Quantopian Getting Started 筆記 (Part2/2)

在上次教程中，我們介紹了notebook環境，並藉由繪製圖表的過程中開發了我們的第一個策略，接下來，我們要嘗試將我們開發出來的策略放到algorithm環境中進行回測。

Section1: Algorithm API介紹

我們首先先簡單介紹algorithm環境所需要的一些知識，在這邊，algorithm裡面提供的函數可以處理包括訂單調度以及執行等功能，我們在這邊介紹一些核心函數。

initialize(context)

這是一個初始化函數，以context作為一個input，所有參數的初始化與一次性邏輯都寫在這。這裡另一個重點為context為一python 字典，是用儲存整個過程的狀態，所有的變數應該存入context裡面，而非在宣告一個全局變數，存變數的方法(contex.some_attribute)

before_trading_start(context, data)

這個函數會在每天交易開始前被呼叫，使用context(參數)與data(每天資料)作為input。這個函數可作為資料的前處理。

schedule_function(func, day_rule, time_rule)

Quantopian預設在9:30AM-4PM Eastern Time 作交易，schedule_function可以指定時間執行特定的函數。

一個簡單的架構

```
# Import Algorithm API
import quantopian.algorithm as algo


def initialize(context):
    # Initialize algorithm parameters
    context.day_count = 0
    context.daily_message = "Day {}."
    context.weekly_message = "Time to place some trades!"

    # Schedule rebalance function
    algo.schedule_function(
        rebalance,
        date_rule=algo.date_rules.week_start(),
        time_rule=algo.time_rules.market_open()
    )


def before_trading_start(context, data):
    # Execute any daily actions that need to happen
    # before the start of a trading session
```

```
    context.day_count += 1
    log.info(context.daily_message, context.day_count)


def rebalance(context, data):
    # Execute rebalance logic
    log.info(context.weekly_message)
```

Section 2 Data Processing in Algorithms

這邊，我們將要學習如何將在research版面研究好的算法部屬到algorithm環境之中，為了將我們的資料管線放到algorithm環境之中，這邊我們需要用到attch_pipeline方法，如以下的簡單例子。

```
# Import Algorithm API
import quantopian.algorithm as algo

# Pipeline imports
from quantopian.pipeline import Pipeline
from quantopian.pipeline.data.psychsignal import stocktwits
from quantopian.pipeline.factors import SimpleMovingAverage
from quantopian.pipeline.filters import QTradableStocksUS


def initialize(context):
    # Attach pipeline to algorithm
    algo.attach_pipeline(
        make_pipeline(),
        'data_pipe'
    )

    # Schedule rebalance function
    algo.schedule_function(
        rebalance,
        date_rule=algo.date_rules.week_start(),
        time_rule=algo.time_rules.market_open()
    )


def before_trading_start(context, data):
    # Get pipeline output and
    # store it in context
    context.pipeline_data = algo.pipeline_output('data_pipe')


def rebalance(context, data):
    # Display first 10 rows
    # of pipeline output
    log.info(context.pipeline_data.head(10))


# Pipeline definition
def make_pipeline():
```

```
    base_universe = QTradableStocksUS()

    sentiment_score = SimpleMovingAverage(
        inputs=[stocktwits.bull_minus_bear],
        window_length=3,
    )

    return Pipeline(
        columns={
            'sentiment_score': sentiment_score,
        },
        screen=(
            base_universe
            & sentiment_score.notnull()
        )
    )
```

重點一：我們可以看到在initialize的時候，我們把我的管道函數配置在演算法中(使用attach_pipeline)，並配置一個schedule函數，會在每週開盤時打印出我們管道資料的前10行，且在交易開始前，管道資料存到context之中。

重點二： 在make_pipeline的部份，可以看到與我們在research部份所作的基本一致，但在這裡，我們並沒有限制資產數量，我們考慮了所有的資產，只要其sentiment_score不為0

Section 3 Portfolio Management

我們現在已經成功將管道資料引進每一天的資料中，接下來我們就要利用資料進行資產配置。首先，我們需要從這些資料計算出每支資產的分數(alpha值)，然後我們將會利用quantopian所提供的工具去最佳化，請看下面的sample code

```
# Import Optimize API module
import quantopian.optimize as opt


def rebalance(context, data):
  # Retrieve alpha from pipeline output
  alpha = context.pipeline_data.sentiment_score

  if not alpha.empty:
      # Create MaximizeAlpha objective
      objective = opt.MaximizeAlpha(alpha)
```

在這邊，我們定義sentiment_score為alpha值，在alpha值不是空值的情況下，我們的目標函數為最大化alpha值，而這個最佳化工具MaximizeAlpha將會幫助我們由alpha值來配置我們的資產。接下來，我們必須給定配置資產的限制式，我們將限制條件存在context變數之中

```
# Constraint parameters
context.max_leverage = 1.0
context.max_pos_size = 0.015
context.max_turnover = 0.95
```

則我們完整的程式碼為

```
# Import Algorithm API
import quantopian.algorithm as algo

# Import Optimize API
import quantopian.optimize as opt


def rebalance(context, data):
  # Retrieve alpha from pipeline output
  alpha = context.pipeline_data.sentiment_score

  if not alpha.empty:
      # Create MaximizeAlpha objective
      objective = opt.MaximizeAlpha(alpha)

      # Create position size constraint
      constrain_pos_size = opt.PositionConcentration.with_equal_bounds(
          -context.max_pos_size,
          context.max_pos_size
      )

      # Constrain target portfolio's leverage
      max_leverage = opt.MaxGrossExposure(context.max_leverage)

      # Ensure long and short books
      # are roughly the same size
      dollar_neutral = opt.DollarNeutral()

      # Constrain portfolio turnover
      max_turnover = opt.MaxTurnover(context.max_turnover)

      # Rebalance portfolio using objective
      # and list of constraints
      algo.order_optimal_portfolio(
          objective=objective,
          constraints=[
              constrain_pos_size,
              max_leverage,
              dollar_neutral,
              max_turnover,
          ]
      )
```

可以看到，我們在rebalance部份寫下來配置資產的邏輯，在最後一段，我們調用了order_optimal_porfolio函數來配置資產，其目標為最大化alpha值，並在上述四個限制下。

Risk Management

除了設置資產配置的限制式之外，我們還希望限制我們的風險暴露，舉例來說，由於情緒數據可能只會短暫維持，和我們利用情緒評分高峰的意圖，我們的算法可能會面臨短期逆轉風險。

在這邊，我們將使用Quantopian的風險模型來管理我們的投資組合對常見風險因素的風險。 風險模型計算了16種不同風險因素的資產敞口：11個部門因素和5個風格因素（包括短期逆轉）。 我們可以使用risk_loading_pipeline函數在我們的算法中輕鬆訪問這些數據，該函數返回一個數據管道，該管道為風險模型中的每個因子生成一列輸出。

```
# Import Algorithm API
import quantopian.algorithm as algo

# Import Risk API method
from quantopian.pipeline.experimental import risk_loading_pipeline

def initialize(context):
    # Constraint parameters
    context.max_leverage = 1.0
    context.max_pos_size = 0.015
    context.max_turnover = 0.95

    # Attach data pipelines
    algo.attach_pipeline(
        make_pipeline(),
        'data_pipe'
    )
    algo.attach_pipeline(
        risk_loading_pipeline(),
        'risk_pipe'
    )

    # Schedule rebalance function
    algo.schedule_function(
        rebalance,
        algo.date_rules.week_start(),
        algo.time_rules.market_open(),
    )


def before_trading_start(context, data):
    # Get pipeline outputs and
    # store them in context
    context.pipeline_data = algo.pipeline_output(
      'data_pipe'
    )

    context.risk_factor_betas = algo.pipeline_output(
      'risk_pipe'
    )
```

在這邊，我們加入風險管道，並將beta值存下來，而我們的下一步便是加入RiskModelExposure進入我們的策略執行中

```
# Constrain target portfolio's risk exposure
# By default, max sector exposure is set at
# 0.2, and max style exposure is set at 0.4
factor_risk_constraints = opt.experimental.RiskModelExposure(
    context.risk_factor_betas,
    version=opt.Newest
)
```

