

06_performance_eval_alphalens

September 29, 2021

1 Separating signal and noise – how to use alphalens

Quantopian has open sourced the Python library, alphalens, for the performance analysis of predictive stock factors that integrates well with the backtesting library zipline and the portfolio performance and risk analysis library pyfolio that we will explore in the next chapter. alphalens facilitates the analysis of the predictive power of alpha factors concerning the: - Correlation of the signals with subsequent returns - Profitability of an equal or factor-weighted portfolio based on a (subset of) the signals - Turnover of factors to indicate the potential trading costs - Factor-performance during specific events - Breakdowns of the preceding by sector

The analysis can be conducted using tearsheets or individual computations and plots.

This notebook requires the `conda` environment `backtest`. Please see the [installation instructions](#) for running the latest Docker image or alternative ways to set up your environment.

1.1 Imports & Settings

```
[1]: import warnings
      warnings.filterwarnings('ignore')
```

```
[2]: %matplotlib inline
      import re
      from alphalens.utils import get_clean_factor_and_forward_returns
      from alphalens.performance import *
      from alphalens.plotting import *
      from alphalens.tears import *

      import seaborn as sns
      import matplotlib.pyplot as plt
```

```
[3]: sns.set_style('whitegrid')
```

1.2 Creating forward returns and factor quantiles

To utilize `alphalens`, we need to provide signals for a universe of assets like those returned by the ranks of the MeanReversion factor, and the forward returns earned by investing in an asset for a given holding period. .

This notebook uses the file `single_factor.pickle` with the results generated in the notebook `single_factor_zipline.ipynb` in this directory.

We will recover the prices from the `single_factor.pickle` file as follows (`factor_data` accordingly):

```
[4]: performance = pd.read_pickle('single_factor.pickle')
```

```
[5]: performance.info()
```

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 755 entries, 2015-01-02 21:00:00+00:00 to 2017-12-29
21:00:00+00:00
Data columns (total 39 columns):
#   Column                                Non-Null Count  Dtype
---  -
0   period_open                          755 non-null    datetime64[ns, UTC]
1   period_close                         755 non-null    datetime64[ns, UTC]
2   shorts_count                        755 non-null    int64
3   long_value                          755 non-null    float64
4   short_value                         755 non-null    float64
5   long_exposure                      755 non-null    float64
6   pnl                                755 non-null    float64
7   capital_used                       755 non-null    float64
8   short_exposure                     755 non-null    float64
9   orders                             755 non-null    object
10  transactions                        755 non-null    object
11  positions                          755 non-null    object
12  gross_leverage                     755 non-null    float64
13  starting_exposure                  755 non-null    float64
14  net_leverage                       755 non-null    float64
15  ending_exposure                    755 non-null    float64
16  starting_value                     755 non-null    float64
17  ending_value                       755 non-null    float64
18  starting_cash                      755 non-null    float64
19  ending_cash                        755 non-null    float64
20  portfolio_value                    755 non-null    float64
21  returns                            755 non-null    float64
22  longs_count                        755 non-null    int64
23  algo_volatility                    754 non-null    float64
24  benchmark_period_return            755 non-null    float64
25  benchmark_volatility                754 non-null    float64
26  alpha                              0 non-null     object
27  beta                               0 non-null     object
28  sharpe                             753 non-null    float64
29  sortino                            753 non-null    float64
30  max_drawdown                       755 non-null    float64
31  max_leverage                       755 non-null    float64
```

```

32 excess_return          755 non-null    float64
33 treasury_period_return 755 non-null    float64
34 trading_days           755 non-null    int64
35 period_label           755 non-null    object
36 algorithm_period_return 755 non-null    float64
37 factor_data            754 non-null    object
38 prices                 754 non-null    object
dtypes: datetime64[ns, UTC](2), float64(26), int64(3), object(8)
memory usage: 235.9+ KB

```

```

[6]: prices = pd.concat([df.to_frame(d) for d, df in performance.prices.dropna().
    ↳items()],axis=1).T
prices.columns = [re.findall(r"\[(.+)\]", str(col))[0] for col in prices.
    ↳columns]
prices.index = prices.index.normalize()
prices.info()

```

```

<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 754 entries, 2015-01-05 00:00:00+00:00 to 2017-12-29
00:00:00+00:00
Columns: 1649 entries, A to NETE
dtypes: float64(1649)
memory usage: 9.5 MB

```

```

[7]: factor_data = pd.concat([df.to_frame(d) for d, df in performance.factor_data.
    ↳dropna().items()],axis=1).T
factor_data.columns = [re.findall(r"\[(.+)\]", str(col))[0] for col in_
    ↳factor_data.columns]
factor_data.index = factor_data.index.normalize()
factor_data = factor_data.stack()
factor_data.index.names = ['date', 'asset']
factor_data.head()

```

```

[7]: date          asset
2015-01-05 00:00:00+00:00 A          2707.0
                                AAL          870.0
                                AAP          1253.0
                                AAPL         2977.0
                                ABBV         2806.0

dtype: float64

```

```

[8]: with pd.HDFStore('../data/assets.h5') as store:
    sp500 = store['sp500/stooq'].close
sp500 = sp500.resample('D').ffill().tz_localize('utc').filter(prices.index.
    ↳get_level_values(0))
sp500.head()

```

```
[8]: Date
2015-01-05 00:00:00+00:00    2020.58
2015-01-06 00:00:00+00:00    2002.61
2015-01-07 00:00:00+00:00    2025.90
2015-01-08 00:00:00+00:00    2062.14
2015-01-09 00:00:00+00:00    2044.81
Name: close, dtype: float64
```

We can create the alphas input data in the required format using the `get_clean_factor_and_forward_returns` utility function that also returns the signal quantiles and the forward returns for the given holding periods:

```
[9]: HOLDING_PERIODS = (5, 10, 21, 42)
QUANTILES = 5
alphalens_data = get_clean_factor_and_forward_returns(factor=factor_data,
                                                         prices=prices,
                                                         periods=HOLDING_PERIODS,
                                                         quantiles=QUANTILES)
```

Dropped 5.6% entries from factor data: 5.6% in forward returns computation and 0.0% in binning phase (set `max_loss=0` to see potentially suppressed Exceptions). `max_loss` is 35.0%, not exceeded: OK!

The `alphalens_data` DataFrame contains the returns on an investment in the given asset on a given date for the indicated holding period, as well as the factor value, that is, the asset's `MeanReversion` ranking on that date, and the corresponding quantile value:

```
[10]: alphalens_data.head()
```

```
[10]:
```

		5D	10D	21D	42D \
date	asset				
2015-01-05 00:00:00+00:00	A	0.007789	-0.046985	-0.027889	0.072864
	AAL	-0.079722	-0.020882	-0.095684	-0.103295
	AAP	0.015722	-0.024350	-0.003196	-0.010865
	AAPL	0.028235	0.023247	0.116518	0.214965
	ABBV	0.017169	-0.018561	-0.061098	-0.064811

		factor	factor_quantile
date	asset		
2015-01-05 00:00:00+00:00	A	2707.0	5
	AAL	870.0	1
	AAP	1253.0	2
	AAPL	2977.0	5
	ABBV	2806.0	5

```
[11]: alphalens_data.reset_index().head().to_csv('factor_data.csv', index=False)
```

The forward returns and the signal quantiles are the basis for evaluating the predictive power of the signal. Typically, a factor should deliver markedly different returns for distinct quantiles, such

as negative returns for the bottom quintile of the factor values and positive returns for the top quintile.

1.3 Summary Tear Sheet

