08_backtesting_with_zipline

September 29, 2021

1 Backtesting with zipline - Pipeline API with Custom Data

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]:
```
from pathlib import Path
from collections import defaultdict
from time import time
import warnings

import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import pandas_datareader.data as web
from logbook import Logger, StderrHandler, INFO, WARNING

from zipline import run_algorithm
from zipline.api import (attach_pipeline, pipeline_output,
                         date_rules, time_rules, record,
                         schedule_function, commission, slippage,
                         set_slippage, set_commission, set_max_leverage,
                         order_target, order_target_percent,
                         get_open_orders, cancel_order)
from zipline.data import bundles
from zipline.utils.run_algo import load_extensions
from zipline.pipeline import Pipeline, CustomFactor
from zipline.pipeline.data import Column, DataSet
from zipline.pipeline.domain import US_EQUITIES
from zipline.pipeline.filters import StaticAssets
from zipline.pipeline.loaders import USEquityPricingLoader
from zipline.pipeline.loaders.frame import DataFrameLoader
from trading_calendars import get_calendar

import pyfolio as pf
```

```
from pyfolio.plotting import plot_rolling_returns, plot_rolling_sharpe
from pyfolio.timeseries import forecast_cone_bootstrap

from alphalens.tears import (create_returns_tear_sheet,
                             create_summary_tear_sheet,
                             create_full_tear_sheet)

from alphalens.performance import mean_return_by_quantile
from alphalens.plotting import plot_quantile_returns_bar
from alphalens.utils import get_clean_factor_and_forward_returns, rate_of_return
```

[2]:
```
sns.set_style('whitegrid')
warnings.filterwarnings('ignore')
np.random.seed(42)
idx = pd.IndexSlice
```

[3]:
```
results_path = Path('results', 'cnn_for_trading')
if not results_path.exists():
    results_path.mkdir()
```

1.2 Alphalens Analysis

[4]:
```
DATA_STORE = Path('..', 'data', 'assets.h5')
```

[12]:
```
def get_trade_prices(tickers):
    prices = (pd.read_hdf(DATA_STORE, 'quandl/wiki/prices').swaplevel().
 ↪sort_index())
    prices.index.names = ['symbol', 'date']
    prices = prices.loc[idx[tickers, '2010':'2018'], 'adj_open']
    return (prices
            .unstack('symbol')
            .sort_index()
            .shift(-1)
            .tz_localize('UTC'))
```

[13]:
```
predictions = (pd.read_hdf(results_path / 'predictions.h5', 'predictions')
               .iloc[:, :4]
               .mean(1)
               .to_frame('prediction'))
```

[14]:
```
factor = (predictions
          .unstack('symbol')
          .asfreq('D')
          .dropna(how='all')
          .stack()
          .tz_localize('UTC', level='date')
```

```
          .sort_index())
tickers = factor.index.get_level_values('symbol').unique()
```

[15]:
```
factor.info()
```

```
<class 'pandas.core.frame.DataFrame'>
MultiIndex: 893670 entries, (2011-04-27 00:00:00+00:00, A) to (2017-12-28
00:00:00+00:00, ZTS)
Data columns (total 1 columns):
prediction    893670 non-null float32
dtypes: float32(1)
memory usage: 6.8+ MB
```

[16]:
```
trade_prices = get_trade_prices(tickers)
```

[17]:
```
trade_prices.info()
```

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 2072 entries, 2010-01-04 to 2018-03-27
Columns: 600 entries, A to ZTS
dtypes: float64(600)
memory usage: 9.5 MB
```

[18]:
```
factor_data = get_clean_factor_and_forward_returns(factor=factor,
                                                   prices=trade_prices,
                                                   quantiles=5,
                                                   periods=(1, 5, 10, 21)).
 ↪sort_index()
factor_data.info()
```

