02_multiple_factors_quantopian_research

September 29, 2021

1 Combining Factors from Multiple Sources

The Quantopian research environment is tailored to the rapid testing of predictive alpha factors. The process is very similar because it builds on zipline, but offers much richer access to data sources. The following code sample illustrates how to compute alpha factors not only from market data as previously but also from fundamental and alternative data.

Quantopian provides several hundred MorningStar fundamental variables for free and also includes stocktwits signals as an example of an alternative data source. There are also custom universe definitions such as QTradableStocksUS that applies several filters to limit the backtest universe to stocks that were likely tradeable under realistic market conditions.

[ ]:
```
from quantopian.research import run_pipeline
from quantopian.pipeline import Pipeline
from quantopian.pipeline.data.builtin import USEquityPricing
from quantopian.pipeline.data.morningstar import income_statement,␣
 ↪operation_ratios, balance_sheet
from quantopian.pipeline.data.psychsignal import stocktwits
from quantopian.pipeline.factors import CustomFactor, SimpleMovingAverage,␣
 ↪Returns
from quantopian.pipeline.filters import QTradableStocksUS

import numpy as np
import pandas as pd
from time import time
```

[8]:
```
# periods in trading days
MONTH = 21
QTR = 4 * MONTH
YEAR = 12 * MONTH
```

We will use a custom `AggregateFundamentals` class to use the last reported fundamental data point. This aims to address the fact that fundamentals are reported quarterly, and Quantopian does not currently provide an easy way to aggregate historical data, say to obtain the sum of the last four quarters, on a rolling basis.

We will again use the custom `MeanReversion` factor from 01_single_factor_zipline. We will also compute several other factors for the given universe definition using the `rank()` method's mask parameter.

This algorithm uses a naive method to combine the six individual factors by simply adding the ranks of assets for each of these factors. Instead of equal weights, we would like to take into account the relative importance and incremental information in predicting future returns. The ML algorithms of the next chapters will allow us to do exactly this, using the same backtesting framework.

Execution also relies on run_algorithm(), but the return DataFrame on the Quantopian platform only contains the factor values created by the Pipeline. This is convenient because this data format can be used as input for alphalens, the library for the evaluation of the predictive performance of alpha factors.

[12]:
```
class AggregateFundamentals(CustomFactor):
    def compute(self, today, assets, out, inputs):
        out[:] = inputs[0]


class MeanReversion(CustomFactor):
    inputs = [Returns(window_length=MONTH)]
    window_length = YEAR

    def compute(self, today, assets, out, monthly_returns):
        df = pd.DataFrame(monthly_returns)
        out[:] = df.iloc[-1].sub(df.mean()).div(df.std())


def compute_factors():

    universe = QTradableStocksUS()

    profitability = (AggregateFundamentals(inputs = [income_statement.
 ↪gross_profit],
                                           window_length = YEAR) /
                     balance_sheet.total_assets.latest).rank(mask=universe)

    roic = operation_ratios.roic.latest.rank(mask=universe)

    ebitda_yield = (AggregateFundamentals(inputs = [income_statement.ebitda],
                                          window_length = YEAR) /
                    USEquityPricing.close.latest).rank(mask=universe)

    mean_reversion = MeanReversion().rank(mask=universe)

    price_momentum = Returns(window_length=QTR).rank(mask=universe)

    sentiment = SimpleMovingAverage(inputs=[stocktwits.bull_minus_bear],
                                    window_length=5).rank(mask=universe)

    factor = profitability + roic + ebitda_yield + mean_reversion +␣
 ↪price_momentum + sentiment
```

```
    return Pipeline(
        columns={'Profitability': profitability,
                 'ROIC': roic,
                 'EBITDA Yield': ebitda_yield,
                 "Mean Reversion (1M)": mean_reversion,
                 'Sentiment': sentiment,
                 "Price Momentum (3M)": price_momentum,
                 'Alpha Factor': factor})
```

[ ]:
```
start_timer = time()
start = pd.Timestamp("2015-01-01")
end = pd.Timestamp("2018-01-01")
results = run_pipeline(compute_factors(), start_date=start, end_date=end)
results.index.names = ['date', 'security']
print("Pipeline run time {:.2f} secs".format(time() - start_timer))
```

[ ]:
```
results.head()
```

[ ]: