ST790: Quantopian Final Project

Introduction

As fund managers, we were tasked with constructing a cross-sectional, long-short US equity strategy on Quantopian. There were not many explicit constraints on the specific strategy we utilized, but it must pass the following criteria: (i) trade liquid stocks, (ii) have no more than 5% of capital invested in any one asset, (iii) have no more than 10% net dollar exposure, (iv) achieve mean daily turnover between 5% and 65% over a 63-trading-day rolling window, (v) attain gross leverage between 0.8x and 1.1x, (vi) have low correlation to the market, (vii) have less than 20% exposed to each of the 11 sectors as defined on Quantopian, and (viii) result in positive returns. While the last return criteria was not a constraint we included in our optimization, we did design our algorithm with the rest of the seven criteria in mind.

Trading Strategy

The backtests we describe below are derived from the following trading algorithm:

1. Once a week, we choose a universe of liquid assets from `QTradeableStocksUS` that pass the following filters:
    - It is not trading within 2 days of any earnings announcements as assets are generally more volatile within these dates.
    - It has not been announced as an acquistion target. To further reduce any possible volatility, we avoid acquisition targets as they often pose huge risk to quant strategies.
    - We are able to calculate a 5 day moving average of the bull-minus-bear signal from the `StockTwits` API.
2. Every day, we build an alpha vector for the universe of liquid assets filtered from our step above. The alpha model we use is quite simple: we rank the assets by its bull-to-bear intensity, averaged over the past 5 days as evaluated from `StockTwits`, and find a set of new portfolio weights that maximizes the sum of each asset's weight times this alpha value. Our objective defined in `MaximizeAlpha` is thus simply a function of this sentiment datastream as we believe this ranking is similar to expected returns of each asset. As a result, our routine effectively goes long on assets with high bullish signal and short on those with a high bearish signal.
3. Once a week, we calculate the portfolio that maximizes the alpha-weighted sum of our position sizes, subject to the following constraints:
    - Our portfolio maintains a gross leverage of, or less than, 1.0x.
    - Our portfolio has no more than 5% in any single asset.
    - Our portfolio does not pass mean daily turnover of 80%.

With this simple strategy, we achieve the following leaderboard results at the end of our one-month trading period from November 1^{st} to November 30^{th}:

Key to our strategy is the output from the `bull_minus_bear` API call as it is this signal that is fed into the optimization function and this result that determines the order size of each asset.

	Metric	Our Result	Overall
1	rank	64	-
2	name	Gray Fox	-
3	score	0.502	0.36
4	max_beta_to_spy_126day	0.076	0.14
5	max_cumulative_common_returns	0.009	0.04
6	max_leverage	1.047	1.05
7	max_max_drawdown	0	-0.00
8	max_net_dollar_exposure	0.032	0.04
9	max_total_returns	0.025	0.14
10	min_total_returns	-0.007	-0.02
11	max_turnover	0.905	1.07
12	max_volatility_126day	0.044	0.06

While we do not have exact clarity on the natural language processing engine that calculates the bullish intensity and bearish intensity of a stock, we do know how the `bull_minus_bear` signal arises; traders attached either a bull emoji or a bear emoji to any message they release on the `StockTwits` platform as well as a ticker symbol that identifies which asset is under discussion. While traders attached a clear label to the asset of interest, `StockTwits` also has an in-house proprietary algorithm that processes some of the language in the message to ascribe an intensity level of the bull or bear indicator. Messages across the `StockTwits` platform is thus aggregated to arrive a sentiment score that is a function of subtracting the bearish intensity from the bullish intensity result.

Choice of the sentiment score

In addition to the ease of implementation, we chose to rely on the sentiment factor as it seems to have some good characteristics, namely:

1. *Predictive alpha* We calculated the mean information coefficient using the built-in function in `alphalens` and found our sentiment signal matches the direction of actual asset returns. As shown in the top-left of the figure below, we only get the full exposure of this factor 5 days after this signal becomes available. In fact, prior to 5 trading days, it appears the signal hurts us, in that its forecast is negative. Counterintuitively, after we cross 5 trading days, it appears our sentiment factor gains more predictive power. This is further corroborated in the top-right graph in that returns are highest when the signal is delayed by four days. Given this characteristic, we may want to build greater exposure to this factor since it still benefits us after 10 trading days. We can also leverage the positive effects of this factor by increasing our turnover constraints as it appears it does not hurt our portfolio to keep in there for a longer period of time. Nevertheless, the mean information coefficient still lingers around 0, suggesting the forecasting benefits provided by our sentiment factor may be no better than the results we get from randomly selecting our asset weights.