```
Dropped 30.0% entries from factor data: 0.4% in forward returns computation and
29.6% in binning phase (set max_loss=0 to see potentially suppressed
Exceptions).
max_loss is 35.0%, not exceeded: OK!
<class 'pandas.core.frame.DataFrame'>
MultiIndex: 625157 entries, (2011-04-27 00:00:00+00:00, A) to (2017-12-28
00:00:00+00:00, ZTS)
Data columns (total 6 columns):
1D                 625157 non-null float64
5D                 625157 non-null float64
10D                625157 non-null float64
21D                625157 non-null float64
factor             625157 non-null float32
factor_quantile    625157 non-null float64
dtypes: float32(1), float64(5)
memory usage: 28.6+ MB
```

[19]:
```
create_summary_tear_sheet(factor_data);
```

Quantiles Statistics

factor_quantile	min	max	mean	std	count	count %
1.0	-0.005168	0.001667	0.000644	0.000625	125475	20.070958
2.0	-0.000586	0.001673	0.000684	0.000610	124853	19.971463
3.0	-0.000584	0.001677	0.000702	0.000606	124813	19.965065
4.0	-0.000582	0.001682	0.000715	0.000601	125301	20.043125
5.0	-0.000579	0.048656	0.000758	0.000784	124715	19.949389

Returns Analysis

	1D	5D	10D	21D
Ann. alpha	-0.092	-0.123	-0.089	-0.059
beta	0.053	0.107	0.130	0.253
Mean Period Wise Return Top Quantile (bps)	0.965	0.106	-0.185	-0.174
Mean Period Wise Return Bottom Quantile (bps)	-1.500	-1.164	-1.533	-1.526
Mean Period Wise Spread (bps)	2.465	1.244	1.315	1.322

Information Analysis

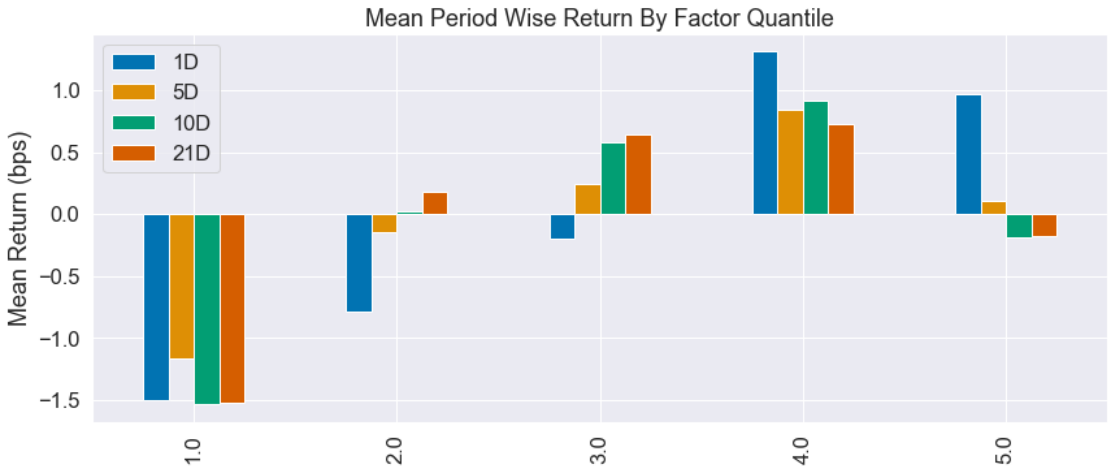
	1D	5D	10D	21D
IC Mean	0.008	0.011	0.015	0.014
IC Std.	0.143	0.139	0.137	0.128
Risk-Adjusted IC	0.056	0.079	0.108	0.106
t-stat(IC)	1.926	2.697	3.675	3.610
p-value(IC)	0.054	0.007	0.000	0.000
IC Skew	0.062	0.092	0.046	0.416
IC Kurtosis	0.351	0.730	0.265	0.631

Turnover Analysis

	1D	5D	10D	21D
Quantile 1 Mean Turnover	0.236	0.480	0.604	0.708
Quantile 2 Mean Turnover	0.462	0.688	0.746	0.776
Quantile 3 Mean Turnover	0.497	0.709	0.761	0.783
Quantile 4 Mean Turnover	0.437	0.669	0.735	0.775
Quantile 5 Mean Turnover	0.218	0.455	0.582	0.691

	1D	5D	10D	21D
Mean Factor Rank Autocorrelation	0.876	0.588	0.388	0.194

<Figure size 432x288 with 0 Axes>



1.2.1 Load zipline extensions

Only need this in notebook to find bundle.

[20]:
```
load_extensions(default=True,
                extensions=[],
                strict=True,
                environ=None)
```

[21]:
```
log_handler = StderrHandler(format_string='[{record.time:%Y-%m-%d %H:%M:%S.%f}]:
↪ ' +
                            '{record.level_name}: {record.func_name}: {record.
↪message}',
                            level=WARNING)
log_handler.push_application()
log = Logger('Algorithm')
```

1.3 Algo Params

[22]:
```
N_LONGS = 25
N_SHORTS = 25
MIN_POSITIONS = 10
```

1.4 Load Data

1.4.1 Quandl Wiki Bundel

[23]:
```
bundle_data = bundles.load('quandl')
```

1.4.2 ML Predictions

[26]:
```
def load_predictions(bundle):
    predictions = (pd.read_hdf(results_path / 'predictions.h5', 'predictions')
                   .iloc[:, :4]
                   .mean(1)
                   .to_frame('prediction'))
    tickers = predictions.index.get_level_values('symbol').unique().tolist()

    assets = bundle.asset_finder.lookup_symbols(tickers, as_of_date=None)
    predicted_sids = pd.Int64Index([asset.sid for asset in assets])
    ticker_map = dict(zip(tickers, predicted_sids))

    return (predictions
            .unstack('symbol')
            .rename(columns=ticker_map)
            .prediction
            .tz_localize('UTC')), assets
```

[27]:
```
predictions, assets = load_predictions(bundle_data)
```

[28]:
```
predictions.info()
```

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 1680 entries, 2011-04-27 to 2017-12-28
Columns: 600 entries, 0 to 3197
dtypes: float32(600)
memory usage: 3.9 MB
```

1.4.3 Define Custom Dataset

[29]:
```
class SignalData(DataSet):
    predictions = Column(dtype=float)
    domain = US_EQUITIES
```

1.4.4 Define Pipeline Loaders

[30]:
```
signal_loader = {SignalData.predictions:
                     DataFrameLoader(SignalData.predictions, predictions)}
```

1.5 Pipeline Setup

1.5.1 Custom ML Factor

[31]:
```
class MLSignal(CustomFactor):
    """Converting signals to Factor
        so we can rank and filter in Pipeline"""
    inputs = [SignalData.predictions]
```

```
    window_length = 1

    def compute(self, today, assets, out, predictions):
        out[:] = predictions
```

1.5.2 Create Pipeline

[32]:
```
def compute_signals():
    signals = MLSignal()
    return Pipeline(columns={
        'longs' : signals.top(N_LONGS),
        'shorts': signals.bottom(N_SHORTS)},
            screen=StaticAssets(assets))
```

