09_backtesting_with_zipline

September 29, 2021

1 Long-Short Strategy, Part 6: Backtesting with Zipline

In this section, we'll start designing, implementing, and evaluating a trading strategy for US equities driven by daily return forecasts produced by gradient boosting models.

As in the previous examples, we'll lay out a framework and build a specific example that you can adapt to run your own experiments. There are numerous aspects that you can vary, from the asset class and investment universe to more granular aspects like the features, holding period, or trading rules. See, for example, the **Alpha Factor Library** in the Appendix for numerous additional features.

We'll keep the trading strategy simple and only use a single ML signal; a real-life application will likely use multiple signals from different sources, such as complementary ML models trained on different datasets or with different lookahead or lookback periods. It would also use sophisticated risk management, from simple stop-loss to value-at-risk analysis.

Six notebooks cover our workflow sequence:

1. preparing_the_model_data: we engineer a few simple features from the Quandl Wiki data
2. trading_signals_with_lightgbm_and_catboost: we tune hyperparameters for LightGBM and CatBoost to select a model, using 2015/16 as our validation period.
3. evaluate_trading_signals: we compare the cross-validation performance using various metrics to select the best model.
4. model_interpretation: we take a closer look at the drivers behind the best model's predictions.
5. making_out_of_sample_predictions : we predict returns for our out-of-sample period 2017.
6. `backtesting_with_zipline` (this noteboook): evaluate the historical performance of a long-short strategy based on our predictive signals using Zipline.

1.1 Imports & Settings

[1]:
```
from collections import defaultdict
from time import time
import warnings

import numpy as np
import pandas as pd

import matplotlib.pyplot as plt
import pandas_datareader.data as web
from logbook import Logger, StderrHandler, INFO, WARNING
```

```python
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
```

[2]:
```python
# optional; not pre-installed; see above
# import seaborn as sns
# sns.set_style('darkgrid')
```

[3]:
```python
warnings.filterwarnings('ignore')
np.random.seed(42)
```

1.1.1 Load zipline extensions

Only need this in notebook to find bundle.

[4]:
```python
load_extensions(default=True,
                extensions=[],
                strict=True,
                environ=None)
```

[5]:
```python
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')
```

1.2 Algo Params

[6]:
```
N_LONGS = 25
N_SHORTS = 25
MIN_POSITIONS = 20
```

1.3 Load Data

1.3.1 Quandl Wiki Bundle

Requires running `zipline ingest` (see installation instructions and Chapter 8). If you haven't done so yet (but have provided your QUANDL API KEY when launching Docker), uncomment and run the following cell:

[7]:
```
# !zipline ingest
```

[8]:
```
bundle_data = bundles.load('quandl')
```

1.3.2 ML Predictions

If you run into difficulties reading the predictions, run the following to upgrade `tables` (source).

[9]:
```
# !pip install --upgrade tables
```

[10]:
```
def load_predictions(bundle):
    predictions = (pd.read_hdf('data/predictions.h5', 'lgb/train/01')
                   .append(pd.read_hdf('data/predictions.h5', 'lgb/test/01').
 ↪drop('y_test', axis=1)))
    predictions = (predictions.loc[~predictions.index.duplicated()]
                   .iloc[:, :10]
                   .mean(1)
                   .sort_index()
                   .dropna()
                  .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
```

[11]:
```
predictions, assets = load_predictions(bundle_data)
```

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

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 753 entries, 2015-01-02 00:00:00+00:00 to 2017-12-28
00:00:00+00:00
Columns: 995 entries, 0 to 3188
dtypes: float64(995)
memory usage: 5.7 MB
```

1.3.3 Define Custom Dataset

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

1.3.4 Define Pipeline Loaders

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

1.4 Pipeline Setup

1.4.1 Custom ML Factor

```
[15]: 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
```

1.4.2 Create Pipeline

```
[16]: def compute_signals():
          signals = MLSignal()
          return Pipeline(columns={
              'longs' : signals.top(N_LONGS, mask=signals > 0),
              'shorts': signals.bottom(N_SHORTS, mask=signals < 0)},
                  screen=StaticAssets(assets))
```

1.5 Initialize Algorithm

[17]:
```
def initialize(context):
    """
    Called once at the start of the algorithm.
    """
    context.n_longs = N_LONGS
    context.n_shorts = N_SHORTS
    context.min_positions = MIN_POSITIONS
    context.universe = assets
    context.trades = pd.Series()
    context.longs = context.shorts = 0

