Investment Management and Machine Learning

Week 6 Project

Robert Gorman

import pandas as pd

import yfinance as yf

import numpy as np

from scipy.optimize import minimize

from datetime import datetime

import matplotlib.pyplot as plt

# Stock portfolio

tickers = ['QQQ', 'IWB', 'VONE']

start\_date = '2013-01-01'

end\_date = '2024-01-01'

# Download historical price data

stock\_data = yf.download(tickers, start=start\_date, end=end\_date)['Adj Close']

stock\_data.reset\_index(inplace=True)

stock\_data['Year']=stock\_data['Date'].dt.year

stock\_data['Month']=stock\_data['Date'].dt.month

stock\_data.rename(columns={'QQQ':'fund1','IWB':'fund2','VONE':'fund3'},inplace=True)

stock\_data[['fund1\_ret\_d','fund2\_ret\_d','fund3\_ret\_d']]=stock\_data[['fund1','fund2','fund3']].pct\_change()

stock\_data.sort\_values(by=['Date'], inplace=True)

stock\_data[['fund1\_ret\_d+1','fund2\_ret\_d+1','fund3\_ret\_d+1']]=stock\_data[['fund1\_ret\_d','fund2\_ret\_d','fund3\_ret\_d']]+1

stock\_data1=stock\_data[['fund1\_ret\_d+1','fund2\_ret\_d+1','fund3\_ret\_d+1','Year','Month']].groupby(['Year','Month']).prod()

stock\_data1[['fund1\_ret\_m','fund2\_ret\_m','fund3\_ret\_m']]=stock\_data1[['fund1\_ret\_d+1','fund2\_ret\_d+1','fund3\_ret\_d+1']]-1

stock\_data2=stock\_data1[['fund1\_ret\_m','fund2\_ret\_m','fund3\_ret\_m']]

print(stock\_data2)

returns=stock\_data2

weight=[0.1,0.1,0.1]

# Multiply each fund's return with its weight

weighted\_returns = returns.multiply(weight)

# Calculate the weighted sum of returns for each year-month

portfolio\_returns = weighted\_returns.sum(axis=1)

# Print or use the resulting Series as needed

print('\n', portfolio\_returns)

#find the average portfolio monthly return

def pret(weight):

pret1=returns.multiply(weight).sum(axis=1)

pret1\_mean=pret1.mean() #find the average portfolio return

return pret1\_mean

print('AVG Portfolio Monthly Return: ',pret(weight))

#find the volatililty of portfolio returns

def pvol(weight):

pret1=returns.multiply(weight).sum(axis=1)

pret1\_vol=pret1.std()\*np.sqrt(12)#annualize volatlitiy based on monthly returns

return pret1\_vol

print('Volatility of Portfolio Returns: ', pvol(weight))

"""Search for the optimal asset allocation"""

#set values for parameters

target\_vol=[0.15]

#target\_vol is the target annualized portfolio volatility

no\_fund=3

#no\_fund indicates the number of ETFs in the portfolio

#give initial values for weights

weight=[0.1,0.1,0.1]

returns=stock\_data2

def search\_weight(weight, returns, target\_vol, no\_fund):

def pret(weight):

pret1=returns.multiply(weight).sum(axis=1)

#find the average portfolio return

#we use the minimize function, so we mutiply portfolio average return with negative one

pret1\_mean\_flip=-pret1.mean()

return pret1\_mean\_flip

def pvol(weight,target\_vol):

pret1=returns.multiply(weight).sum(axis=1)

#find the volatililty of portfolio returns equals to the target\_vol

pret1\_vol=pret1.std()\*np.sqrt(12)-target\_vol

return pret1\_vol

# For any portfolio, the sum of all weights should be 1

#a function to make sure the sum of weights =1

def sumweight(weight):

return weight.sum()-1

#use scipy library-minimize to search for weights

solve1=minimize(pret, weight,

constraints=({"fun": pvol, "type": "eq", 'args': target\_vol},{"fun": sumweight, "type": "eq"}),

bounds=[(0,1)]\*no\_fund)

#the weight of each of the give ETFs should be between 0 and 1; bounds=[(0,1)]\*no\_fund

#report selected weights

weight\_select=solve1.x

#report annualized average portfolio return based on selected weights

portfolio\_ret=-solve1.fun\*12

#report whether the search is successful

success=solve1.success

return portfolio\_ret, weight\_select, success;

# Get the returning values from the function

portfolio\_ret, weight\_select, success = search\_weight(weight, returns, target\_vol, no\_fund)

# Print the results

print("Success:", success)

print("Portfolio Return:", portfolio\_ret)

print("Optimal Weights:", weight\_select)

#####################Question 4

"""Simulate returns using random numbers from normal distribution"""

#simulate monthly portfolio returns

portfolio\_ret=0.04134428041883573

target\_vol=0.25

no\_month=120

no\_simulation=10000

#simulate portfolio returns

pret\_sim1=np.random.normal(portfolio\_ret/12,target\_vol/np.sqrt(12),size=(no\_month,no\_simulation))

annualfee=0.0035

pret\_sim2=pret\_sim1-annualfee/12

"""Simulate account balance"""

#account balance at the end of the first month

value=0

monthlypayment=500

value=(value+monthlypayment)\*(1+pret\_sim2[0,:])

balance=[]

balance.append(value)

balance1=pd.DataFrame(balance)

#account balance at the end of each month

value = 0

balance = []

no\_month = min(50, len(pret\_sim2)) # Adjust to the minimum of 50 and the size of pret\_sim2

for i in range(no\_month):

value = (value + monthlypayment) \* (1 + pret\_sim2[i, :])

balance.append(value)

balance1 = pd.DataFrame(balance)

balance1['month\_no'] = balance1.index + 1

#The function to calculate accountbalance

def accountbalance(age\_current, age\_retire, monthlypayment, no\_simulation, annualfee, portfolio\_ret, target\_vol):

#Define the variables in def(accountbalance)

no\_month=(age\_retire-age\_current)\*12

#simulate returns using random numbers from normal distribution

pret\_sim1=np.random.normal(portfolio\_ret/12,target\_vol/np.sqrt(12),size=(no\_month,no\_simulation))

#take into account advisory fees

pret\_sim2=pret\_sim1-annualfee/12

#simulate account balance over time

value=0

balance=[]

for i in range (no\_month):

value=(value+monthlypayment)\*(1+pret\_sim2[i,:])

balance.append(value)

balance1=pd.DataFrame(balance)

balance1['month\_no']=balance1.index+1

#Reshape the balance1 file

balance2=pd.melt(balance1, id\_vars=['month\_no'], var\_name='Sim\_no', value\_name='balance')

#We set the median account balance in each month as the balance under normal market condition

#With 50% chance, the account balance is at least this amount

normal1=balance2[['month\_no','balance']].groupby(['month\_no']).quantile(0.5)

normal1['balance\_m']=normal1['balance']/1000000

#We set the 10th percentile account balance in each month as the balance under weak market condition

#With 90% chance, the account balance is at least this amount

weak1=balance2[['month\_no','balance']].groupby(['month\_no']).quantile(0.1)

weak1['balance\_m']=weak1['balance']/1000000

return normal1,weak1;

#Before we run the accountbalance function, We should set the target\_vol and run the search\_weight function

no\_fund=3

weight=[0.1,0.1,0.1]

returns=stock\_data2

target\_vol=[0.15] #set target volatility

#target\_vol is the annualized portfolio volatility

portfolio\_ret,weight\_select,success=search\_weight(weight,returns,target\_vol,no\_fund)

success

portfolio\_ret

#Run the search\_weight function help use obtain the portfolio\_ret for a specific target\_vol

#We give the portfolio\_ret value to the accountbalance function

#filled in my numbers for the following variables to plot

normal1, weak1=accountbalance(25, 60, 450, 10000, 0.0035,0.1381, 0.15)

plt.rcParams['ytick.right'] = plt.rcParams['ytick.labelright'] = True

plt.rcParams['ytick.left'] = plt.rcParams['ytick.labelleft'] = False

plt.plot(normal1['balance\_m'], label="normal market")

plt.plot(weak1['balance\_m'], label='weak market')

plt.xlabel("No. of months", size=15)

plt.title("Account Balance ($million)", size=36)

plt.xticks(size=22)

plt.yticks(size=22)

plt.legend(fontsize=22)

plt.show()