1.6 Initialize Algorithm

[33]:
```
def initialize(context):
    """
    Called once at the start of the algorithm.
    """
    context.longs = context.shorts = None
    set_slippage(slippage.FixedSlippage(spread=0.00))
#     set_commission(commission.PerShare(cost=0.001, min_trade_cost=0))

    schedule_function(rebalance,
                      date_rules.every_day(),
#                       date_rules.week_start(),
                      time_rules.market_open(hours=1, minutes=30))

    schedule_function(record_vars,
                      date_rules.every_day(),
                      time_rules.market_close())

    pipeline = compute_signals()
    attach_pipeline(pipeline, 'signals')
```

1.6.1 Get daily Pipeline results

[34]:
```
def before_trading_start(context, data):
    """
    Called every day before market open.
    """
    output = pipeline_output('signals')
    longs = pipeline_output('signals').longs.astype(int)
    shorts = pipeline_output('signals').shorts.astype(int)
    holdings = context.portfolio.positions.keys()
```

```
    if longs.sum() > MIN_POSITIONS and shorts.sum() > MIN_POSITIONS:
        context.longs = longs[longs!=0].index
        context.shorts = shorts[shorts!=0].index
        context.divest = holdings - set(context.longs) - set(context.shorts)
    else:
        context.longs = context.shorts = pd.Index([])
        context.divest = set(holdings)
```

1.7 Define Rebalancing Logic

[35]:
```
def rebalance(context, data):
    """
    Execute orders according to schedule_function() date & time rules.
    """

    for symbol, open_orders in get_open_orders().items():
        for open_order in open_orders:
            cancel_order(open_order)

    for stock in context.divest:
        order_target(stock, target=0)

#     log.warning('{} {:,.0f}'.format(len(context.portfolio.positions), context.
 ↪portfolio.portfolio_value))
    if not (context.longs.empty and context.shorts.empty):
        for stock in context.shorts:
            order_target_percent(stock, -1 / len(context.shorts) / 2)
        for stock in context.longs:
            order_target_percent(stock, 1 / len(context.longs))
```

1.8 Record Data Points

[36]:
```
def record_vars(context, data):
    """
    Plot variables at the end of each day.
    """
    record(leverage=context.account.leverage,
           longs=context.longs,
           shorts=context.shorts)
```

1.9 Run Algorithm

[37]:
```
dates = predictions.index.get_level_values('date')
start_date, end_date = dates.min(), dates.max()
```

[38]:
```
print('Start: {}\nEnd:   {}'.format(start_date.date(), end_date.date()))
```

```
Start: 2011-04-27
End:   2017-12-28
```

[39]:
```
start = time()
results = run_algorithm(start=start_date,
                        end=end_date,
                        initialize=initialize,
                        before_trading_start=before_trading_start,
                        capital_base=1e5,
                        data_frequency='daily',
                        bundle='quandl',
                        custom_loader=signal_loader)  # need to modify zipline

print('Duration: {:.2f}s'.format(time() - start))
```

```
[2020-06-22 18:19:33.792272]: WARNING: ensure_benchmark_data: Still don't have
expected benchmark data for 'SPY' from 1989-12-29 00:00:00+00:00 to 2020-06-22
00:00:00+00:00 after redownload!
[2020-06-22 18:20:34.915988]: WARNING: _can_order_asset: Cannot place order for
ACE, as it has de-listed. Any existing positions for this asset will be
liquidated on 2016-01-15 00:00:00+00:00.
[2020-06-22 18:20:43.103276]: WARNING: _can_order_asset: Cannot place order for
RAX, as it has de-listed. Any existing positions for this asset will be
liquidated on 2016-11-03 00:00:00+00:00.

Duration: 81.91s
```

1.10 PyFolio Analysis

[40]:
```
returns, positions, transactions = pf.utils.
 ↪extract_rets_pos_txn_from_zipline(results)
```

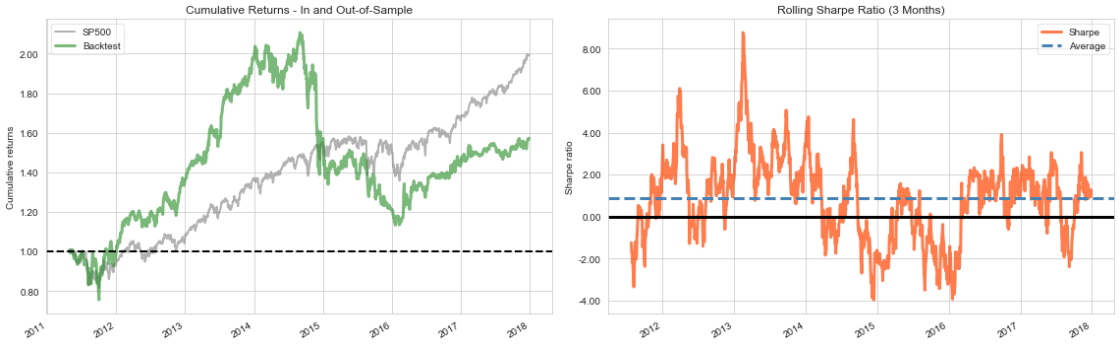
[44]:
```
benchmark = web.DataReader('SP500', 'fred', '2010', '2018').squeeze()
benchmark = benchmark.pct_change().tz_localize('UTC')
```

