Long Short Equity Strategy 筆記

這邊是針對

Lecture 37

Lecture 38

的筆記

長空策略是一個有長期歷史的策略，這個策略的思想十分簡單，我們利用模型去對市場所有資產進行排名，然後我們買入排名前n的資產，並賣出底部後n的資產，同時維持多頭與空頭之間的相等美元交易量。

Long and Short Baskets

假設我們排了m個資產，且我們有 d元去進行投資，我們打算交易$2n$資產，則我們交易的規則如下。排名1到n的資產我們用$\frac{d}{2n}$的金錢買入;對於排名$m-n,\cdots,m$的資產我們也用$\frac{d}{2n}$的錢買入。

Friction Because of Prices

Because equity prices will not always divide

因為我們的資產價格可能沒辦法被整除，所以在實施這個策略時，其實做多與做空部位之間的錢其實有誤差。另一方面，如果我們擁有的部位並不大，假設$d=100000$ ，且 $n=500$ 則一個資產只能被分到100，對於昂貴的資產會有麻煩。

Returns Come From The Ranking Spread

接下來我們來模擬一個虛擬資產來說明這個策略是如何創造利潤。

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

# We'll generate a random factor
current_factor_values = np.random.normal(0, 1, 10000)
equity_names = ['Equity ' + str(x) for x in range(10000)]
# Put it into a dataframe
factor_data = pd.Series(current_factor_values, index = equity_names)
factor_data = pd.DataFrame(factor_data, columns=['Factor Value'])
# Take a look at the dataframe
factor_data.head(10)
```

這邊我們隨機產生的10000個資產的因子數值，接下來每個資產的回報率為此因子值加上一個常態擾動項，這也代表我們掌握一個非常好的因子，可以對所有的資產的回報率有好的解釋能力。

```
# Now let's say our future returns are dependent on our factor values
future_returns = current_factor_values + np.random.normal(0, 1, 10000)

returns_data = pd.Series(future_returns, index=equity_names)
returns_data = pd.DataFrame(returns_data, columns=['Returns'])
# Put both the factor values and returns into one dataframe
data = returns_data.join(factor_data)
# Take a look
data.head(10)
```

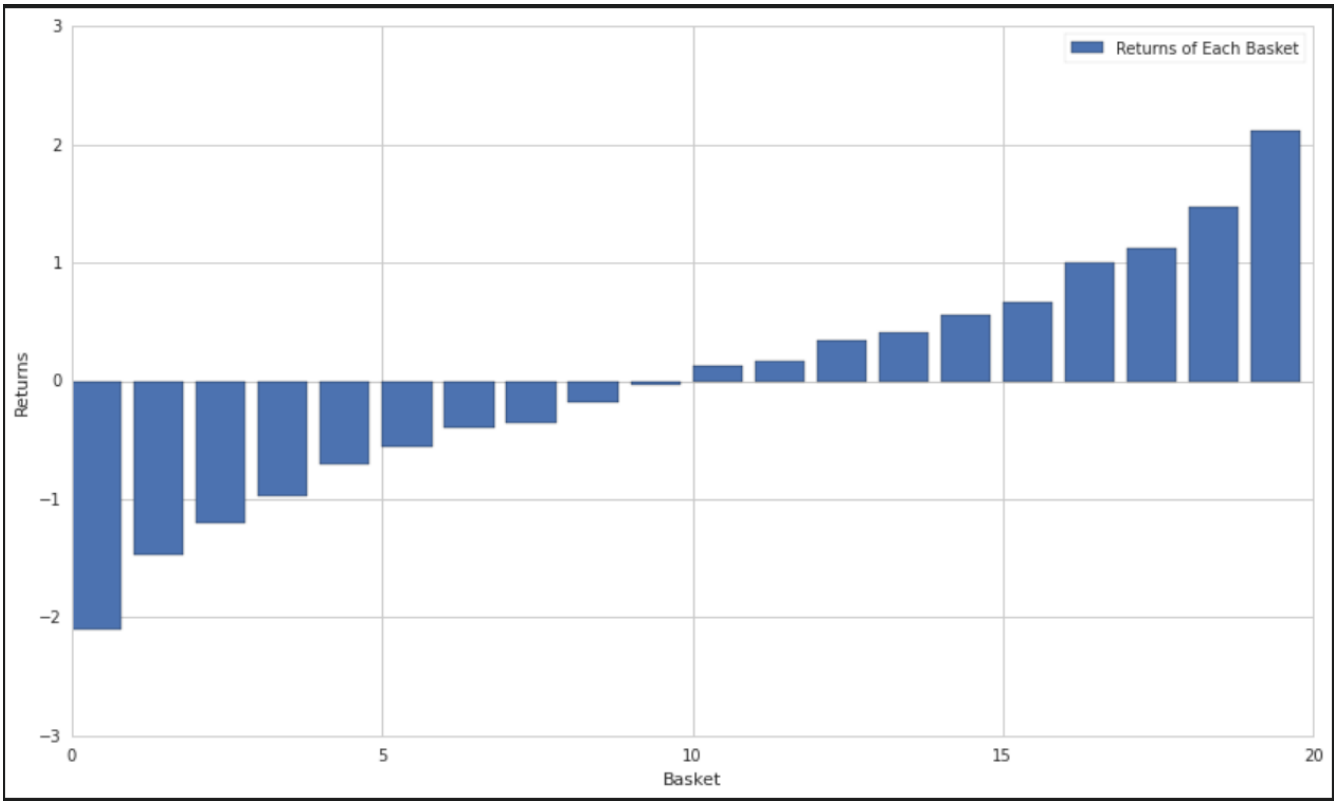
接下來我們實施長空策略，我們發現

