Getting Started 筆記

我們先複習一下上禮拜的程式碼。

```
# 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

# Import Optimize API
import quantopian.optimize as opt

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


    # Attch 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):
    # Retrieve alpha from pipeline output
    alpha = context.pipeline_data.semtiment_score

    if not alpha.empty:
        # Create MaximizeAlpha objective
        objective = opt.MaximizeAlpha(alpha)
        # Create position size contraint
        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 are roughly the same size
        dollar_neutral = opt.DollarNeutral()

        #Constraint 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,
                ]
            )

def make_pipeline():
    base_universe = QTradableStocksUS()

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

    return Pipeline(
        columns={
            'semtiment_score':semtiment_score,

            },
        screen=(
            base_universe & semtiment_score.notnull()
            )
        )
```

首先第一個部份

```
# 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

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

這邊匯入需要用到的套件，其中包括algorithm部份主要的套件quantopian.algorithm，然後與pipeline相關的套件(與在Research部份一致)，然後用來執行下單邏輯的quantopian.optimize。

再來是initialize部份

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


    # Attch 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()
        )
```

如同我們上次所說的，這邊如果我們之前所說的，initialize部份是來處理參數的初始化與一次性邏輯，所以我們先定義了三個變數，並根據規範存在context這個字典裡，然後我們加入data的pipeline以及設定schedule function.

接下來，我們來看一下我們的管道函數

```
def make_pipeline():
    base_universe = QTradableStocksUS()

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

    return Pipeline(
        columns={
            'semtiment_score':semtiment_score,

            },
        screen=(
            base_universe & semtiment_score.notnull()
            )
        )
```

與在research所作的基本一致。

然後我們看一下剩餘的部份

```
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):
    # Retrieve alpha from pipeline output
    alpha = context.pipeline_data.semtiment_score

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

        # Create position size contraint
        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 are roughly the same size
        dollar_neutral = opt.DollarNeutral()

        #Constraint 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,
                ]
            )
```

首先，在每個交易日之前，我們都會把pipeline抓出來的資料存到context下面，取名叫pipeline_data，而我們在定義了reblance函數，用來處理交易邏輯。

在reblance函數可以看到，我們先將抓到的三天移動平均情緒指標存為alpha變數。如果alpha不是空值的話，我們會用opt的MaximizeAlpha方法來帶入alpha進行計算，而這個函數是在做什麼勒，我們看一下文檔

```
class MaximizeAlpha(alphas)
Objective that maximizes weights.dot(alphas) for an alpha vector.

Ideally, alphas should contain coefficients such that alphas[asset] is proportional to
the expected return of asset for the time horizon over which the target portfolio will be
held.

In the special case that alphas is an estimate of expected returns for each asset, this
objective simply maximizes the expected return of the total portfolio.

Parameters: alphas (pd.Series[Asset -> float] or dict[Asset -> float]) – Map from assets
to alpha coefficients for those assets.
Notes

This objective should almost always be used with a MaxGrossExposure constraint, and
should usually be used with a PositionConcentration constraint.

Without a constraint on gross exposure, this objective will raise an error attempting to
allocate an unbounded amount of capital to every asset with a nonzero alpha.

Without a constraint on individual position size, this objective will allocate all of its
capital in the single asset with the largest expected return.
```

這邊告訴我們alpha基本上是可以預測期望報酬的指標，我們給定alpha後，演算法會選擇最大化alpha的資產配置，因為要解一個最佳化問題，所以我們必須加入至少兩個constraints來得到合理的結果，第一個限制式為constraint為gross exposure

```
class MaxGrossExposure(max)
Constraint on the maximum gross exposure for the portfolio.

Requires that the sum of the absolute values of the portfolio weights be less than max.

Parameters: max (float) – The maximum gross exposure of the portfolio.
Examples

MaxGrossExposure(1.5) constrains the total value of the portfolios longs and shorts to be
no more than 1.5x the current portfolio value.
```

這個限制式限制了最大的gross exposure，舉例來說MaxGrossExposure(1.5)代表作多或是最多不能超過目前的資產配置價值的1.5倍，因此如果沒有加入這個限制式的話，MaximizeAlpha回報錯，因為演算法會想要將無限的資產配置的alpha為正的資產上。

而另一個需要加上去的限制式為

```
class PositionConcentration(min_weights, max_weights, default_min_weight=0.0,
default_max_weight=0.0, etf_lookthru=None)
Constraint enforcing minimum/maximum position weights.
```

```
Parameters:
min_weights (pd.Series[Asset -> float] or dict[Asset -> float]) – Map from asset to
minimum position weight for that asset.
max_weights (pd.Series[Asset -> float] or dict[Asset -> float]) – Map from asset to
maximum position weight for that asset.
default_min_weight (float, optional) – Value to use as a lower bound for assets not found
in min_weights. Default is 0.0.
default_max_weight (float, optional) – Value to use as a lower bound for assets not found
in max_weights. Default is 0.0.
etf_lookthru (pd.DataFrame, optional) –
Indexed by constituent assets x ETFs, expresses the weight of each constituent in each
ETF.

A DataFrame containing ETF constituents data. Each column of the frame should contain
weights (from 0.0 to 1.0) representing the holdings for an ETF. Each row should contain
weights for a single stock in each ETF. Columns should sum approximately to 1.0. If
supplied, ETF holdings in the current and target portfolio will be decomposed into their
constituents before constraints are applied.

Notes

Negative weight values are interpreted as bounds on the magnitude of short positions. A
minimum weight of 0.0 constrains an asset to be long-only. A maximum weight of 0.0
constrains an asset to be short-only.

A common special case is to create a PositionConcentration constraint that applies a
shared lower/upper bound to all assets. An alternate constructor,
PositionConcentration.with_equal_bounds(), provides a simpler API supporting this use-
case.
```

這個限制式限制的每個資產最大配置大小，如果沒有配置這個限制式的話，演算法將會配置所有的資本到有最大alpha的資產上面。

所以我們的例子基本上就用maxalpha進行下單其中包含maxgross、position、貨幣中立以及turnover條件，其中turnover意思是absolute value of trades in a day / portfolio value。