以下就是我們的完整程式碼

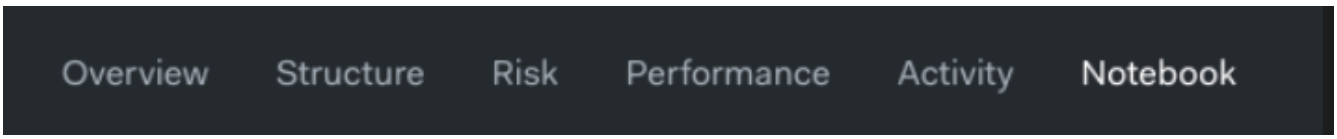
```
# Import Algorithm API
import quantopian.algorithm as algo

# Import Optimize API
import quantopian.optimize as opt

# Pipeline imports
from quantopian.pipeline import Pipeline
from quantopian.pipeline.data.psychsignal import stocktwits
from quantopian.pipeline.factors import SimpleMovingAverage

# Import built-in universe and Risk API method
from quantopian.pipeline.filters import QTradableStocksUS
from quantopian.pipeline.experimental import risk_loading_pipeline


def initialize(context):
    # Constraint parameters
    context.max_leverage = 1.0
    context.max_pos_size = 0.015
    context.max_turnover = 0.95

    # Attach data pipelines
    algo.attach_pipeline(
        make_pipeline(),
        'data_pipe'
    )
    algo.attach_pipeline(
        risk_loading_pipeline(),
        'risk_pipe'
    )

    # Schedule rebalance function
    algo.schedule_function(
        rebalance,
        algo.date_rules.week_start(),
        algo.time_rules.market_open(),
    )


def before_trading_start(context, data):
    # Get pipeline outputs and
```

```
    # store them in context
    context.pipeline_data = algo.pipeline_output('data_pipe')

    context.risk_factor_betas = algo.pipeline_output('risk_pipe')


# Pipeline definition
def make_pipeline():

    sentiment_score = SimpleMovingAverage(
        inputs=[stocktwits.bull_minus_bear],
        window_length=3,
        mask=QTradableStocksUS()
    )

    return Pipeline(
        columns={
            'sentiment_score': sentiment_score,
        },
        screen=sentiment_score.notnull()
    )


def rebalance(context, data):
    # Retrieve alpha from pipeline output
    alpha = context.pipeline_data.sentiment_score

    if not alpha.empty:
        # Create MaximizeAlpha objective
        objective = opt.MaximizeAlpha(alpha)

        # Create position size constraint
        constrain_pos_size = opt.PositionConcentration.with_equal_bounds(
            -context.max_pos_size,
            context.max_pos_size
        )

        # Constrain target portfolio's leverage
        max_leverage = opt.MaxGrossExposure(context.max_leverage)

        # Ensure long and short books
        # are roughly the same size
        dollar_neutral = opt.DollarNeutral()

        # Constrain portfolio turnover
        max_turnover = opt.MaxTurnover(context.max_turnover)

        # Constrain target portfolio's risk exposure
        # By default, max sector exposure is set at
        # 0.2, and max style exposure is set at 0.4
        factor_risk_constraints = opt.experimental.RiskModelExposure(
            context.risk_factor_betas,
            version=opt.Newest
```

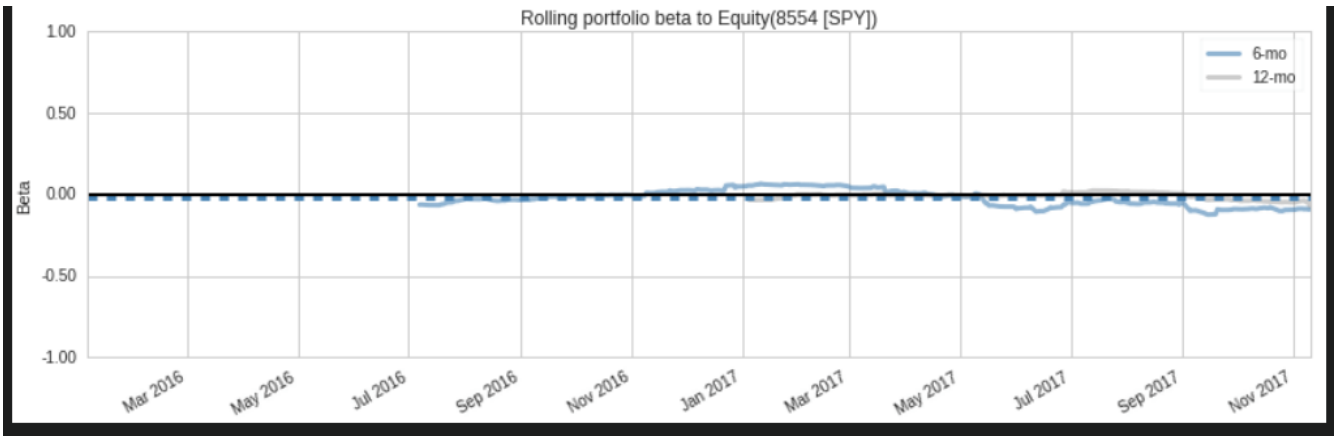
```
    )

    # Rebalance portfolio using objective
    # and list of constraints
    algo.order_optimal_portfolio(
        objective=objective,
        constraints=[
            constrain_pos_size,
            max_leverage,
            dollar_neutral,
            max_turnover,
            factor_risk_constraints,
        ]
    )
```

