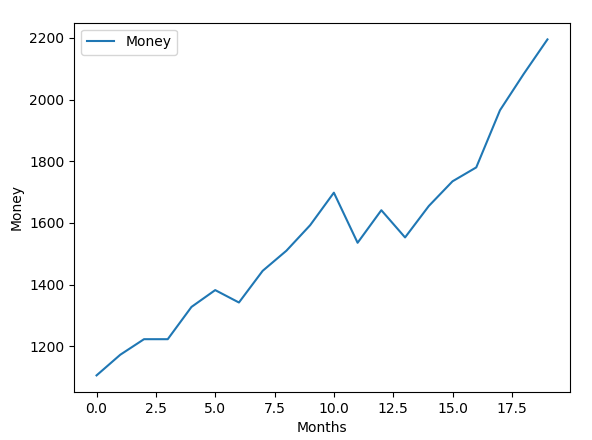
Train: 2010-2015 Test: 2010-2015:



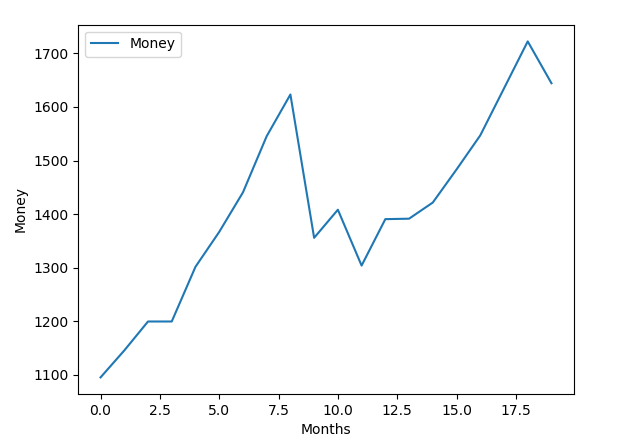
Train: 2014-2019 Test: 2014-2019



Train: 2017-2019 Test: 2017-2019



Train: 2012-2014 Test: 2017-2019



**1. the codes (Main)**

**Main.py:**

import pandas as pd

import tensorflow as tf

import numpy as np

import yfinance as yf

import stockstats as SS

from tensorflow.contrib import rnn

from tensorflow.keras import datasets, layers, models

import random

import datetime

from dateutil.relativedelta import relativedelta

import matplotlib.pyplot as plt

import alp1 # import alpha features in alp1.py

import Testing

import Prediction

import Alphas

import Process

import warnings

warnings.filterwarnings("ignore")

symbols = pd.read\_excel('SP500.xlsx')

symbols = list(symbols['Symbol'])

#print(symbols)

# Available types from Yahoo Finance

tem = ['adj close', 'close', 'high', 'low', 'open', 'volume']

def YFI(choice, startDate = '2010-01-01', endDate = '2015-01-01'):

'''import stock data from yahoo finance'''

# Get data from Yahoo Finance

# Turn into Stockstats dataframe, did some initial cleaning

dist = {}

# get monthly data from yahoo finance

if choice == 0:

for i in symbols:

dist[i] = SS.StockDataFrame.retype(yf.download(i, period = '5y', interval = '1mo', auto\_adjust = False))

else:

for i in symbols:

dist[i] = SS.StockDataFrame.retype(yf.download(i, start = startDate, end = endDate, interval = '1mo', auto\_adjust = False))

# Data cleaning

# delete empty dataframes or dataframes with large amount of NAs.

'''

today1 = datetime.date.today() - relativedelta(days = datetime.date.today().day-1)

poped = []

for i in dist.keys():

keyss = list(dist[i].index.values)

if not ((np.datetime64(today1)) in keyss):

poped += [i]

for k in poped:

dist.pop(k)

print(dist.keys())

'''

#dist.pop('BRKB')

#dist.pop('ABMD')

na = []

for k in dist:

if(dist[k].empty):

na = na+[k]

for i in na:

del dist[i]

# delete invalid data

for i in dist.keys():

dist[i] = dist[i][dist[i].close.isna() == False]

for i in dist.keys():

temp = dist[i].index.values

for j in temp:

if str(j)[8:10] != '01':

dist[i] = dist[i].drop(j)

for i in dist.keys():

for j in ['close', 'open', 'high', 'low']:

for k in range(dist[i][j].shape[0]-1):

a = dist[i][j][k]

b = dist[i][j][k+1].copy()

if a==b:

dist[i][j][k+1] = b+0.001

na = []

for k in dist:

if(dist[k].empty):

na = na+[k]

for i in na:

del dist[i]

return dist

def main():

'''main function for backtest etc'''

pp = []

choices = []

dist = YFI(0)

X, Y = Process.ProcessData(dist, 2)

dist2 = YFI(1)

X2, Y2 = Process.ProcessData(dist2, 2)

for i in Y2.keys():

a = Y2[i].copy()

a = np.where(a != 0, 1, 0)

a = np.where(a != 1, 0, 1)

Y2[i] = a

goodAlphas = ['alpha083','alpha101','alpha024','alpha042','alpha028','alpha025','alpha018','alpha010','alpha047','alpha033','alpha009','alpha005','alpha051']

goodAlphas1 = goodAlphas[:7]

goodAlphas2 = goodAlphas[7:]

#allAlphas = list(X['MSFT'].columns)

sums = []

'''

#for j in range(2):

alphaIndex = Alphas.SingleAlpha(X, Y, 15)

#alphaIndex = goodAlphas

print(alphaIndex)

sp5 = list(SP(1,2))[:-1]

#Plot returns for alphas

res1 = plotSingleAlpha(alphaIndex, X, Y, dist)

alphaIndex = Alphas.SingleAlpha(X2, Y2, 15)

res2 = plotSingleAlpha(alphaIndex, X, Y, dist)

print(sp5[3])

print('Train on same period:', res1)

print('Train on 2010-2015:', res2)

'''

alphaIndex = Alphas.SingleAlpha(X, Y, 15)

Date = datetime.datetime(year = 2019, month = 8, day = 1)

dataX, dataY, result, rmX, rmY = Testing.Backtest(60, alphaIndex, 1000, dist, X, Y, 10, stDate = Date)

print('Test and train on 2010-2015:', result)

print(result)

#Calculate percentage return on particular month

'''

target = datetime.datetime(year = 2016, month = 3, day = 1)

a = Testing.Designated(target, dist, X, Y, alphaIndex, 10)

print(a)

'''

#result = Prediction.AvgedPredict(dist, X, Y, alphaIndex, 10, 10)

return result

'''

return [1,2,3,4,5]

'''

def plotSingleAlpha(alphas, X, Y, dist):

sums = []

sp5 = list(SP(1,2))[:-1]

for i in alphas:

print(i)

dataX, dataY, result, rmX, rmY = Testing.Backtest(24, [i], sp5[3], dist, X, Y, 10)

plt.plot(np.asarray(dataX), np.asarray(dataY), label =( 'Alpha = ' + str(i)))

#plt.plot(np.asarray(rmX), np.asarray(rmY))

sums += [[result, i]]

sums.sort(reverse = True)

dataX, dataY, result, rmX, rmY = Testing.Backtest(24, alphas, sp5[3], dist, X, Y, 10)

plt.plot(np.asarray(dataX), np.asarray(dataY), label =('Cumulative'))

plt.plot(np.asarray(dataX), np.asarray(sp5[4:]), label = ('SP500'))

plt.xlabel('Months')

plt.ylabel('Money')

plt.legend()

plt.show()

return result

def SP(startD, endD):

dist = SS.StockDataFrame.retype(yf.download('^GSPC', period = '2y', interval = '1mo', auto\_adjust = False))

return dist['close']

#test area

print(main())

**2. the codes (Alphas)**

**Alphas.py:**

import pandas as pd

import tensorflow as tf

import numpy as np

import yfinance as yf

import stockstats as SS

from tensorflow.contrib import rnn

from tensorflow.keras import datasets, layers, models

import random

import datetime

from dateutil.relativedelta import relativedelta

import matplotlib.pyplot as plt

import alp1 # import alpha features in alp1.py

def SingleAlpha(X,Y, alphasize):

'''Single alpha machine to select alphas for actual predicting model'''

X\_train, X\_test = splitterX(X)

Y\_train, Y\_test = splitterY(Y)