    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')
```

1.5.1 Get daily Pipeline results

[18]:
```
def before_trading_start(context, data):
    """
    Called every day before market open.
    """
    output = pipeline_output('signals')
    df = (output['longs'].astype(int)
                      .append(output['shorts'].astype(int).mul(-1)))

    holdings = df[df!=0]
    other = df[df==0]
    other = other[~other.index.isin(holdings.index) & ~other.index.duplicated()]
    context.trades = holdings.append(other)
    assert len(context.trades.index.unique()) == len(context.trades)
```

1.6 Define Rebalancing Logic

[19]:
```python
def rebalance(context, data):
    """
    Execute orders according to schedule_function() date & time rules.
    """
    trades = defaultdict(list)
    for symbol, open_orders in get_open_orders().items():
        for open_order in open_orders:
            cancel_order(open_order)

    positions = context.portfolio.positions
    s=pd.Series({s:v.amount*v.last_sale_price for s, v in positions.items()}).
 ↪sort_values(ascending=False)
    for stock, trade in context.trades.items():
        if trade == 0:
            order_target(stock, target=0)
        else:
            trades[trade].append(stock)

    context.longs, context.shorts = len(trades[1]), len(trades[-1])
#     log.warning('{} {:,.0f}'.format(len(positions), context.portfolio.
 ↪portfolio_value))
    if context.longs > context.min_positions and context.shorts > context.
 ↪min_positions:
        for stock in trades[-1]:
            order_target_percent(stock, -1 / context.shorts)
        for stock in trades[1]:
            order_target_percent(stock, 1 / context.longs)
    else:
        for stock in trades[-1] + trades[1]:
            if stock in positions:
                order_target(stock, 0)
```

1.7 Record Data Points

[20]:
```python
def record_vars(context, data):
    """
    Plot variables at the end of each day.
    """
    record(leverage=context.account.leverage,
           longs=context.longs,
           shorts=context.shorts)
```

1.8 Run Algorithm

We backtest our strategy during the (in-sample) validation and out-of-sample test period:

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

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

```
Start: 2015-01-02
End:   2017-12-28
```

```
[23]: 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))
```

```
Duration: 92.61s
```

1.9 PyFolio Analysis

To visualize the out-of-sample performance, we pass '2017-01-01' as start date for the `live_start_date`:

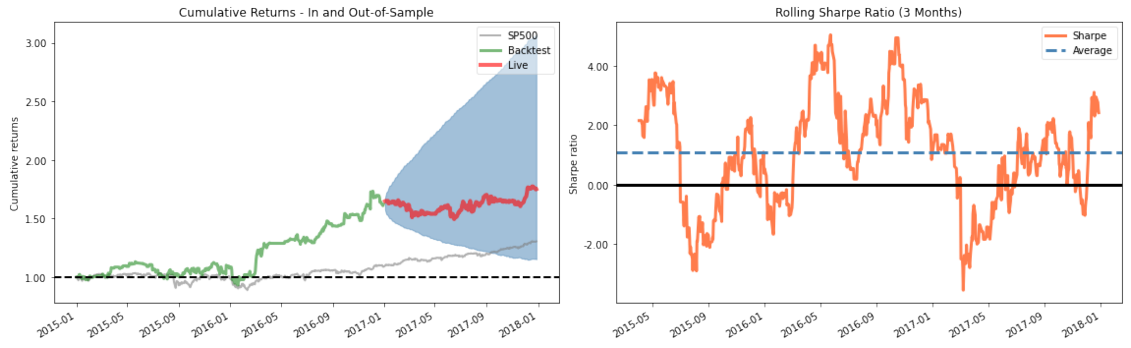
```
[25]: returns, positions, transactions = pf.utils.
       ↪extract_rets_pos_txn_from_zipline(results)
```

```
[26]: benchmark = web.DataReader('SP500', 'fred', '2014', '2018').squeeze()
      benchmark = benchmark.pct_change().tz_localize('UTC')
```

1.9.1 Custom Plots

```
[27]: fig, axes = plt.subplots(ncols=2, figsize=(16, 5))
      plot_rolling_returns(returns,
                           factor_returns=benchmark,
                           live_start_date='2017-01-01',
                           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)')
```

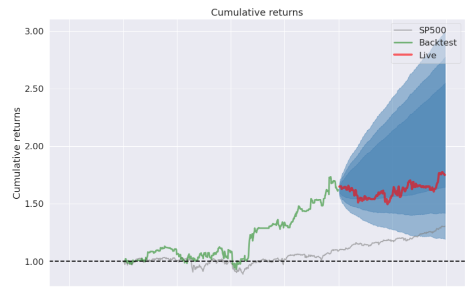
```
fig.tight_layout();
```

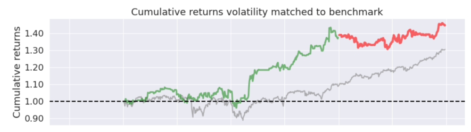


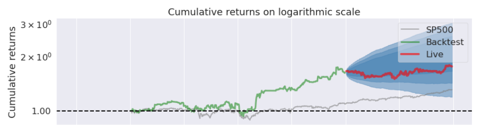
1.9.2 Tear Sheets

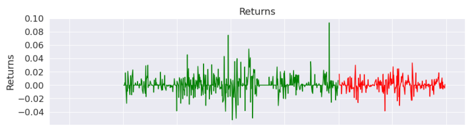
[28]:
```
pf.create_full_tear_sheet(returns,
                          positions=positions,
                          transactions=transactions,
                          benchmark_rets=benchmark,
                          live_start_date='2017-01-01',
                          round_trips=True)
```

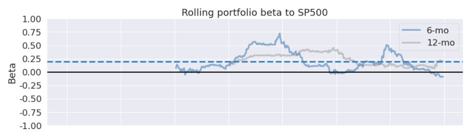
```
<IPython.core.display.HTML object>

