from quantopian.algorithm import (

attach\_pipeline,

pipeline\_output,

order\_optimal\_portfolio,

)

import quantopian.algorithm as algo

from quantopian.pipeline import Pipeline

from quantopian.pipeline.data.builtin import USEquityPricing

from quantopian.pipeline.factors import SimpleMovingAverage

from quantopian.pipeline.filters import Q1500US

from quantopian.pipeline.filters import StaticAssets

import quantopian.optimize as opt

from quantopian.pipeline.experimental import risk\_loading\_pipeline

from quantopian.optimize import TargetWeights

import pandas as pd

from sklearn.neighbors import KNeighborsClassifier

from collections import deque

def initialize(context):

# SPY capital, EFA fondos, TIP bonos, GSG indince.

context.security\_list = symbols('SPY', 'EFA','TIP','GSG')

context.classifier = KNeighborsClassifier(n\_neighbors=7, weights='uniform', algorithm='kd\_tree') # Usar un clasificador KNN

#Parametros de restriccion

# Attach de pileine datos

attach\_pipeline(

make\_pipeline(),

'data\_pipe'

)

attach\_pipeline(

risk\_loading\_pipeline(),

'risk\_pipe'

)

# Rebalance 2 horas de abrir el mercado

algo.schedule\_function(

rebalance,

algo.date\_rules.month\_end(),

algo.time\_rules.market\_open(hours=2),

)

# guardar al cierre todos los dias

algo.schedule\_function(

record\_vars,

algo.date\_rules.every\_day(),

algo.time\_rules.market\_close()

)

# crea el conjunto seleccionado

algo.attach\_pipeline(make\_pipeline(), 'pipeline')

def make\_pipeline():

#mencionados anteriormente

base\_universe = StaticAssets(symbols('SPY', 'EFA','TIP','GSG'))

# precio de cierre del dia anterior

yesterday\_close = USEquityPricing.close.latest

sma\_25 = SimpleMovingAverage(inputs=[USEquityPricing.close], window\_length=25) # Variables independientes

sma\_200 = SimpleMovingAverage(inputs=[USEquityPricing.close], window\_length=200) # Variables dependientes

pipe = Pipeline(

screen=base\_universe & (sma\_25 > sma\_200),

columns={

'price': yesterday\_close,

'sma\_200': sma\_200,

}

)

return pipe

def before\_trading\_start(context, data):

context.output = algo.pipeline\_output('pipeline')

#Lo que se va a negociar

#context.security\_list = context.output.index

log.info(context.security\_list)

context.risk\_factor\_betas = pipeline\_output('risk\_pipe')

#securities a comerciar

context.securities\_for\_month = context.output.index

context.weights = pd.Series([.25, .25, .25, .25],

index = context.security\_list)

context.prediction = 0 # Se guarda la predicción más reciente

def rebalance(context, data):

#se ejecuta de acuerdo a schedule\_fuction

log.info(context.output.head(10))

context.recent\_prices.append(data.current(context.security, 'price')) # Agrega el precio actual a la cola

weights = {}

#context.security\_list = symbols('SPY', 'EFA','TIP','GSG')

for equity in context.security\_list:

#comprueba si el capital del pipeline para el mes y asignar peso

if(context.securities\_for\_month.any(equity)):

#weights[equity] = float(context.weights[equity])

weights[equity] = .20

else:

# Trade if there is more than a 1% decrease (SHORT)

weights[equity] = -1 \* context.weights[equity]

log.info("context weights %s" %context.weights)

# optimizate api contest

objective = opt.TargetWeights(weights)

order\_optimal\_portfolio(objective, [opt.MaxGrossExposure(1)])

if len(context.recent\_prices) == context.window\_length+2: # Garantiza que hayan datos suficientes para generar un buen modelo

# opt.order\_optimal\_portfolio(objective=TargetWeights(context.weights),constraints=[])

# Hace una lista de booleanos para saber si los precios han cambiado respecto a los anteriores

changes = np.diff(context.recent\_prices) > 0

sma\_25.append(changes[:-1]) # Variables independientes

sma\_200.append(changes[-1]) # Variables dependientes

if len(sma\_200) >= 100: # Garantiza que hayan datos suficientes para generar un buen modelo

context.classifier.fit(context.X, context.Y) # Generar el modelo

context.prediction = context.classifier.predict(changes[1:]) # Predicción

# Si la predicción es True, se compran todas las acciones posibles, si es False, se venden todas las acciones posibles del portafolio

order\_target\_percent(context.security, context.prediction)

# tiene el tiempo de cambio actual

exchange\_time = get\_datetime('US/Eastern')

def record\_vars(context, data):

record(prediction=int(context.prediction))