X\_train, Y\_train = check(X\_train, Y\_train)

X\_test, Y\_test = check(X\_test, Y\_test)

#print(Y\_train)

alphas = X\_train.columns

accc = 0.5

#while (accc < 0.7):

alphaAcc = []

for i in alphas:

print(i)

modeli, iacc, t = trainSingleAlpha(X\_train[i], X\_test[i], Y\_train, Y\_test)

alphaAcc+= [[iacc, i, t]]

alphaAcc.sort(reverse=True)

#print(alphaAcc)

selectedAlphas = []

for j in range(len(alphaAcc)):

if alphaAcc[j][2]==1:

selectedAlphas += [alphaAcc[j]]

if len(selectedAlphas) >=alphasize:

selectedAlphas = selectedAlphas[:alphasize]

alphaIndex = extractAlpha(selectedAlphas)

#alphaIndex = ['alpha001', 'alpha101']

model, acc = train(X\_train[alphaIndex], X\_test[alphaIndex], Y\_train, Y\_test)

accc = acc

print('Final Accuracy:', acc)

return alphaIndex

def extractAlpha(lis):

'''get alpha names from 2-d list'''

res = []

for i in lis:

res+= [i[1]]

return res

def splitterX(dist):

'''Concat dataset for training and testing'''

newT = pd.DataFrame()

newR = pd.DataFrame()

for i in dist.keys():

cut = int(dist[i].shape[0]\*0.8)

train = dist[i].iloc[:cut]

test = dist[i].iloc[cut:]

newT = pd.concat([newT, train], axis=0, ignore\_index=True)

newR = pd.concat([newR, test], axis=0, ignore\_index=True)

return newT.copy(), newR.copy()

def splitterY(dist):

'''Concat dataset for training and testing'''

newT = np.asarray([])

newR = np.asarray([])

for i in dist.keys():

cut = int(dist[i].shape[0]\*0.8)

train = dist[i][:cut]

test = dist[i][cut:]

newT = np.append(newT, train)

newR = np.append(newR, test)

return newT.copy(), newR.copy()

def check(X, Y):

''' to keep X and Y in the same length'''

lx = len(X)

ly = len(Y)

if lx != ly:

temp = min(lx,ly)

return (X[:temp]), (Y[:temp])

else:

return X, Y

def trainSingleAlpha(X\_train, X\_test, Y\_train, Y\_test):

'''Train and test a regular neural network'''

model = models.Sequential()

model.add(layers.Flatten(input\_shape = [1]))

#model.add(layers.Dense(28000, activation = tf.nn.relu))

#model.add(layers.Dense(20000, activation = tf.nn.relu))

model.add(layers.Dense(512, activation = tf.nn.relu))

#model.add(layers.Dense(12000, activation = tf.nn.relu))

#model.add(layers.Dense(8000, activation = tf.nn.relu))

#model.add(layers.Dense(6000, activation = tf.nn.relu))

#model.add(layers.Dense(4000, activation = tf.nn.relu))

#model.add(layers.Dense(2000, activation = tf.nn.relu))

#model.add(layers.Dense(1000, activation = tf.nn.relu))

#model.add(layers.Dense(500, activation = tf.nn.relu))

model.add(layers.Dense(256, activation = tf.nn.relu))

model.add(layers.Dense(16, activation=tf.nn.relu))

model.add(layers.Dense(8, activation=tf.nn.softmax))

# model.summary()

model.compile(optimizer='adam',

loss='sparse\_categorical\_crossentropy',

metrics=['accuracy'])

model.fit(X\_train, Y\_train, epochs=5, verbose = 0)

test\_loss, test\_acc = model.evaluate(X\_test, Y\_test, verbose = 0)

print(test\_loss, test\_acc)

temp = 1

if pd.isna(test\_loss):

temp = 0

# print('Correct Prediction (%): ', accuracy\_score(Y\_test, model.predict(X\_test), normalize=True)\*100.0)

return model, test\_acc, temp

def train(X\_train, X\_test, Y\_train, Y\_test):

'''Train and test a regular neural network'''

model = models.Sequential()

model.add(layers.Flatten(input\_shape = [len(X\_train.columns)]))

#model.add(layers.Dense(28000, activation = tf.nn.relu))

#model.add(layers.Dense(20000, activation = tf.nn.relu))

model.add(layers.Dense(512, activation = tf.nn.relu))

#model.add(layers.Dense(12000, activation = tf.nn.relu))

#model.add(layers.Dense(8000, activation = tf.nn.relu))

#model.add(layers.Dense(6000, activation = tf.nn.relu))

#model.add(layers.Dense(4000, activation = tf.nn.relu))

#model.add(layers.Dense(2000, activation = tf.nn.relu))

#model.add(layers.Dense(1000, activation = tf.nn.relu))

#model.add(layers.Dense(500, activation = tf.nn.relu))

model.add(layers.Dense(256, activation = tf.nn.relu))

model.add(layers.Dense(16, activation=tf.nn.relu))

model.add(layers.Dense(8, activation=tf.nn.softmax))

# model.summary()

model.compile(optimizer='adam',

loss='sparse\_categorical\_crossentropy',

metrics=['accuracy'])

model.fit(X\_train, Y\_train, epochs=5, verbose = 0)

test\_loss, test\_acc = model.evaluate(X\_test, Y\_test, verbose = 0)

# print('Correct Prediction (%): ', accuracy\_score(Y\_test, model.predict(X\_test), normalize=True)\*100.0)

return model, test\_acc

**3.the codes(Testing)**

**Testing.py:**

import pandas as pd

import tensorflow as tf

import numpy as np

import datetime

from dateutil.relativedelta import relativedelta

from tensorflow.keras import datasets, layers, models

import random

import datetime

from dateutil.relativedelta import relativedelta

#import matplotlib.pyplot as plt

def Designated(Target, dist, X, Y, alphaIndex, portSize):

startDate = Target - relativedelta(months = 3)

endDate = Target - relativedelta(months = 1)

inputX = {}

inputY = {}

poped = []

for i in Y.keys():

indices = dist[i].index.values

if (np.datetime64(startDate) in indices) and (np.datetime64(endDate) in indices) and (np.datetime64(Target) in indices):

#print(i)

st, en = getIndex(dist[i], startDate, endDate)

#print(st,en)

inputX[i] = X[i].loc[np.datetime64(startDate):np.datetime64(endDate)]

inputY[i] = Y[i][st:en]

else:

poped += [i]

inputX = splitterX2(inputX)

inputY = splitterY2(inputY)

model = train2(inputX[alphaIndex], inputY)

scores = []

for i in X.keys():

if not (i in poped):

#print(i)

b = model.predict(X[i][alphaIndex].loc[np.datetime64(startDate):np.datetime64(Target)])

#print(b)

scores += [[b[-1][0], i]]

scores.sort(reverse=True)

res = []

for j in scores[:10]:

res+=[j[1]]

temp = []

for k in res:

Target2 = Target + relativedelta(months = 1)

a = dist[k]['close'][Target]

b = dist[k]['close'][Target2]

pctr = (b-a)/a

temp += [pctr]

return sum(temp)/len(temp)

def getIndex(dist, start, end):

indices = dist.index.values

startd = np.where(indices == np.datetime64(start))[0][0]

endd = np.where(indices == np.datetime64(end))[0][0]

return startd, endd

def Backtest(numOfMonth, alphaIndex, initial, dist, X, Y, portSize, stDate = datetime.date.today() - relativedelta(days = datetime.date.today().day-1)):

'''test on model and alphas on its performance on a period of time'''

actinitial = initial

rminitial = initial

handinitial = initial

#today1 = datetime.date.today() - relativedelta(days = datetime.date.today().day-1)

dataX = []

dataY = []

rmX = []

rmY = []

handX = []

handY = []

rmports = RandomPort(numOfMonth-1, list(dist.keys()))

for i in range(numOfMonth-1):

startD = i

startDate = stDate - relativedelta(months=numOfMonth-i-1)

endDate = stDate - relativedelta(months=numOfMonth-i-3)

endD = i+2

print(startDate,endDate)

tempDist, tempX, tempY = ExtractDist(dist.copy(), X.copy(), Y.copy(), startDate, endDate, startD, endD)

tempX = splitterX2(tempX)

tempY = splitterY2(tempY)

if tempX.empty == False:

model = train2(tempX[alphaIndex], tempY)

Portfolio, AvgPctr = SelectAndPCTR(model, dist, X.copy(), alphaIndex, startD, endD, startDate, endDate, portSize)

print(Portfolio, AvgPctr)

rmport = rmports[i]

rmpctr = CalcPctr(rmport, endDate + relativedelta(months = 1), dist)

handpctr = CalcPctr(Portfolio, endDate + relativedelta(months = 1), dist)

#print(Portfolio, AvgPctr, rmpctr, handpctr)

handinitial = handinitial \* (1+handpctr)

rminitial = rminitial \* (1+rmpctr)

rmX += [i]

rmY += [rminitial]

handX += [i]

handY += [handinitial]

if AvgPctr <0.2:

initial = initial\*(1+AvgPctr)

dataX += [i]

dataY += [initial]

else:

dataX += [i]

dataY += [initial]

'''

plt.plot(np.asarray(handX), np.asarray(handY), label = 'Projected Return')

plt.xlabel('Months')

plt.ylabel('Money')

plt.title("Change of money on the past 2 years")

plt.legend()

plt.show()

print('Initial Money:', actinitial, 'Resulting Money:', initial)

'''

return dataX, dataY, initial, rmX, rmY

def CalcPctr(companies, date, dist):

'''calculate percentage return of certain portfolio on certain date'''

temp = 0

for j in companies:

if (np.datetime64(date) in dist[j].index.values) and (np.datetime64(date+relativedelta(months=1)) in dist[j].index.values):

pctr = (dist[j]['close'][np.datetime64(date+relativedelta(months = 1))] - dist[j]['close'][np.datetime64(date)]) / dist[j]['close'][np.datetime64(date)]

temp += pctr / len(companies)

return temp

def ExtractDist(dist, X, Y, startDate, endDate, startD, endD):

'''Extract needed data for rolling window of training set'''

tempDist = {}

tempX = {}

tempY = {}

for j in dist.keys():

indices = dist[j].index.values

if (np.datetime64(startDate) in indices) and (np.datetime64(endDate) in indices):

#startD, endD = caliDate(startD, endD, startDate, endDate, indices)

if checkdate(startDate, indices[0], endDate+relativedelta(months=2), indices[-1]):

'''Data is valid for selected time'''

#print(j)

startD, endD = caliDate(startD, endD, startDate, endDate, indices)

tempDist[j] = dist[j].loc[startDate:endDate]

tempX[j] = X[j].loc[startDate:endDate]

tempY[j] = Y[j][startD:endD+1]

return tempDist.copy(), tempX.copy(), tempY.copy()

def checkdate(start, startc, end, endc):

'''check if data of index is available for certain company'''

start = np.datetime64(start)

end = np.datetime64(end)

if start >= startc and end <= endc:

return True

else:

return False

def SelectAndPCTR(model, dist, X, alphaIndex, startD, endD, startDate, endDate, portSize):

'''Use model to predict on alpha values and select portfolio'''

scores = []

for i in dist:

indices = dist[i].index.values

if (np.datetime64(startDate) in indices) and (np.datetime64(endDate) in indices) and (np.datetime64(endDate+relativedelta(months=1)) in indices) and (np.datetime64(endDate+relativedelta(months=2)) in indices):

if checkdate(startDate, indices[0], endDate+relativedelta(months=2), indices[-1]):

#print(i, dist[i])

#startD, endD = caliDate(startD, endD, startDate, endDate, indices)

a = (dist[i]['close'].loc[np.datetime64(endDate+relativedelta(months=2))] - dist[i]['close'].loc[np.datetime64(endDate+relativedelta(months=1))])/dist[i]['close'].loc[np.datetime64(endDate+relativedelta(months=1))]

b = model.predict(X[i][alphaIndex].loc[np.datetime64(endDate):np.datetime64(endDate+relativedelta(months=1))])

scores += [[b[-1][0], i, a]]

scores.sort(reverse=True)

temp = 0

for i in scores[:portSize]:

temp+=i[2]/portSize

res = []

for j in scores[:portSize]:

res+=[j[1]]

return res, temp

def caliDate(startD, endD, startDate, endDate, indices):

'''calibrate date of index'''

sd = np.datetime64(startDate)

ind = np.where(indices == sd)[0]

if len(ind) == 0:

return startD, endD

else:

return ind[0], ind[0]+2

def splitterX2(dist):

'''Concat dataset for rolling predicting and training'''

newT = pd.DataFrame()

for i in dist.keys():

train = dist[i]

newT = pd.concat([newT, train], axis=0, ignore\_index=True)

return newT.copy()

def splitterY2(dist):

'''Concat dataset for rolling predicting and training'''

newT = np.asarray([])

for i in dist.keys():

train = dist[i]

newT = np.append(newT, train)

return newT.copy()

def check(X, Y):

''' to keep X and Y in the same length'''

lx = len(X)

ly = len(Y)

if lx != ly:

temp = min(lx,ly)

return (X[:temp]), (Y[:temp])

else:

return X, Y

def train2(X,Y):

'''Train and test a regular neural network'''

X, Y = check(X,Y)

model = models.Sequential()

model.add(layers.Flatten(input\_shape = [len(X.columns)]))

#model.add(layers.Dense(28000, activation = tf.nn.relu))

#model.add(layers.Dense(20000, activation = tf.nn.relu))

model.add(layers.Dense(512, activation = tf.nn.relu))

#model.add(layers.Dense(12000, activation = tf.nn.relu))

#model.add(layers.Dense(8000, activation = tf.nn.relu))

#model.add(layers.Dense(6000, activation = tf.nn.relu))

#model.add(layers.Dense(4000, activation = tf.nn.relu))

#model.add(layers.Dense(2000, activation = tf.nn.relu))

#model.add(layers.Dense(1000, activation = tf.nn.relu))

#model.add(layers.Dense(500, activation = tf.nn.relu))

model.add(layers.Dense(256, activation = tf.nn.relu))

model.add(layers.Dense(16, activation=tf.nn.relu))

model.add(layers.Dense(8, activation=tf.nn.softmax))

# model.summary()

model.compile(optimizer='adam',

loss='sparse\_categorical\_crossentropy',

metrics=['accuracy'])

model.fit(X, Y, epochs=5, verbose = 0)

# print('Correct Prediction (%): ', accuracy\_score(Y\_test, model.predict(X\_test), normalize=True)\*100.0)

return model

def RandomPort(months, keys):

'''Generate Random Portfolios'''

syms = keys

ports = []

for i in range(months):

random.shuffle(syms)

ports += [syms[:20]]

return ports

**4. the codes (Prediction)**

**Prediction.py:**

import pandas as pd

import tensorflow as tf

import numpy as np

import yfinance as yf

import stockstats as SS

from tensorflow.contrib import rnn

from tensorflow.keras import datasets, layers, models

import random

import datetime

from dateutil.relativedelta import relativedelta

import matplotlib.pyplot as plt

def AvgedPredict(dist, X, Y, alphaIndex, n, portsize):

'''Get averaged portfolio for future prediction'''

res = []

for j in range(n):

res += [LatestPredict(dist, X, Y, alphaIndex)]

temp = {}

temp2 = []

for k in res:

for l in k:

if l in temp.keys():

temp[l] += 1

else:

temp[l] = 1

for i in temp.keys():

temp2 += [[temp[i], i]]

temp2.sort(reverse=True)

temp2 = temp2[:portsize]

result = []

for i in temp2:

result += [i[1]]

return result

def LatestPredict(dist, X, Y, alphaIndex):

'''Return the return for current month'''

today1 = datetime.date.today() - relativedelta(days = datetime.date.today().day-1)

startDate = today1 - relativedelta(months=3)

endDate = today1

inputX = {}

inputY = {}

poped = []

for i in dist.keys():

indices = dist[i].index.values

if (np.datetime64(startDate) in indices) and (np.datetime64(endDate) in indices):

inputX[i] = X[i].loc[np.datetime64(startDate):np.datetime64(endDate-relativedelta(months=1))]

inputY[i] = Y[i][-4:-1]

else:

poped += [i]

#print(inputX)

inputX = splitterX2(inputX)

inputY = splitterY2(inputY)