1.10.1 Custom Plots

[45]:
```
LIVE_DATE = '2018-01-01'
```

[47]:
```
fig, axes = plt.subplots(ncols=2, figsize=(16, 5))
plot_rolling_returns(returns,
```

```
                      factor_returns=benchmark,
                      live_start_date=LIVE_DATE,
                      logy=False,
                      cone_std=2,
                      legend_loc='best',
                      volatility_match=False,
                      cone_function=forecast_cone_bootstrap,
                      ax=axes[0])
plot_rolling_sharpe(returns, ax=axes[1], rolling_window=63)
axes[0].set_title('Cumulative Returns - In and Out-of-Sample')
axes[1].set_title('Rolling Sharpe Ratio (3 Months)')
fig.tight_layout()
fig.savefig((results_path / 'pyfolio_out_of_sample').as_posix(), dpi=300)
```



1.10.2 Tear Sheets

[48]:
```
pf.create_full_tear_sheet(returns,
                          positions=positions,
                          transactions=transactions,
                          benchmark_rets=benchmark,
                          round_trips=True)
```

```
<IPython.core.display.HTML object>
```

```
<IPython.core.display.HTML object>
```

```
<IPython.core.display.HTML object>
```

```
<IPython.core.display.HTML object>
```

```
<IPython.core.display.HTML object>
```

```
<IPython.core.display.HTML object>
```

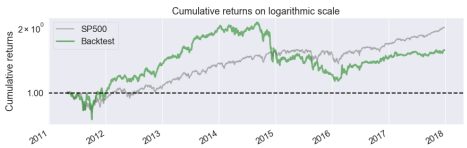
```
<IPython.core.display.HTML object>
```

```
<IPython.core.display.HTML object>
```

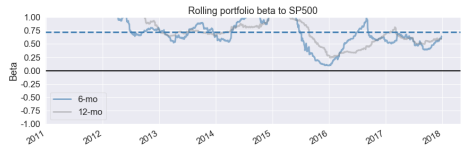
```
<IPython.core.display.HTML object>
<IPython.core.display.HTML object>
<IPython.core.display.HTML object>
<IPython.core.display.HTML object>
```


Cumulative returns
SP500
Backtest
Cumulative returns
2.00
1.80
1.60
1.40
1.20
1.00
0.80
2011
2012
2013
2014
2015
2016
2017
2018

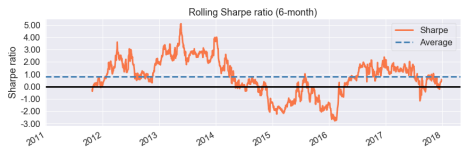

Cumulative returns volatility matched to benchmark
Cumulative returns
2.00
1.80
1.60
1.40
1.20
1.00
0.80
2011
2012
2013
2014
2015
2016
2017
2018


Cumulative returns on logarithmic scale
SP500
Backtest
Cumulative returns
2 × 10^0
1.00
2011
2012
2013
2014
2015
2016
2017
2018

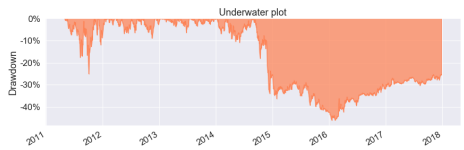

Returns
Returns
0.075
0.050
0.025
0.000
−0.025
−0.050
−0.075
−0.100
2011
2012
2013
2014
2015
2016
2017
2018


Rolling portfolio beta to SP500
Beta
1.00
0.75
0.50
0.25
0.00
-0.25
-0.50
-0.75
-1.00
6-mo
12-mo
2011
2012
2013
2014
2015
2016
2017
2018

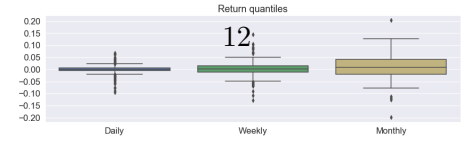

Rolling volatility (6-month)
Volatility
Benchmark volatility
Average volatility
Volatility
0.40
0.35
0.30
0.25
0.20
0.15
0.10
0.05
0.00
2011
2012
2013
2014
2015
2016
2017
2018

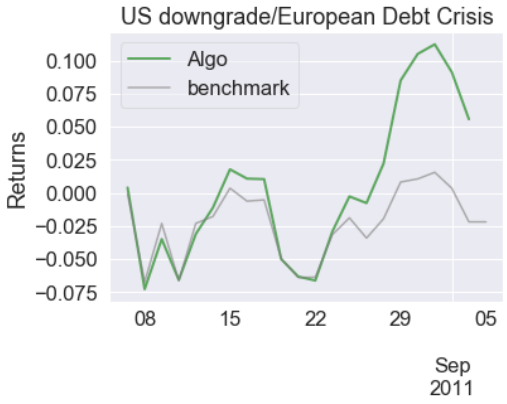

Rolling Sharpe ratio (6-month)
Sharpe
Average
Sharpe ratio
5.00
4.00
3.00
2.00
1.00
0.00
-1.00
-2.00
-3.00
2011
2012
2013
2014
2015
2016
2017
2018

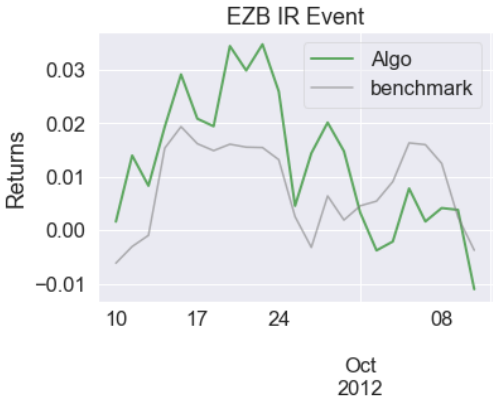

Top 5 drawdown periods
Portfolio
Cumulative returns
2.00
1.80
1.60
1.40
1.20
1.00
0.80
2011
2012
2013
2014
2015
2016
2017
2018

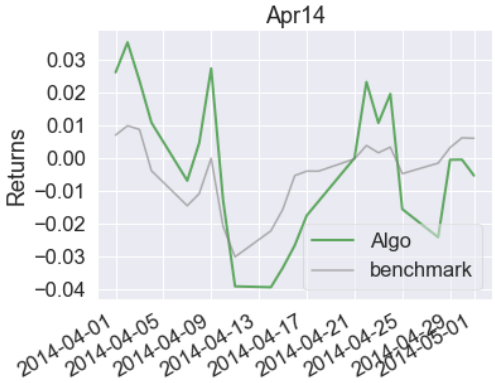

Underwater plot
Drawdown
0%
-10%
-20%
-30%
-40%
2011
2012
2013
2014
2015
2016
2017
2018


Monthly returns (%)
Year
Month
Annual returns
Year
Returns
Mean
Distribution of monthly returns
Number of months
Returns
Mean

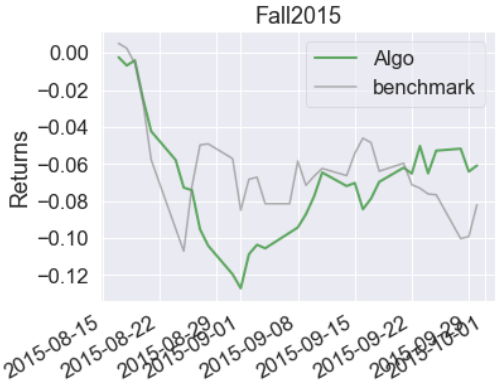

Return quantiles
0.20
0.15
0.10
0.05
0.00
−0.05
−0.10
−0.15
−0.20
Daily
Weekly
Monthly

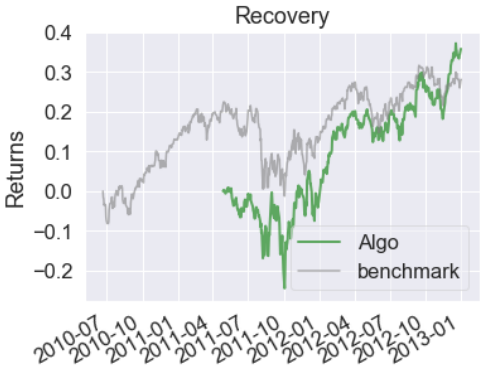

US downgrade/European Debt Crisis
Algo
benchmark
Returns
0.100
0.075
0.050
0.025
0.000
−0.025
−0.050
−0.075
08
15
22
29
05
Sep
2011

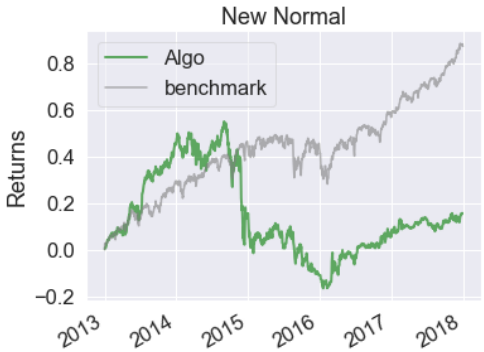

EZB IR Event
Algo
benchmark
Returns
0.03
0.02
0.01
0.00
−0.01
10
17
24
08
Oct
2012

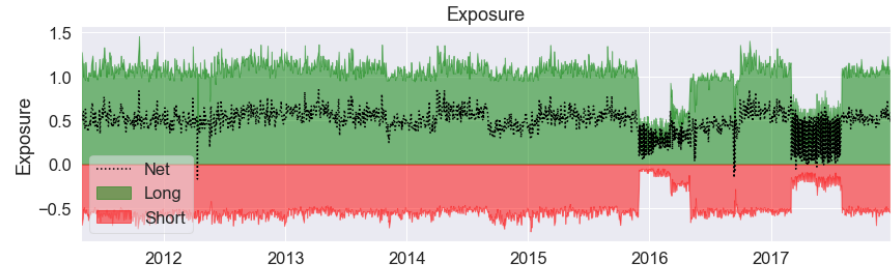

Apr14
Algo
benchmark
Returns
0.03
0.02
0.01
0.00
−0.01
−0.02
−0.03
−0.04
2014-04-01
2014-04-05
2014-04-09
2014-04-13
2014-04-17
2014-04-21
2014-04-25
2014-04-29
2014-05-01

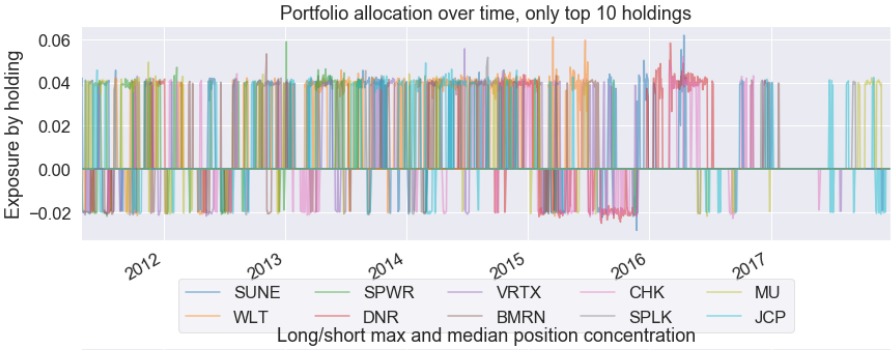

Oct14
Algo
benchmark
Returns
0.02
0.00
−0.02
−0.04
−0.06
−0.08
−0.10
06
13
20
27
Oct
2014
Nov