```
# Rank the equities
ranked_data = data.sort('Factor Value')

# Compute the returns of each basket
# Baskets of size 500, so we create an empty array of shape (10000/500)
number_of_baskets = 10000/500
basket_returns = np.zeros(number_of_baskets)

for i in range(number_of_baskets):
    start = i * 500
    end = i * 500 + 500
    basket_returns[i] = ranked_data[start:end]['Returns'].mean()

# Plot the returns of each basket
plt.bar(range(number_of_baskets), basket_returns)
plt.ylabel('Returns')
plt.xlabel('Basket')
plt.legend(['Returns of Each Basket']);
```



可以看到分組後的表現，顯示利用好的因子可以得到好的分組，而作多最好的組，做空最優的組得到的報酬率為4.04

```
basket_returns[number_of_baskets-1] - basket_returns[0]
```

一些其餘的考量

Ranking Scheme

可以看出最重要的是我們如何分組(排名)，這不一定要基於因子模型，可以利用機器學習技術等等。但能不能找到一個很賺錢的排名方法就不是一個簡單回答的問題，甚至可能我們沒有辦法發現一個聖盃方法可以在任何時間都成功分出好的組，但一個好的開始方法應該是找一些存在的方法，看看能不能做一些有效的改進。

在決定完分組後，其他的部份基本上都是一樣的，所以在實施多空策略時，我們只需要修改排名策略，其餘不變。

Rebalancing Frequency

每個排名系統都會在稍微不同的時間範圍內預測回報。 基於價格的均值回歸可能在幾天內可預測，而基於價值的因子模型可能在幾個月內具有預測性。 確定模型應該預測的時間範圍非常重要，並在執行策略之前進行統計驗證。

Capital Capacity

Every strategy has a minimum and maximum amount of capital it can trade before it stops being profitable.

Number of Equities Traded

Transaction Costs

交易許多股票將導致高交易成本。假設您要購買1000股票，每次重新平衡將產生數千美元的成本。您的資本基礎必須足夠高，以使交易成本佔您的策略產生的回報的一小部分。假設您運行100,000美元並且每月賺1％，那麼每月1000美元的交易費將佔用您的所有回報。您需要以數百萬美元的價格運行該策略才能獲得超過1000的利潤股票。

一個實際的例子

```
"""
This algorithm demonstrates the concept of long-short equity. It uses a
combination of factors to construct a ranking of securities in a liquid
tradable universe. It then goes long on the highest-ranked securities and short
on the lowest-ranked securities.