#print(inputX.shape, inputY.shape)

model = train2(inputX[alphaIndex], inputY)

scores = []

for i in dist.keys():

if not (i in poped):

b = model.predict(X[i][alphaIndex].loc[np.datetime64(startDate):np.datetime64(endDate)])

scores += [[b[-1][0], i]]

scores.sort(reverse=True)

res = []

for j in scores[:10]:

res+=[j[1]]

return res

def splitterX2(dist):

'''Concat dataset for rolling predicting and training'''

newT = pd.DataFrame()

for i in dist.keys():

train = dist[i]

newT = pd.concat([newT, train], axis=0, ignore\_index=True)

return newT.copy()

def splitterY2(dist):

'''Concat dataset for rolling predicting and training'''

newT = np.asarray([])

for i in dist.keys():

train = dist[i]

newT = np.append(newT, train)

return newT.copy()

def train2(X,Y):

'''Train and test a regular neural network'''

X, Y = check(X,Y)

model = models.Sequential()

model.add(layers.Flatten(input\_shape = [len(X.columns)]))

#model.add(layers.Dense(28000, activation = tf.nn.relu))

#model.add(layers.Dense(20000, activation = tf.nn.relu))

model.add(layers.Dense(512, activation = tf.nn.relu))

#model.add(layers.Dense(12000, activation = tf.nn.relu))

#model.add(layers.Dense(8000, activation = tf.nn.relu))

#model.add(layers.Dense(6000, activation = tf.nn.relu))

#model.add(layers.Dense(4000, activation = tf.nn.relu))

#model.add(layers.Dense(2000, activation = tf.nn.relu))

#model.add(layers.Dense(1000, activation = tf.nn.relu))

#model.add(layers.Dense(500, activation = tf.nn.relu))

model.add(layers.Dense(256, activation = tf.nn.relu))

model.add(layers.Dense(16, activation=tf.nn.relu))

model.add(layers.Dense(8, activation=tf.nn.softmax))

# model.summary()

model.compile(optimizer='adam',

loss='sparse\_categorical\_crossentropy',

metrics=['accuracy'])

model.fit(X, Y, epochs=5, verbose = 0)

# print('Correct Prediction (%): ', accuracy\_score(Y\_test, model.predict(X\_test), normalize=True)\*100.0)

return model

def check(X, Y):

''' to keep X and Y in the same length'''

lx = len(X)

ly = len(Y)

if lx != ly:

temp = min(lx,ly)

return (X[:temp]), (Y[:temp])

else:

return X, Y

**5.the codes (Process)**

**Process.py:**

import pandas as pd

import tensorflow as tf

import numpy as np

import yfinance as yf

import stockstats as SS

from tensorflow.contrib import rnn

from tensorflow.keras import datasets, layers, models

import random

import datetime

from dateutil.relativedelta import relativedelta

import matplotlib.pyplot as plt

import alp1 # import alpha features in alp1.py

def ProcessData(dist, choice):

'''Prepare X and Y for model'''

X = GetAlphasAll(dist)

Y = TrueYTransform(dist, choice)

return X.copy(), Y.copy()

def TrueYTransform(dist, choice):

'''rescale price (Y) into 0/1'''

if choice == 0:

'''Decrease in price = 0, increase or equal = 1'''

new = choice0(dist)

elif choice == 1:

'''If percentage return is higher than that of S&P 500, Y = 1, otherwise Y = 0'''

new = choice1(dist)

elif choice == 2:

'''Top x% pctr companies have Y = 1, others have Y = 0'''

new = choice2(dist, 0.5)

elif choice == 3:

'''PCTR > rolling S&P = 1'''

new = choice3(dist)

return new

def choice0(dist):

'''Decrease in price = 0, increase or equal = 1'''

new = {}

for i in dist.keys():

new[i]=np.asarray([])

for j in range(dist[i].shape[0]-1):

temp1 = dist[i]['close'][j]

temp2 = dist[i]['close'][j+1]

if temp1 < temp2:

new[i] = np.append(new[i], [0])

else:

new[i] = np.append(new[i], [1])

return new

def choice1(dist):

'''If percentage return is higher than that of S&P 500, Y = 1, otherwise Y = 0'''

new = {}

sppctrs = SPpctr()

for i in dist.keys():

new[i]=np.asarray([])

indices = dist[i].index.values

if indices[0] == np.datetime64('2017-08-01T00:00:00.000000000'):

indices = indices[1:]