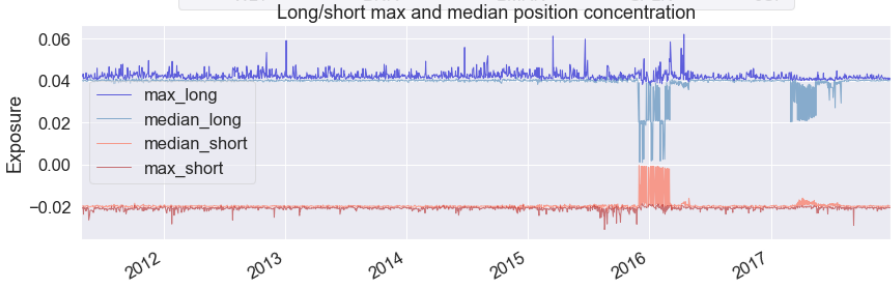

Fall2015
Algo
benchmark
Returns
0.00
−0.02
−0.04
−0.06
−0.08
−0.10
−0.12
2015-08-15
2015-08-22
2015-08-29
2015-09-01
2015-09-08
2015-09-15
2015-09-22
2015-09-29
2015-10-01

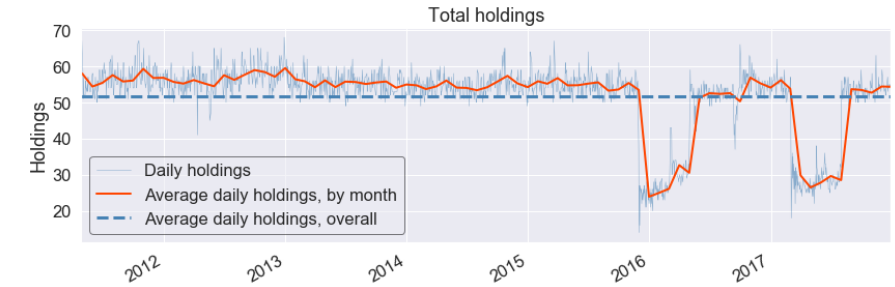

Recovery
Algo
benchmark
Returns
0.4
0.3
0.2
0.1
0.0
−0.1
−0.2
2010-07
2010-10
2011-01
2011-04
2011-07
2011-10
2012-01
2012-04
2012-07
2012-10
2013-01

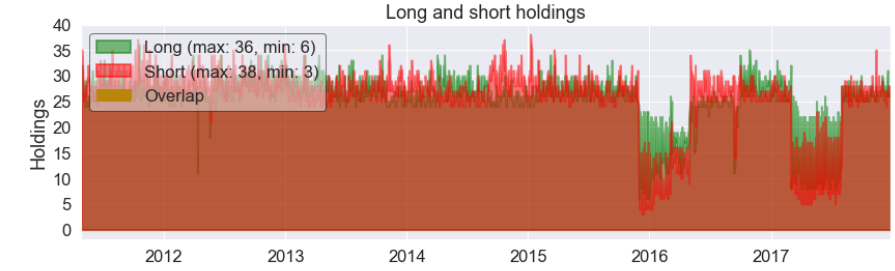

New Normal
Algo
benchmark
Returns
0.8
0.6
0.4
0.2
0.0
−0.2
2013
2014
2015
2016
2017
2018

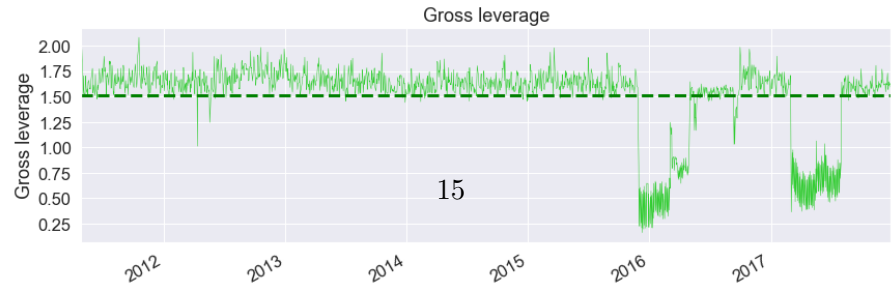

Exposure
Exposure
1.5
1.0
0.5
0.0
−0.5
Net
Long
Short
2012
2013
2014
2015
2016
2017

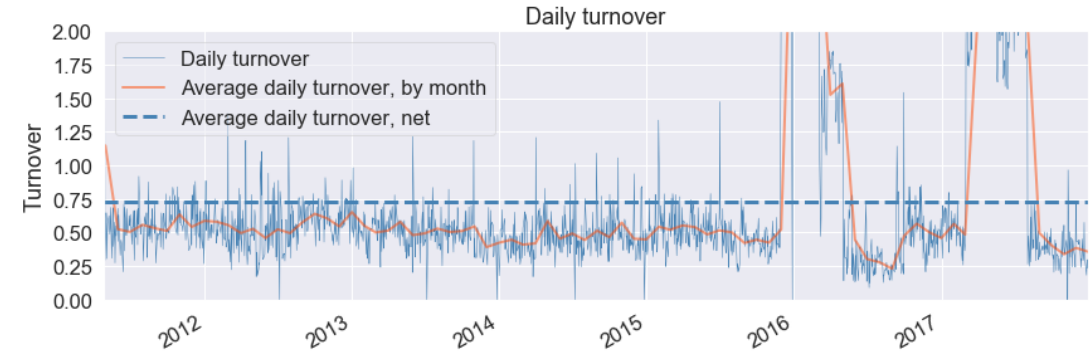

Portfolio allocation over time, only top 10 holdings
Exposure by holding
0.06
0.04
0.02
0.00
−0.02
2012
2013
2014
2015
2016
2017
SUNE
SPWR
VRTX
CHK
MU
WLT
DNR
BMRN
SPLK
JCP