```
[12]: create_summary_tear_sheet(alphalens_data)
```

Quantiles Statistics

	min	max	mean	std	count	count %
factor_quantile						
1	1.0	1011.0	303.057978	188.562449	142313	20.020145
2	352.0	1636.0	856.678631	234.705707	142117	19.992572
3	794.0	2153.0	1417.174265	259.942787	142019	19.978786
4	1273.0	2621.0	1979.178578	253.504821	142117	19.992572
5	1827.0	3050.0	2519.578804	227.348609	142283	20.015925

Returns Analysis

	5D	10D	21D	42D
Ann. alpha	0.046	0.036	0.009	0.001
beta	0.083	0.098	0.077	0.019
Mean Period Wise Return Top Quintile (bps)	11.724	9.110	3.948	-0.376
Mean Period Wise Return Bottom Quintile (bps)	-16.862	-13.259	-4.742	-1.979
Mean Period Wise Spread (bps)	28.587	22.343	8.711	1.582

Information Analysis

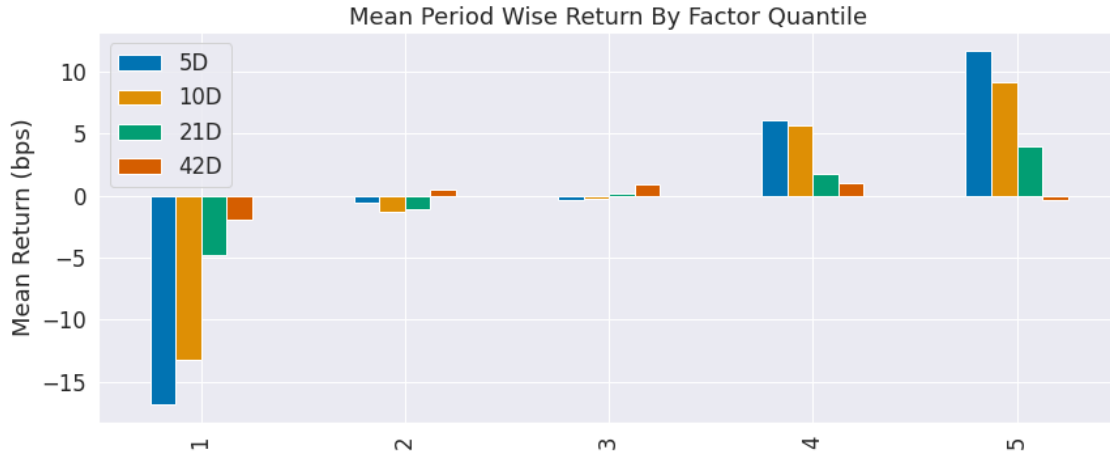
	5D	10D	21D	42D
IC Mean	0.022	0.026	0.017	0.003
IC Std.	0.140	0.127	0.116	0.115
Risk-Adjusted IC	0.160	0.207	0.148	0.027
t-stat(IC)	4.261	5.529	3.953	0.729
p-value(IC)	0.000	0.000	0.000	0.466
IC Skew	0.372	0.266	0.115	0.113
IC Kurtosis	0.054	-0.515	-0.333	-0.557

Turnover Analysis

	5D	10D	21D	42D
Quantile 1 Mean Turnover	0.411	0.590	0.830	0.831
Quantile 2 Mean Turnover	0.645	0.740	0.804	0.812
Quantile 3 Mean Turnover	0.679	0.765	0.808	0.812
Quantile 4 Mean Turnover	0.642	0.741	0.810	0.814
Quantile 5 Mean Turnover	0.394	0.569	0.811	0.819

	5D	10D	21D	42D
Mean Factor Rank Autocorrelation	0.713	0.454	-0.013	-0.017

<Figure size 432x288 with 0 Axes>



1.4 Predictive performance by factor quantiles - Returns Analysis

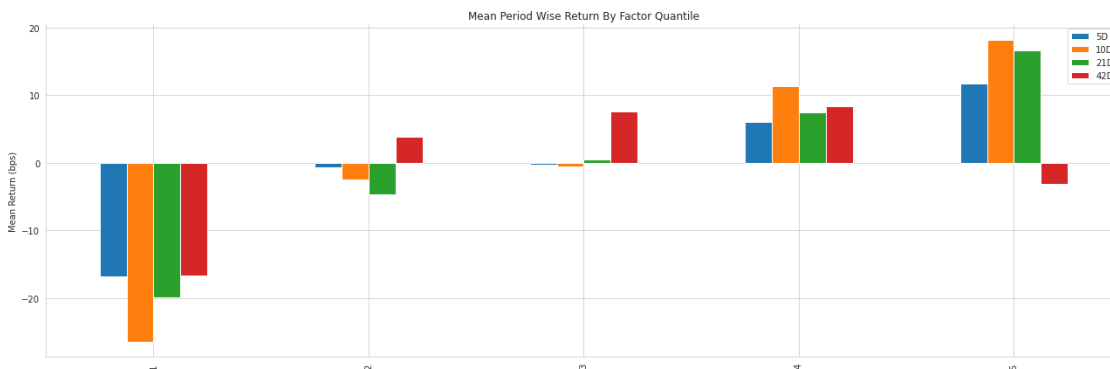
As a first step, we would like to visualize the average period return by factor quantile. We can use the built-in function `mean_return_by_quantile` from the `performance` and `plot_quantile_returns_bar` from the `plotting` modules

```
[13]: mean_return_by_q, std_err = mean_return_by_quantile(alphalens_data)
mean_return_by_q_norm = mean_return_by_q.apply(lambda x: x.add(1).pow(1/int(x.
↪name[: -1])).sub(1))
```

1.4.1 Mean Return by Holding Period and Quintile

The result is a bar chart that breaks down the mean of the forward returns for the four different holding periods based on the quintile of the factor signal. As you can see, the bottom quintiles yielded markedly more negative results than the top quintiles, except for the longest holding period:

```
[14]: plot_quantile_returns_bar(mean_return_by_q)
plt.tight_layout()
sns.despine();
```



The 10D holding period provides slightly better results for the first and fourth quintiles. We would also like to see the performance over time of investments driven by each of the signal quintiles.

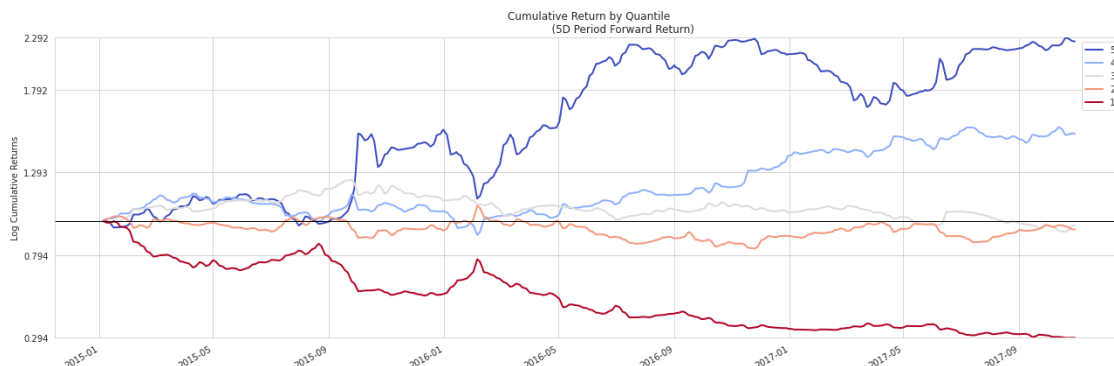
We will calculate daily, as opposed to average returns for the 5D holding period, and alphalens will adjust the period returns to account for the mismatch between daily signals and a longer holding period (for details, see docs):

```
[15]: mean_return_by_q_daily, std_err = mean_return_by_quantile(alphalens_data,
    ↪by_date=True)
```

1.4.2 Cumulative 5D Return

The resulting line plot shows that, for most of this three-year period, the top two quintiles significantly outperformed the bottom two quintiles. However, as suggested by the previous plot, signals by the fourth quintile produced a better performance than those by the top quintile

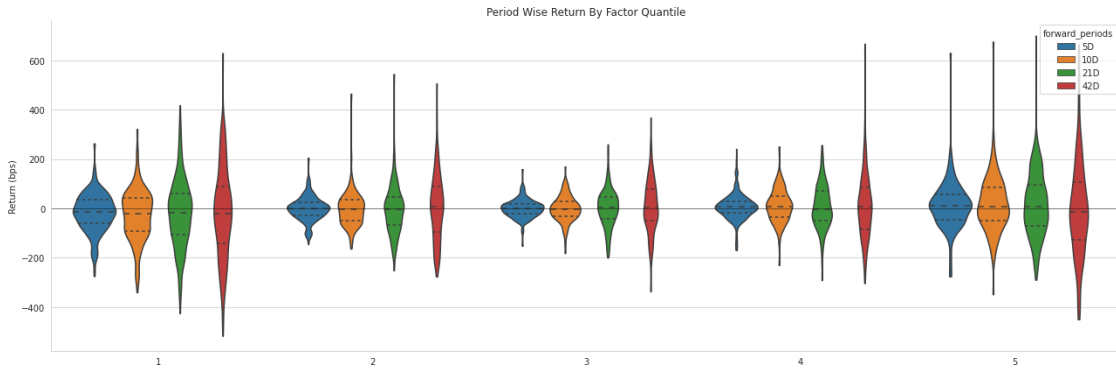
```
[16]: plot_cumulative_returns_by_quantile(mean_return_by_q_daily['5D'], period='5D',
    ↪freq=None)
plt.tight_layout()
sns.despine();
```



1.4.3 Return Distribution by Holding Period and Quintile

This distributional plot highlights that the range of daily returns is fairly wide and, despite different means, the separation of the distributions is very limited so that, on any given day, the differences in performance between the different quintiles may be rather limited:

```
[17]: plot_quantile_returns_violin(mean_return_by_q_daily)
plt.tight_layout()
sns.despine();
```



1.5 Information Coefficient

Most of this book is about the design of alpha factors using ML models. ML is about optimizing some predictive objective, and in this section, we will introduce the key metrics used to measure the performance of an alpha factor. We will define alpha as the average return in excess of a benchmark. This leads to the information ratio (IR) that measures the average excess return per unit of risk taken by dividing alpha by the tracking risk. When the benchmark is the risk-free rate, the IR corresponds to the well-known Sharpe ratio, and we will highlight crucial statistical measurement issues that arise in the typical case when returns are not normally distributed. We will also explain the fundamental law of active management that breaks the IR down into a combination of forecasting skill and a strategy's ability to effectively leverage the forecasting skills.

1.5.1 5D Information Coefficient (Rolling Average)

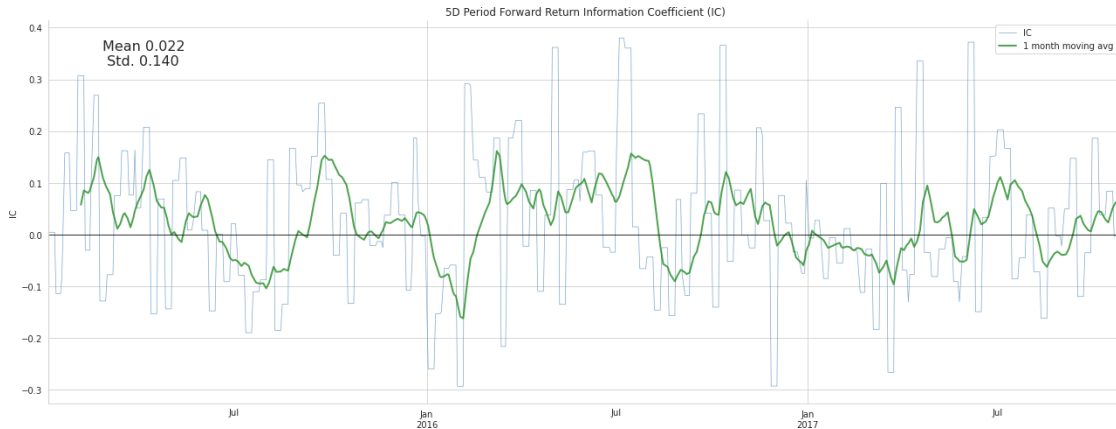
The goal of alpha factors is the accurate directional prediction of future returns. Hence, a natural performance measure is the correlation between an alpha factor's predictions and the forward returns of the target assets.

It is better to use the non-parametric Spearman rank correlation coefficient that measures how well the relationship between two variables can be described using a monotonic function, as opposed to the Pearson correlation that measures the strength of a linear relationship.

We can obtain the information coefficient using `alphalens`, which relies on `scipy.stats.spearmanr` under the hood.

The `factor_information_coefficient` function computes the period-wise correlation and `plot_ic_ts` creates a time-series plot with one-month moving average:

```
[18]: ic = factor_information_coefficient(alphalens_data)
      plot_ic_ts(ic[['5D']])
      plt.tight_layout()
      sns.despine();
```

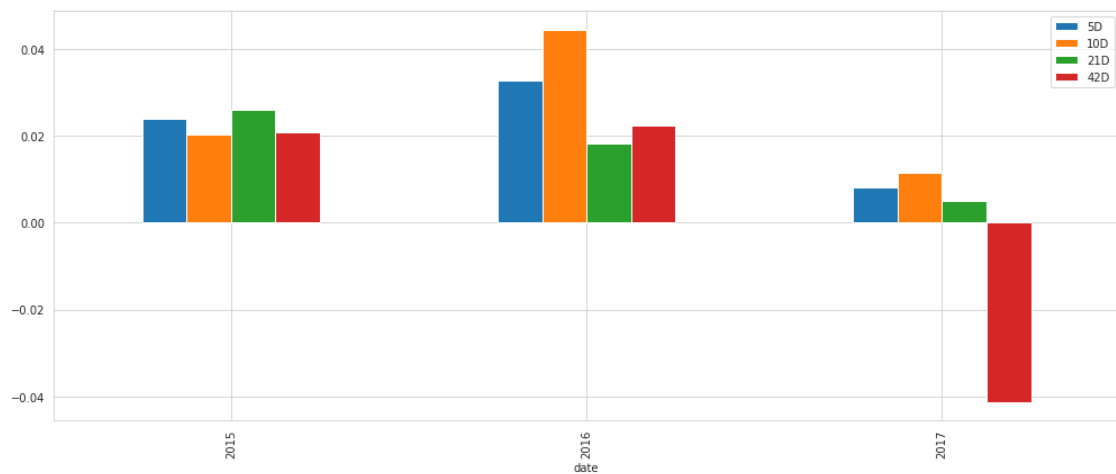



1.5.2 Information Coefficient by Holding Period

This time series plot shows extended periods with significantly positive moving-average IC. An IC of 0.05 or even 0.1 allows for significant outperformance if there are sufficient opportunities to apply this forecasting skill, as the fundamental law of active management will illustrate:

A plot of the annual mean IC highlights how the factor's performance was historically uneven:

```
[19]: ic = factor_information_coefficient(alphalens_data)
ic_by_year = ic.resample('A').mean()
ic_by_year.index = ic_by_year.index.year
ic_by_year.plot.bar(figsize=(14, 6))
plt.tight_layout();
```



1.6 Turnover Tear Sheet

Factor turnover measures how frequently the assets associated with a given quantile change, that is, how many trades are required to adjust a portfolio to the sequence of signals. More specifically, it measures the share of assets currently in a factor quantile that was not in that quantile in the last period.

```
[20]: create_turnover_tear_sheet(alphalens_data);
```

Turnover Analysis

	5D	10D	21D	42D
Quantile 1 Mean Turnover	0.411	0.590	0.830	0.831
Quantile 2 Mean Turnover	0.645	0.740	0.804	0.812
Quantile 3 Mean Turnover	0.679	0.765	0.808	0.812
Quantile 4 Mean Turnover	0.642	0.741	0.810	0.814
Quantile 5 Mean Turnover	0.394	0.569	0.811	0.819
	5D	10D	21D	42D
Mean Factor Rank Autocorrelation	0.713	0.454	-0.013	-0.017

<Figure size 432x288 with 0 Axes>