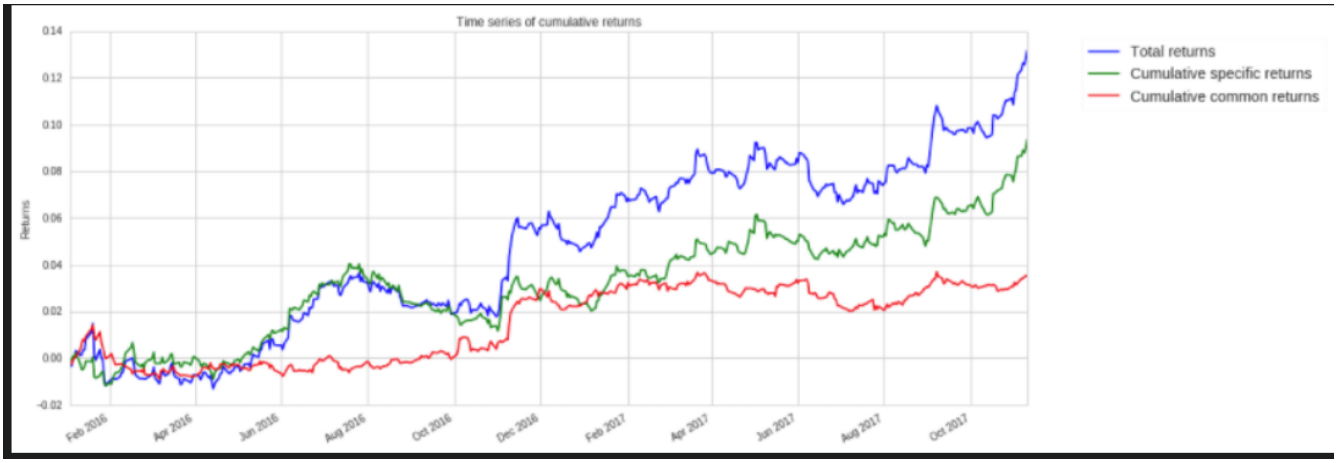
Section 4 Backtest Analysis

當我們完成full backtest後，我們可以點擊notebook按鈕



此時可以帶我們回到research環境，執行此單元格（Shift + Enter）會將backtest生成的數據加載到研究筆記本中，並使用它創建Pyfolio頁面。 Pyfolio是Quantopian的投資組合和風險分析的開源工具。 它提供了許多可視化工具，旨在幫助您更好地了解算法的行為和風險暴露。 例如，下圖顯示了我們的投資組合隨著時間的推移滾動市場。 我們想要建立一個多空股票交易算法的原因之一是保持與市場的低相關性，因此我們希望這個圖在整個回溯測試期間始終為0左右。




Time series of cumulative returns
Returns
0.14
0.12
0.10
0.08
0.06
0.04
0.02
0.00
-0.02
Feb 2016
Apr 2016
Jun 2016
Aug 2016
Oct 2016
Dec 2016
Feb 2017
Apr 2017
Jun 2017
Aug 2017
Oct 2017
Total returns
Cumulative specific returns
Cumulative common returns