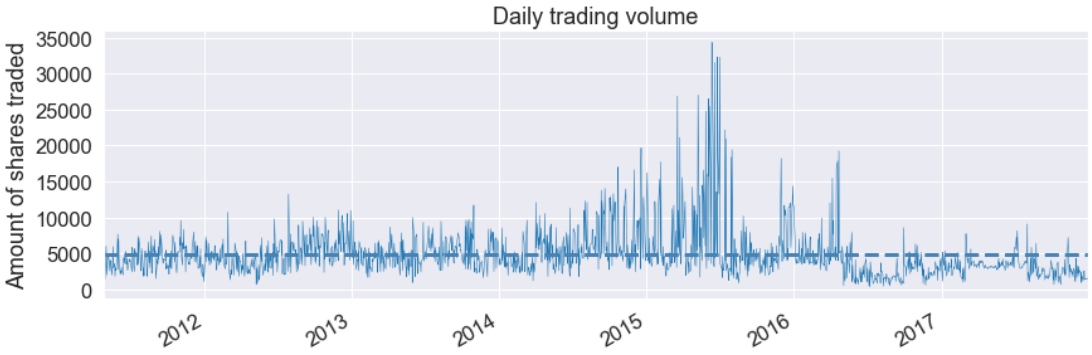

Long/short max and median position concentration
Exposure
0.06
0.04
0.02
0.00
−0.02
max_long
median_long
median_short
max_short
2012
2013
2014
2015
2016
2017

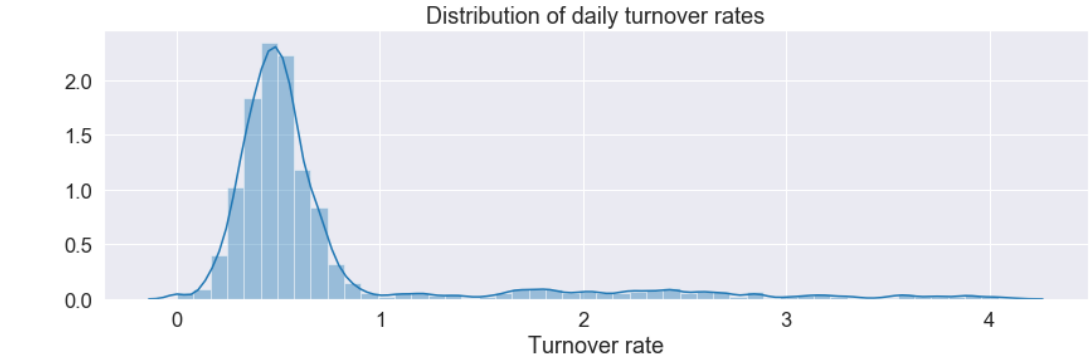

Total holdings
Holdings
70
60
50
40
30
20
Daily holdings
Average daily holdings, by month
Average daily holdings, overall
2012
2013
2014
2015
2016
2017

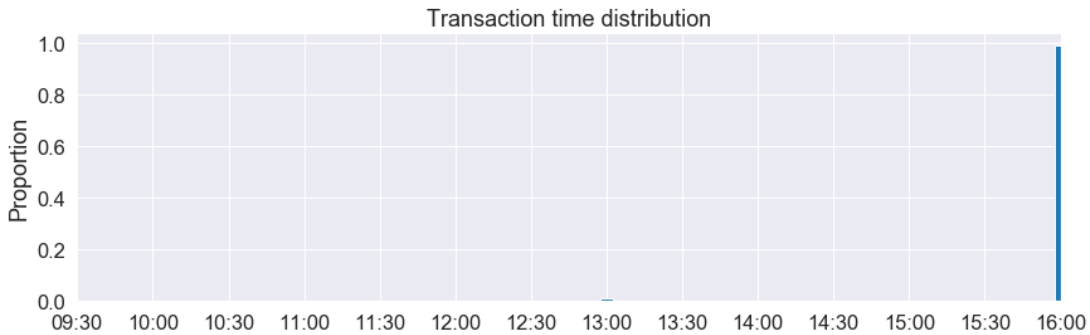

Long and short holdings
Holdings
40
35
30
25
20
15
10
5
0
Long (max: 36, min: 6)
Short (max: 38, min: 3)
Overlap
2012
2013
2014
2015
2016
2017