For information on long-short equity strategies, please see the corresponding
lecture on our lectures page:

https://www.quantopian.com/lectures

This algorithm was developed as part of Quantopian's Lecture Series. Please
direct and questions, feedback, or corrections to feedback@quantopian.com
"""

import quantopian.algorithm as algo
import quantopian.optimize as opt
from quantopian.pipeline import Pipeline
from quantopian.pipeline.factors import SimpleMovingAverage

from quantopian.pipeline.filters import QTradableStocksUS
from quantopian.pipeline.experimental import risk_loading_pipeline

from quantopian.pipeline.data.psychsignal import stocktwits
from quantopian.pipeline.data import Fundamentals

# Constraint Parameters
MAX_GROSS_LEVERAGE = 1.0
TOTAL_POSITIONS = 600

# Here we define the maximum position size that can be held for any
# given stock. If you have a different idea of what these maximum
# sizes should be, feel free to change them. Keep in mind that the
# optimizer needs some leeway in order to operate. Namely, if your
# maximum is too small, the optimizer may be overly-constrained.
MAX_SHORT_POSITION_SIZE = 2.0 / TOTAL_POSITIONS
MAX_LONG_POSITION_SIZE = 2.0 / TOTAL_POSITIONS


def initialize(context):
    """
    A core function called automatically once at the beginning of a backtest.

    Use this function for initializing state or other bookkeeping.

    Parameters
```

```
    ----------
    context : AlgorithmContext
        An object that can be used to store state that you want to maintain in
        your algorithm. context is automatically passed to initialize,
        before_trading_start, handle_data, and any functions run via schedule_function.
        context provides the portfolio attribute, which can be used to retrieve
information
        about current positions.
    """

    algo.attach_pipeline(make_pipeline(), 'long_short_equity_template')

    # Attach the pipeline for the risk model factors that we
    # want to neutralize in the optimization step. The 'risk_factors' string is
    # used to retrieve the output of the pipeline in before_trading_start below.
    algo.attach_pipeline(risk_loading_pipeline(), 'risk_factors')

    # Schedule our rebalance function
    algo.schedule_function(func=rebalance,
                           date_rule=algo.date_rules.week_start(),
                           time_rule=algo.time_rules.market_open(hours=0, minutes=30),
                           half_days=True)

    # Record our portfolio variables at the end of day
    algo.schedule_function(func=record_vars,
                           date_rule=algo.date_rules.every_day(),
                           time_rule=algo.time_rules.market_close(),
                           half_days=True)


def make_pipeline():
    """
    A function that creates and returns our pipeline.

    We break this piece of logic out into its own function to make it easier to
    test and modify in isolation. In particular, this function can be
    copy/pasted into research and run by itself.

    Returns
    -------
    pipe : Pipeline
        Represents computation we would like to perform on the assets that make
        it through the pipeline screen.
    """
    # The factors we create here are based on fundamentals data and a moving
    # average of sentiment data
    value = Fundamentals.ebit.latest / Fundamentals.enterprise_value.latest
    quality = Fundamentals.roe.latest
    sentiment_score = SimpleMovingAverage(
        inputs=[stocktwits.bull_minus_bear],
        window_length=3,
    )
```

```
    universe = QTradableStocksUS()

    # We winsorize our factor values in order to lessen the impact of outliers
    # For more information on winsorization, please see
    # https://en.wikipedia.org/wiki/Winsorizing
    value_winsorized = value.winsorize(min_percentile=0.05, max_percentile=0.95)
    quality_winsorized = quality.winsorize(min_percentile=0.05, max_percentile=0.95)
    sentiment_score_winsorized = sentiment_score.winsorize(min_percentile=0.05,
                                                           max_percentile=0.95)