#print(i, indices)

#print(list(sppctrs))

for j in range(indices.shape[0]-1):

temp1 = dist[i]['close'][j]

temp2 = dist[i]['close'][j+1]

pctr = (temp2-temp1)/temp1

if pctr <= list(sppctrs)[j]:

new[i] = np.append(new[i], [0])

else:

new[i] = np.append(new[i], [1])

return new

def choice2(dist, cut):

'''Top x% pctr companies have Y = 1, others have Y = 0'''

pctrs = {}

for i in dist.keys():

pctrs[i] = GetAlphas(dist[i])['pctr']

#new = {}

#temp = pctrs['AES'].shape[0]

indices = pctrs['MSFT'].index.values

for j in range(len(indices)):

allpctr = []

availCompanies = []

for i in dist.keys():

indi = pctrs[i].index.values

if indices[j] in indi:

allpctr += [pctrs[i].loc[indices[j]]]

availCompanies += [i]

allpctr.sort(reverse = True)

splitter = int(cut\*len(allpctr))

cutter = allpctr[splitter]

for i in availCompanies:

tem = pctrs[i].loc[indices[j]]

if tem < cutter:

pctrs[i].loc[indices[j]] = 1

else:

pctrs[i].loc[indices[j]] = 0

for i in pctrs.keys():

pctrs[i] = np.asarray(pctrs[i])

return pctrs

def choice3(dist):

'''PCTR > rolling S&P = 1'''

cutter =( ((3000-2470)/2470) \*\* (1/float(24))) -1

new = {}

for i in dist.keys():

new[i]=np.asarray([])

indices = dist[i].index.values

if indices[0] == np.datetime64('2014-07-01T00:00:00.000000000'):

indices = indices[1:]

#print(i, indices)

for j in range(indices.shape[0]-1):

temp1 = dist[i]['close'][j]

temp2 = dist[i]['close'][j+1]

pctr = (temp2-temp1)/temp1

if pctr <= cutter:

new[i] = np.append(new[i], [0])

else:

new[i] = np.append(new[i], [1])

return new

def GetAlphasAll(dist):

'''Return dataframe of companies with corresponding alpha values'''

df = {}

for i in dist.keys():

temp = GetAlphas(dist[i].copy())

if (temp.empty==False):

df[i] = alp1.get\_alpha(temp).drop(['adj close', 'close', 'high', 'low', 'open', 'volume', 'amount', 'pctr'], axis=1).fillna(value = 0)

return df

def GetAlphas(df):

'''return the percentage return and quantum'''

new = df.copy()[:-1]

pctr = []

amount = []

for i in range(df.shape[0]-1):

pctr += [(df['close'][i+1]-df['close'][i])/df['close'][i]]

amount += [df['close'][i]\*df['volume'][i]]

new['pctr'] = pctr

new['amount'] = amount

return new

def SPpctr():

'''Get list of percentage return of S&P 500'''

prices = yf.download('^GSPC', period = '2y', interval = '1mo', auto\_adjust = False)

indices = prices.index.values

#print(indices[0])

prices = prices[prices.Close.isna() == False]

for j in indices:

if str(j)[8:10] != '01':

prices = prices.drop(j)

new = []

indices = prices.index.values

for k in range(len(indices)-1):

a = prices['Close'][k]

b = prices['Close'][k+1]

pctr = (b-a)/a

new+=[pctr]

indices = prices.index.values

prices = prices.drop(indices[-1])

prices['pctr'] = new

return prices.drop(['Close', 'Open', 'High','Low','Adj Close','Volume'], axis = 1)['pctr']

**6.the codes (alp1)**

**alp1.py:**

This file comes from World Quant.

**REFERENCES**

[1] Yahoo - Business Finance, Stock Market, Quotes, News. *Yahoo! Finance*. Retrieved

from http://finance.yahoo.com/

[2] Zura Kakushadze, “101 Formulaic Alphas.” *Free University of Tbilisi, Business School & School of*  *Physics 240, David Agmashenebeli Alley, Tbilisi, 0159, Georgia.* December 9, 2015