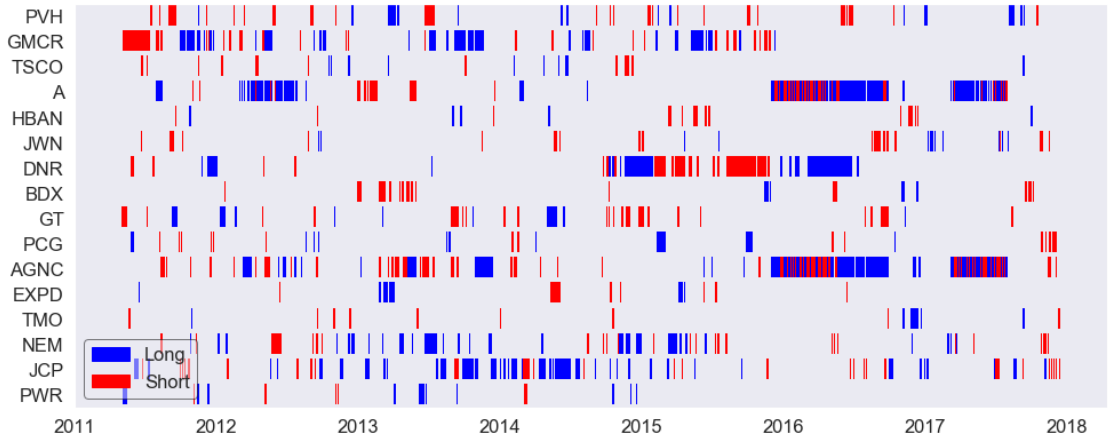

Gross leverage
Gross leverage
2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
2012
2013
2014
2015
2016
2017

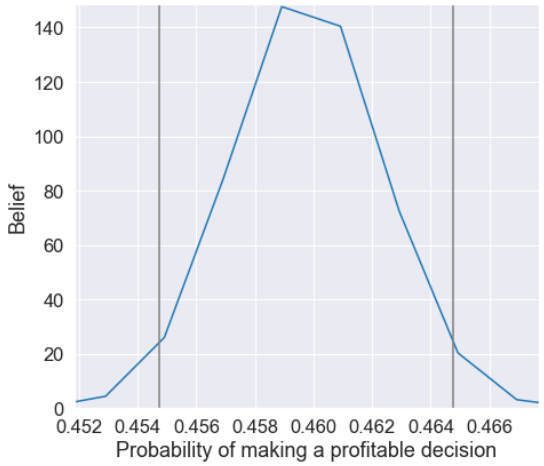

Daily turnover
Daily turnover
Average daily turnover, by month
Average daily turnover, net
Turnover
2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
2012
2013
2014
2015
2016
2017

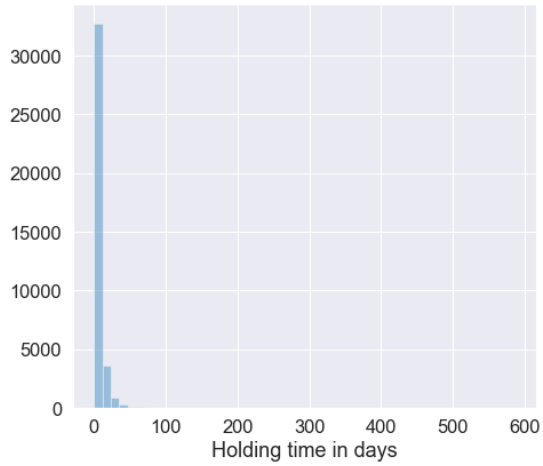

Daily trading volume
Amount of shares traded
35000
30000
25000
20000
15000
10000
5000
0
2012
2013
2014
2015
2016
2017

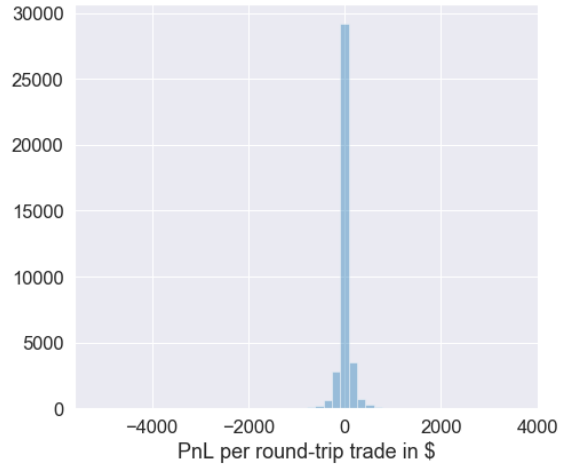

Distribution of daily turnover rates
2.0
1.5
1.0
0.5
0.0
0
1
2
3
4
Turnover rate

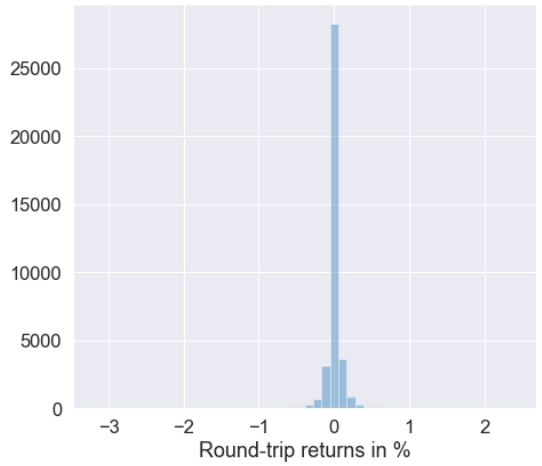

Transaction time distribution
Proportion
1.0
0.8
0.6
0.4
0.2
0.0
09:30
10:00
10:30
11:00
11:30
12:00
12:30
13:00
13:30
14:00
14:30
15:00
15:30
16:00


PVH
GMCR
TSCO
A
HBAN
JWN
DNR
BDX
GT
PCG
AGNC
EXPD
TMO
NEM
JCP
PWR
Long
Short
2011
2012
2013
2014
2015
2016
2017
2018


Belief
Probability of making a profitable decision


Holding time in days


PnL per round-trip trade in $


Round-trip returns in %

[ ]: