Robert Gorman

Investment Management and Machine Learning

Week 9 HW

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import matplotlib.dates as dates

import statsmodels.api as sm

plt.rcParams['figure.figsize'] = [20, 15]

sample1 = pd.read\_stata(r"C:\Users\rdg83\OneDrive - Rutgers University\Course Investment Portfolio Management\Week 9 Stuff\finalsample.dta")

sample1.sort\_values(by=['datadate'], inplace=True)

var\_remove = ['PE', 'BM']

sample2 = sample1.drop(var\_remove, axis=1)

sample2['Year']=sample2['datadate'].dt.year

sample2['Month']=sample2['datadate'].dt.month

sample2=sample1[sample1['lagPrice2']>=5]#remove penny stocks

sample2['Year']=sample2['datadate'].dt.year

sample2['Month']=sample2['datadate'].dt.month

#set gvkey and datadate as the index

sample2=sample2.set\_index(['gvkey','datadate'])

#split training and testing samples

Train1=sample2[sample2['Year']<2018]

Test1=sample2[sample2['Year']>=2018]

X\_train=Train1[['lagRet2','loglagVOL2','loglagPrice2', 'loglagMV2','lagShareturnover2','lagRet2\_sic',

'lagRet12','loglagVOL12','lagShareturnover12','lagRet12\_std','lagRet12\_min',

'lagRet12\_max','lagRet12\_sic','epspiq','dvpspq','sale','div\_p', 'cash',

'debt','logatq',

'sp500\_ret\_d','nasdaq\_ret\_d','r2000\_ret\_d','dollar\_ret\_d','VIX',

'yield\_3m','yield\_10y','gdp\_growth','Bull\_ave','Bull\_Bear']]

Y\_train=Train1[['ret']]

X\_test=Test1[['lagRet2','loglagVOL2','loglagPrice2', 'loglagMV2','lagShareturnover2','lagRet2\_sic',

'lagRet12','loglagVOL12','lagShareturnover12','lagRet12\_std','lagRet12\_min',

'lagRet12\_max','lagRet12\_sic','epspiq','dvpspq','sale','div\_p', 'cash',

'debt','logatq',

'sp500\_ret\_d','nasdaq\_ret\_d','r2000\_ret\_d','dollar\_ret\_d','VIX',

'yield\_3m','yield\_10y','gdp\_growth','Bull\_ave','Bull\_Bear']]

Y\_test=Test1[['ret']]

Factor = pd.read\_excel(r"C:\Users\rdg83\OneDrive - Rutgers University\Course Investment Portfolio Management\Week 5 Stuff\Factors-1.xlsx")

rf1 = pd.read\_excel(r"C:\Users\rdg83\OneDrive - Rutgers University\Course Investment Portfolio Management\Week 7 Stuff\Treasury bill.xlsx")

rf1['rf']=rf1['DGS3MO']/1200

rf2=rf1[['Date','rf']].dropna()

rf2['Year']=rf2['Date'].dt.year

rf2['Month']=rf2['Date'].dt.month

rf3=rf2[['Year','Month','rf']].groupby(['Year','Month'], as\_index=False).mean()

indexret1=pd.read\_stata(r"C:\Users\rdg83\OneDrive - Rutgers University\Course Investment Portfolio Management\Week 9 Stuff\Index return-1.dta")

from sklearn.experimental import enable\_hist\_gradient\_boosting

from sklearn.ensemble import HistGradientBoostingRegressor

from sklearn.model\_selection import TimeSeriesSplit

#Grid Search

from sklearn.model\_selection import GridSearchCV

tsplit=TimeSeriesSplit(n\_splits=5, test\_size=50000, gap=5000)

#specify the candidate hyperparameter settings/values

param\_candidate = {'max\_iter': [ 50, 100, 200]}

#for max\_iter, we try three numbers: 50, 100, and 200

#define the model

model= HistGradientBoostingRegressor(min\_samples\_leaf=100, early\_stopping='auto')

#define the searching

grid = GridSearchCV(estimator=model, param\_grid=param\_candidate,

cv=tsplit,scoring='neg\_mean\_squared\_error')

grid.fit(X\_train, Y\_train)

#execute the search

grid.cv\_results\_

#show the results for each value of the hyperparameter

grid.best\_params\_

#report the best value of the hyperparameter

#search for multiple hyperparameter

param\_candidate = {'max\_iter': [ 50, 100], 'min\_samples\_leaf':[50,100]}

model= HistGradientBoostingRegressor(early\_stopping=True)

grid = GridSearchCV(estimator=model, param\_grid=param\_candidate,

cv=tsplit,scoring='neg\_mean\_squared\_error')

grid.fit(X\_train, Y\_train)

#execute the search

print (grid.cv\_results\_)

print (grid.best\_params\_)

#report the best value of the parameter

################

"""PROBLEM 3"""

################

from sklearn.ensemble import ExtraTreesRegressor

from sklearn.model\_selection import RandomizedSearchCV

from scipy.stats import randint

# Define the parameter distribution

param\_dist = {"n\_estimators": randint(25, 200),

"min\_samples\_leaf": randint(25, 200),

"max\_features": ['auto','sqrt','log2'],

"max\_depth":[None,10,20,30,40]}

# Create the model

model = ExtraTreesRegressor()

# Create the RandomizedSearchCV object

search = RandomizedSearchCV(model, param\_distributions=param\_dist, n\_iter=10, cv=5)

# Fit the RandomizedSearchCV object

search.fit(X\_train, Y\_train)

print("\nBest parameters:")

print(search.best\_params\_)

"""{'max\_depth': 20, 'max\_features': 'log2', 'min\_samples\_leaf': 78, 'n\_estimators': 43}"""

print("\nBest score:")

print(search.best\_score\_)

""""Best score:-0.017399796346371833"""

################

"""PROBLEM 4"""

################

from sklearn.ensemble import RandomForestRegressor

from sklearn.inspection import permutation\_importance

import matplotlib.pyplot as plt

# Create the model with the best parameters

model = RandomForestRegressor(n\_estimators=43, min\_samples\_leaf=78, max\_depth=20, max\_features='log2')

# Fit the model

model.fit(X\_train, Y\_train)

# Get the feature importances

importances = model.feature\_importances\_

# Run permutation importance

perm\_importance = permutation\_importance(model, X\_test, Y\_test)

# Plot the feature importances

plt.barh(range(len(importances)), importances)

plt.show()

FIM = permutation\_importance(model, X\_train, Y\_train, n\_repeats=5, scoring='neg\_mean\_squared\_error')

FIM\_score\_mean=pd.DataFrame(FIM.importances\_mean, columns=['Feature Importance'])

FIM\_score\_std=pd.DataFrame(FIM.importances\_std, columns=['Feature Importance\_std'])

FIM\_score=pd.merge(FIM\_score\_mean, FIM\_score\_std, left\_index=True,right\_index=True)

FIM\_score['Feature']=X\_test.columns.tolist()

FIM\_score.sort\_values(by=['Feature Importance'],inplace=True)

FIM\_score.plot(kind = "barh",x='Feature', y = 'Feature Importance', title = "Feature Importance",

xerr = 'Feature Importance\_std', fontsize=25, color='red')

FIM\_score['benchmark']=FIM\_score['Feature Importance'].max()

FIM\_score['Feature Importance%']=FIM\_score['Feature Importance']/FIM\_score['benchmark']

FIM\_score.plot(kind = "barh",x='Feature', y = 'Feature Importance%', title = "Feature Importance",

fontsize=20, color='red')

################

"""PROBLEM 5"""

################

from sklearn.ensemble import RandomForestRegressor

from sklearn.inspection import PartialDependenceDisplay

model = RandomForestRegressor(n\_estimators=43, min\_samples\_leaf=78, max\_depth=20, max\_features='log2')

model.fit(X\_train, Y\_train)

PartialDependenceDisplay.from\_estimator(model, features=[['cash']], X=X\_train, grid\_resolution=1200,

line\_kw={"color": "blue",

"linestyle":"dashed",

"linewidth":"3"})

plt.xlabel('cash', fontsize=30)

plt.ylabel('RET', fontsize=30)

plt.title('Partial Dependence Plot', fontsize=30, fontstyle='italic', fontname='Times New Roman')