    # Here we combine our winsorized factors, z-scoring them to equalize their influence
    combined_factor = (
        value_winsorized.zscore() +
        quality_winsorized.zscore() +
        sentiment_score_winsorized.zscore()
    )

    # Build Filters representing the top and bottom baskets of stocks by our
    # combined ranking system. We'll use these as our tradeable universe each
    # day.
    longs = combined_factor.top(TOTAL_POSITIONS//2, mask=universe)
    shorts = combined_factor.bottom(TOTAL_POSITIONS//2, mask=universe)

    # The final output of our pipeline should only include
    # the top/bottom 300 stocks by our criteria
    long_short_screen = (longs | shorts)

    # Create pipeline
    pipe = Pipeline(
        columns={
            'longs': longs,
            'shorts': shorts,
            'combined_factor': combined_factor
        },
        screen=long_short_screen
    )
    return pipe


def before_trading_start(context, data):
    """
    Optional core function called automatically before the open of each market day.

    Parameters
    ----------
    context : AlgorithmContext
        See description above.
    data : BarData
        An object that provides methods to get price and volume data, check
        whether a security exists, and check the last time a security traded.
    """
    # Call algo.pipeline_output to get the output
    # Note: this is a dataframe where the index is the SIDs for all
```

```
    # securities to pass my screen and the columns are the factors
    # added to the pipeline object above
    context.pipeline_data = algo.pipeline_output('long_short_equity_template')

    # This dataframe will contain all of our risk loadings
    context.risk_loadings = algo.pipeline_output('risk_factors')


def record_vars(context, data):
    """
    A function scheduled to run every day at market close in order to record
    strategy information.

    Parameters
    ----------
    context : AlgorithmContext
        See description above.
    data : BarData
        See description above.
    """
    # Plot the number of positions over time.
    algo.record(num_positions=len(context.portfolio.positions))


# Called at the start of every month in order to rebalance
# the longs and shorts lists
def rebalance(context, data):
    """
    A function scheduled to run once every Monday at 10AM ET in order to
    rebalance the longs and shorts lists.

    Parameters
    ----------
    context : AlgorithmContext
        See description above.
    data : BarData
        See description above.
    """
    # Retrieve pipeline output
    pipeline_data = context.pipeline_data

    risk_loadings = context.risk_loadings

    # Here we define our objective for the Optimize API. We have
    # selected MaximizeAlpha because we believe our combined factor
    # ranking to be proportional to expected returns. This routine
    # will optimize the expected return of our algorithm, going
    # long on the highest expected return and short on the lowest.
    objective = opt.MaximizeAlpha(pipeline_data.combined_factor)

    # Define the list of constraints
    constraints = []
    # Constrain our maximum gross leverage
```

```
    constraints.append(opt.MaxGrossExposure(MAX_GROSS_LEVERAGE))

    # Require our algorithm to remain dollar neutral
    constraints.append(opt.DollarNeutral())

    # Add the RiskModelExposure constraint to make use of the
    # default risk model constraints
    neutralize_risk_factors = opt.experimental.RiskModelExposure(
        risk_model_loadings=risk_loadings,
        version=0
    )
    constraints.append(neutralize_risk_factors)

    # With this constraint we enforce that no position can make up
    # greater than MAX_SHORT_POSITION_SIZE on the short side and
    # no greater than MAX_LONG_POSITION_SIZE on the long side. This
    # ensures that we do not overly concentrate our portfolio in
    # one security or a small subset of securities.
    constraints.append(
        opt.PositionConcentration.with_equal_bounds(
            min=-MAX_SHORT_POSITION_SIZE,
            max=MAX_LONG_POSITION_SIZE
        ))

    # Put together all the pieces we defined above by passing
    # them into the algo.order_optimal_portfolio function. This handles
    # all of our ordering logic, assigning appropriate weights
    # to the securities in our universe to maximize our alpha with
    # respect to the given constraints.
    algo.order_optimal_portfolio(
        objective=objective,
        constraints=constraints
    )
```