2. *Low exposures* We quantified the exposures via the `perf_attrib` function in `pyfolio`. As shown in the bottom-left, our sentiment factor appears to have relatively low exposures throughout, with most of the returns from volatility risk. However, there does appear to be persistent exposure to value and short term reversal, in that it is shifted to the left of zero; and persistent exposure to size and momentum, in that it is shifted to the right of zero. Ideally, we would choose a factor that doesn't display this skew, but the skew does not appear too large. We also benefit from the fact that the width of each of our box and

whisker plots are not that wide, so our exposures do not vary much over time. Yet, this analysis was conducted only over the available sentiment signal in 2018, so there may be other extended periods prior to 2018 in which our factor over or underperforms. Given what we have here, we may want to identify asset classes to include in our portfolio that would offset the exposure risks identified above.

Unfortunately, when we examine our exposures through cumulative returns and volatility, we find that our sentiment factor may not be as strong as we desired. As shown by the left bar graph (bottom-right), most of our returns are obtained from exposure to common risk factors, and, as shown in the right bar graph, are in fact driven by these common risk factors. Ideally, we would like to strip away the contribution from these common risk factors as portfolio performance from this exposure can easily be replicated from ETFs or other easy, cost-effective methods.

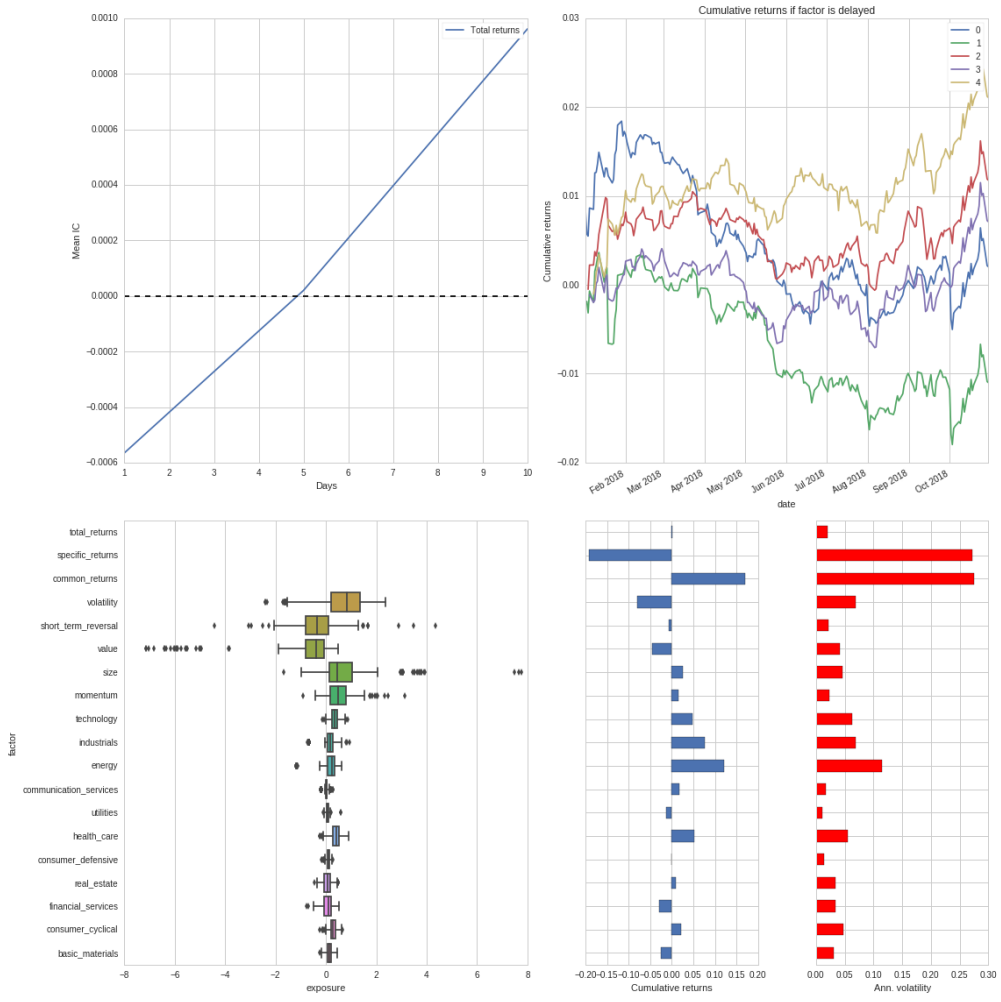

Figure 0.1: Examination of (top-right) information coefficient averaged over all possible asset returns, (top-left) cumulative returns when factor is delayed, (bottom-left) risk exposures, and (bottom-right) cumulative returns and volatility.

