Quantopian API - Pipeline 筆記 Part 1/3

本筆記跟隨者下列的Quantopian官方教程

Pipeline官方教程

Section 1 Introduction

我們在Getting Started教程已經對pipeline api有基本的認識了，接下來我們要對如何建立pipeline有更深入的理解。

為什麼使用pipeline?

許多交易算法具有以下的結構:

- 在已知的資產集合裡面(通常很大)計算某些因子值
- 根據算出的因子值決定交易的資產集合
- 根據選定的交易資產計算所需的投資組合權重
- 下單將目前的資產組合轉到上述的投資組合

要做到這幾件事，我們會有以下的技術挑戰:

- 有效率的查詢大量的資產
- 對大量的資產進行計算
- 資產調整(股利等等)
- 資產除名

而pipeline提供了統一的api來解決這些問題，而一個重要的優點便是，在理想的工作流程為:我們在Research環境中發現想法，並在IDE中實現算法，而pipeline提供了統一的的建構方法，方便我們在兩者之間做切換。

pipeline中的計算

我們可以在pipeline中表達三種不同類型的計算:Factors, Filters and Classifiers

Factors

A factor is a function from an asset and a moment in time to a **numerical value**.

簡單來說，因子是一個實數函數，這個函數的input為資產與現在的時間。譬如說資產的最新價格便是一個因子的例子，或是資產的10天平均交易量。我們通常利用因子來計算alpha值等等。

Filters

A filter is a function from an asset and a moment in time to a **boolean**.

簡單來說，過濾器的本質為一個布林值函數(output為 True or False)，這個函數的input為資產與現在的時間。譬如判斷這隻資產是否低於十美元，我們通常用來描述或是排除某些特定用途的資產集。

Classifiers

A classifier is a function from an asset and a moment in time to a **categorical output**.

簡單來說，分類器是一個整數或是字串函數，將輸入input為資產以及時間，譬如說分類資產所屬的交易所等等，分類器常用於對因子輸出的複雜變換進行資產分組。

Section 2 Creating a Pipeline

這部份在Getting Started我們已經熟悉了基本的Pipeline的建構過程，這邊提供了簡單的例子。

```
from quantopian.pipeline import Pipeline
from quantopian.research import run_pipeline


def make_pipeline():
    return Pipeline()
my_pipe = make_pipeline()
result = run_pipeline(my_pipe, '2015-05-05', '2015-05-05')
```

result將為一個dataframe，沒有任何columns，但是每天的index都有超過8000個資產名。

Section 3 Factors

我們先前已經知道因子是資產與時間打到實數的函數，我們先學習一些內置的因子函數。以下便是使用內建的移動平均函數建構的例子:

```
from quantopian.pipeline.data.builtin import USEquityPricing
from quantopian.pipeline.factors import SimpleMovingAverage

# Creating a Factor
mean_close_10 = SimpleMovingAverage(
    inputs=[USEquityPricing.close],
    window_length=10
)
# Adding a Factor to a Pipeline
def make_pipeline():

    mean_close_10 = SimpleMovingAverage(
        inputs=[USEquityPricing.close],
        window_length=10
    )

    return Pipeline(
        columns={
            '10_day_mean_close': mean_close_10
        }
    )
result = run_pipeline(make_pipeline(), '2015-05-05', '2015-05-05')
result.head()
```

另外一個例子是Latests

```
def make_pipeline():
```

```
    mean_close_10 = SimpleMovingAverage(
        inputs=[USEquityPricing.close],
        window_length=10
    )
    latest_close = USEquityPricing.close.latest

    return Pipeline(
        columns={
            '10_day_mean_close': mean_close_10,
            'latest_close_price': latest_close
        }
    )
```

Section 4 Combining Factors

通過任何內置數學運算符（+，-，*等），可以將因子與其他因子和標量值組合在一起。 這使得編寫組合多個因子的複雜表達式變得容易。 例如，構建一個計算其他兩個因子平均值的因子就是：

```
f1 = SomeFactor(...)
f2 = SomeOtherFactor(...)
average = (f1 + f2) / 2.0
```

在本課程中，我們將創建一個管道，通過將10天平均價格因素與30天平均價格因素相結合來創建percent_difference因子。 讓我們從製作這兩個因子開始吧!

```
def make_pipeline():

    mean_close_10 = SimpleMovingAverage(
        inputs=[USEquityPricing.close],
        window_length=10
    )
    mean_close_30 = SimpleMovingAverage(
        inputs=[USEquityPricing.close],
        window_length=30
    )

    percent_difference = (mean_close_10 - mean_close_30) / mean_close_30

    return Pipeline(
        columns={
            'percent_difference': percent_difference
        }
    )
result = run_pipeline(make_pipeline(), '2015-05-05', '2015-05-05')
result.head()
```