Basic Pairs Trading Algorithm

這邊介紹了一個簡單pairs trading algorithm。

首先，我們先載入需要使用的套件庫

```
import numpy as np
import pandas as pd
import quantopian.optimize as opt
import quantopian.algorithm as algo

MAX_GROSS_EXPOSURE = 1.0 # Set exposure constraint constant value for optimizer
```

然後我們看看initialize部份的設定

```
def initialize(context):
    """
    Called once at the start of the algorithm.
    """
    schedule_function(check_pair_status, date_rules.every_day(),
time_rules.market_close(minutes=60))

    context.stock1 = symbol('ABGB')
    context.stock2 = symbol('FSLR')
    context.stocks = [context.stock1, context.stock2]

    # Our threshold for trading on the z-score
    context.entry_threshold = 0.2
    context.exit_threshold = 0.1

    # Create a variable to store our target weights
    context.target_weights = pd.Series(index=context.stocks, data=0.0)

    # Moving average lengths
    context.long_ma_length = 30
    context.short_ma_length = 1

    # Flags to tell us if we're currently in a trade
    context.currently_long_the_spread = False
    context.currently_short_the_spread = False
```

我們知道initialize部份是用來定義一次性邏輯，在這邊我們先設定schedule function，在每個交易日的休市的前一個小時執行check_pair_status這個函數，跟筆記本我們做得事情一樣，我們把ABGB跟FSLR的symbol分別存為stock1和stock2並如同我們之前所說的，存在context之下，後面也分別存了z score的閥值、移動平均長度還有目前交易的狀態。同時，這邊我們最佳化我們資產配置的方法不再是maxalpha，我們會採取TargetWeights方法，我們先看看文檔

```

```

```
class TargetWeights(weights)

    Objective that minimizes the distance from an already-computed portfolio.
    Parameters: weights (pd.Series[Asset -> float] or dict[Asset -> float]) - Map from
asset to target percentage of holdings.

    Notes

    A target value of 1.0 indicates that 100% of the portfolio's current net liquidation
value should be held in a long position in the corresponding asset.

    A target value of -1.0 indicates that -100% of the portfolio's current net
liquidation value should be held in a short position in the corresponding asset.

    Assets with target values of exactly 0.0 are ignored unless an algorithm has an
existing position in the given asset.

    If an algorithm has an existing position in an asset and no target weight is
provided, the target weight is assumed to be zero.
```

這個函式會最小化目前資產配置與目標資產配置的距離，意思就是，使用這個目標函數，我們會先設定我們目標配置權重，這個函式會幫我們進行下單，讓我們的資產配越接近這個目標配置越近越好。在這邊，我們先設定目標權重為0.0。

然後我們看看check_pair_status這個函數

```
def check_pair_status(context, data):

    # For notational convenience
    s1 = context.stock1
    s2 = context.stock2

    # Get pricing history
    prices = data.history([s1, s2], "price", context.long_ma_length, '1d')

    # Try debugging me here to see what the price
    # data structure looks like
    # To debug, click on the line number to the left of the
    # next command. Line numbers on blank lines or comments
    # won't work.
    short_prices = prices.iloc[-context.short_ma_length:]

    # Get the long mavg
    long_ma = np.mean(prices[s1] - prices[s2])
    # Get the std of the long window
    long_std = np.std(prices[s1] - prices[s2])


    # Get the short mavg
    short_ma = np.mean(short_prices[s1] - short_prices[s2])

    # Compute z-score
```

```
    if long_std > 0:
        zscore = (short_ma - long_ma)/long_std

        # Our two entry cases
        if zscore > context.entry_threshold and \
            not context.currently_short_the_spread:
            context.target_weights[s1] = -0.5 # short top
            context.target_weights[s2] = 0.5 # long bottom
            context.currently_short_the_spread = True
            context.currently_long_the_spread = False

        elif zscore < -context.entry_threshold and \
            not context.currently_long_the_spread:
            context.target_weights[s1] = 0.5 # long top
            context.target_weights[s2] = -0.5 # short bottom
            context.currently_short_the_spread = False
            context.currently_long_the_spread = True

        # Our exit case
        elif abs(zscore) < context.exit_threshold:
            context.target_weights[s1] = 0 # close out
            context.target_weights[s2] = 0 # close out
            context.currently_short_the_spread = False
            context.currently_long_the_spread = False
        record('zscore', zscore)

    # Call the optimizer
    allocate(context, data)
```

首先，程式會先抓取我們鎖定的兩隻股票的歷史資料，接下來計算出對應z score，然後在z-score大於閥值時，做空top 作多 bottom;在z score 小於負的閥值時，作多top ，做空bottom，而在z score在閥值內時出場，在這邊下單的邏輯被寫在 allocate這個函數裡。

讓我們最後看看allocate這個函數

```
def allocate(context, data):
    # Set objective to match target weights as closely as possible, given constraints
    objective = opt.TargetWeights(context.target_weights)

    # Define constraints
    constraints = []
    constraints.append(opt.MaxGrossExposure(MAX_GROSS_EXPOSURE))

    algo.order_optimal_portfolio(
        objective=objective,
        constraints=constraints,
    )
```

可以看到我們的objective function為opt.TargetWeights，我們的constraints為MaxGrossExposure。

```
"""
This is a basic pairs trading algorithm that uses the Optimize API.
```

```
WARNING: THIS IS A LEARNING EXAMPLE ONLY. DO NOT TRY TO TRADE SOMETHING THIS SIMPLE.
https://www.quantopian.com/workshops
https://www.quantopian.com/lectures

For any questions, email max@quantopian.com
"""
import numpy as np
import pandas as pd
import quantopian.optimize as opt
import quantopian.algorithm as algo

MAX_GROSS_EXPOSURE = 1.0 # Set exposure constraint constant value for optimizer

def initialize(context):
    """
    Called once at the start of the algorithm.
    """
    schedule_function(check_pair_status, date_rules.every_day(),
time_rules.market_close(minutes=60))

    context.stock1 = symbol('ABGB')
    context.stock2 = symbol('FSLR')
    context.stocks = [context.stock1, context.stock2]

    # Our threshold for trading on the z-score
    context.entry_threshold = 0.2
    context.exit_threshold = 0.1

    # Create a variable to store our target weights
    context.target_weights = pd.Series(index=context.stocks, data=0.0)

    # Moving average lengths
    context.long_ma_length = 30
    context.short_ma_length = 1

    # Flags to tell us if we're currently in a trade
    context.currently_long_the_spread = False
    context.currently_short_the_spread = False


def check_pair_status(context, data):

    # For notational convenience
    s1 = context.stock1
    s2 = context.stock2

    # Get pricing history
    prices = data.history([s1, s2], "price", context.long_ma_length, '1d')

    # Try debugging me here to see what the price
    # data structure looks like
    # To debug, click on the line number to the left of the
    # next command. Line numbers on blank lines or comments
```

```
    # won't work.
    short_prices = prices.iloc[-context.short_ma_length:]

    # Get the long mavg
    long_ma = np.mean(prices[s1] - prices[s2])
    # Get the std of the long window
    long_std = np.std(prices[s1] - prices[s2])


    # Get the short mavg
    short_ma = np.mean(short_prices[s1] - short_prices[s2])

    # Compute z-score
    if long_std > 0:
        zscore = (short_ma - long_ma)/long_std

        # Our two entry cases
        if zscore > context.entry_threshold and \
            not context.currently_short_the_spread:
            context.target_weights[s1] = -0.5 # short top
            context.target_weights[s2] = 0.5 # long bottom
            context.currently_short_the_spread = True
            context.currently_long_the_spread = False

        elif zscore < -context.entry_threshold and \
            not context.currently_long_the_spread:
            context.target_weights[s1] = 0.5 # long top
            context.target_weights[s2] = -0.5 # short bottom
            context.currently_short_the_spread = False
            context.currently_long_the_spread = True

        # Our exit case
        elif abs(zscore) < context.exit_threshold:
            context.target_weights[s1] = 0 # close out
            context.target_weights[s2] = 0 # close out
            context.currently_short_the_spread = False
            context.currently_long_the_spread = False
        record('zscore', zscore)

    # Call the optimizer
    allocate(context, data)


def allocate(context, data):
    # Set objective to match target weights as closely as possible, given constraints
    objective = opt.TargetWeights(context.target_weights)

    # Define constraints
    constraints = []
    constraints.append(opt.MaxGrossExposure(MAX_GROSS_EXPOSURE))

    algo.order_optimal_portfolio(
        objective=objective,
```

```
        constraints=constraints,
    )
```