Backtesting

Given the extensive literature that sentiment data can predict stock price, we forged ahead and submitted our trading strategy on Quantopian. The back test results

Evaluation

Future Improvement

There are a number of avenues we might consider to improve upon our simplistic approach:

1. *Using non-trading day sentiment data*: While we relied on what was provided to us by `Stocktwits` we should attempt building our own classifier from possiblilty richer sources of messages (e.g., Weibo, Twitter) that can not only provide

- fundamental value measures?
- analyzed bull and bear tagged messages on each asset

Sentiment

- looking at relative frequency of words falling under various themes
- SW uses cashtags with stock ticker symbol, like hashtag, to index ppls thoughts and ideas about the company and stock
- understand that anything from social media, or anything market or finance related comes with its fair share of BS
- stockTwits is no exception
- people "BuY right now".. claiming they made excessive gains
-
- feed stence into algo and know how people are "feeling"
- rely on a pretty sophistic natural language processing engine can parse through online market conversations that can detect trader moods bullishness and bearishnes and intensity

algorithm rules

Trading Strategy

November Portfolio Performance

The following displays my portfolio performance between Nov 1 and Nov 30

Back Testing

Shown for three different time periods in the 21st century: - 2000-2002 - 2008-2010 - 2015-2018 # this doesnt work bc of stocktwits limitation
since we rely on pyschsignal.stocktwits 2007-01-01 to 2018-10-30

Discussion

returns, positions, leverage, risk, control

Also make a poster and print!

Appendix

Quantopian Submission

```
# Import Algorithm API functions
from quantopian.algorithm import (
    attach_pipeline,
    pipeline_output,
    order_optimal_portfolio,
)
# Import Optimize API module
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
# Get event data
from quantopian.pipeline.factors.eventvestor import (
    BusinessDaysUntilNextEarnings,
    BusinessDaysSincePreviousEarnings,
)
from quantopian.pipeline.filters.eventvestor import IsAnnouncedAcqTarget
from quantopian.pipeline.factors import BusinessDaysSincePreviousEvent
def initialize(context):
    # Constraint parameters
    context.max_leverage = 1.0
    context.max_pos_size = 0.05
    context.max_turnover = 0.8
    # Attach data pipelines
    attach_pipeline(
        make_pipeline(),
        'data_pipe'
```

```
    )
    attach_pipeline(
        risk_loading_pipeline(),
        'risk_pipe'
    )
    # Schedule rebalance function
    schedule_function(
        rebalance,
        date_rules.week_start(),
        time_rules.market_open(),
    )
def before_trading_start(context, data):
    # Get pipeline outputs and
    # store them in context
    context.output = pipeline_output('data_pipe')
    context.risk_factor_betas = pipeline_output('risk_pipe')
# Pipeline definition
def make_pipeline():

    not_near_earnings = ~((BusinessDaysUntilNextEarnings() <= 2) |
      (BusinessDaysSincePreviousEarnings() <= 2))

    not_acq_tar = ~IsAnnouncedAcqTarget()

    universe = (
        QTradableStocksUS()
        & not_near_earnings
        & not_acq_tar
    )

    sentiment_score = SimpleMovingAverage(
        inputs=[stocktwits.bull_minus_bear],
        window_length=5,
        mask=universe
    )
    return Pipeline(
        columns={
            'sentiment_score': sentiment_score,
        },
        screen=sentiment_score.notnull()
    )
def rebalance(context, data):
    # Create MaximizeAlpha objective using
    # sentiment_score data from pipeline output
    objective = opt.MaximizeAlpha(
      context.output.sentiment_score
    )
```

```
    # 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)
    # Constrain portfolio turnover
    max_turnover = opt.MaxTurnover(context.max_turnover)
    # Constrain target portfolio's risk exposure
    factor_risk_constraints = opt.experimental.RiskModelExposure(
        context.risk_factor_betas,
        version=opt.Newest
    )
    # Rebalance portfolio using objective
    # and list of constraints
    order_optimal_portfolio(
        objective=objective,
        constraints=[
            max_leverage,
            constrain_pos_size,
            max_turnover,
            factor_risk_constraints,
        ]
    )
```