<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>
```

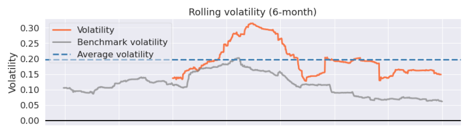

Cumulative returns
SP500
Backtest
Live
Cumulative returns
3.00
2.50
2.00
1.50
1.00

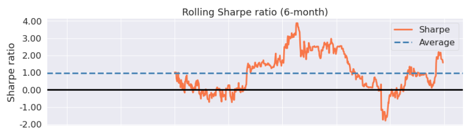

Cumulative returns volatility matched to benchmark
Cumulative returns
1.40
1.30
1.20
1.10
1.00
0.90

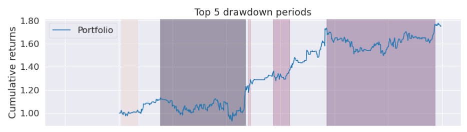

Cumulative returns on logarithmic scale
SP500
Backtest
Live
Cumulative returns
3 × 10^0
2 × 10^0
1.00

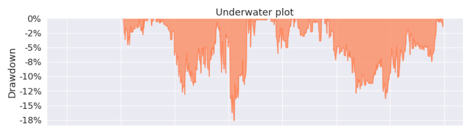

Returns
Returns
0.10
0.08
0.06
0.04
0.02
0.00
−0.02
−0.04

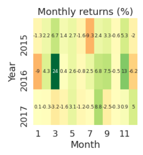

Rolling portfolio beta to SP500
6-mo
12-mo
Beta
1.00
0.75
0.50
0.25
0.00
-0.25
-0.50
-0.75
-1.00

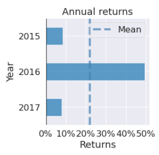

Rolling volatility (6-month)
Volatility
Benchmark volatility
Average volatility
Volatility
0.30
0.25
0.20
0.15
0.10
0.05
0.00

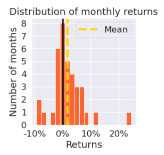

Rolling Sharpe ratio (6-month)
Sharpe
Average
Sharpe ratio
4.00
3.00
2.00
1.00
0.00
-1.00
-2.00

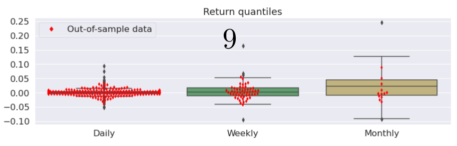

Top 5 drawdown periods
Portfolio
Cumulative returns
1.80
1.60
1.40
1.20
1.00

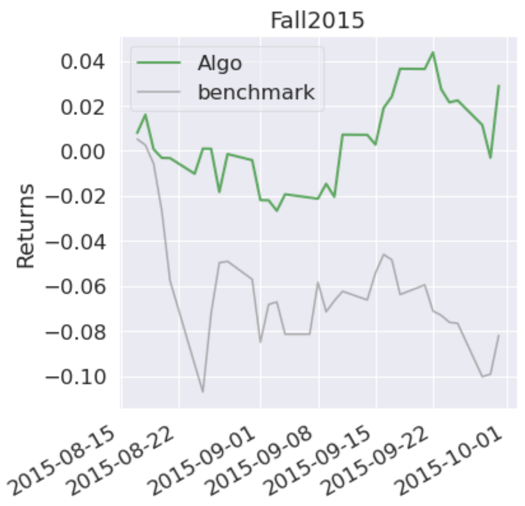

Underwater plot
Drawdown
0%
-2%
-5%
-8%
-10%
-12%
-15%
-18%

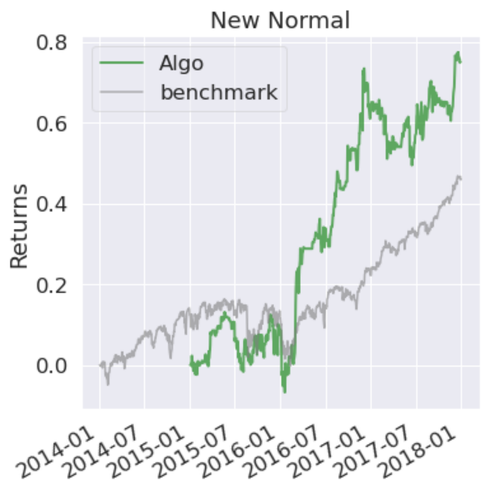

Monthly returns (%)
Year
2015
2016
2017
Month
1
3
5
7
9
11

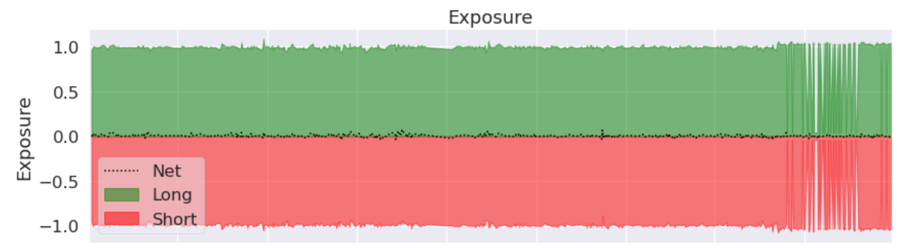

Annual returns
Mean
Year
2015
2016
2017
0%
10%
20%
30%
40%
50%
Returns

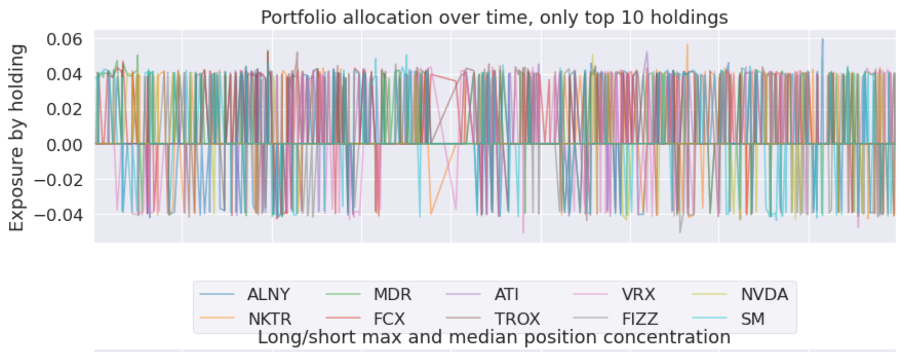

Distribution of monthly returns
Mean
Number of months
Returns
-10%
0%
10%
20%

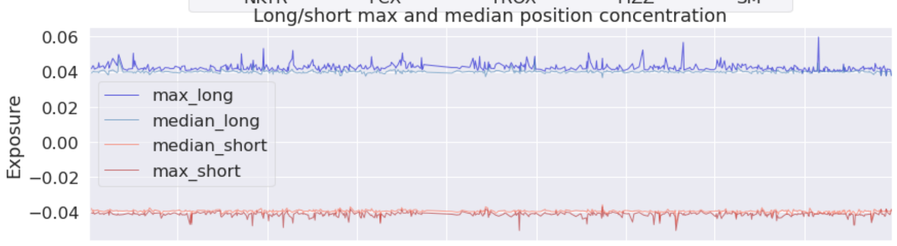

Return quantiles
Out-of-sample data
0.25
0.20
0.15
0.10
0.05
0.00
-0.05
-0.10
Daily
Weekly
Monthly

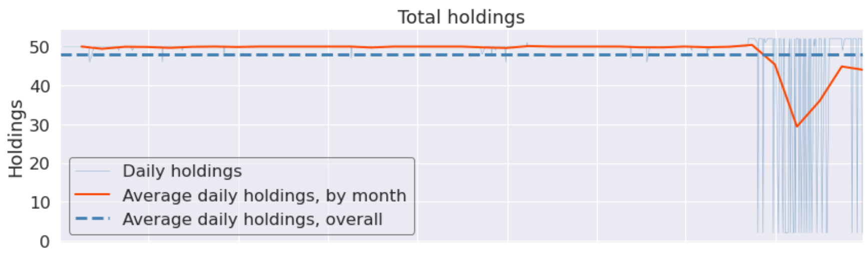

Fall2015
Algo
benchmark
Returns
0.04
0.02
0.00
−0.02
−0.04
−0.06
−0.08
−0.10
2015-08-15
2015-08-22
2015-09-01
2015-09-08
2015-09-15
2015-09-22
2015-10-01

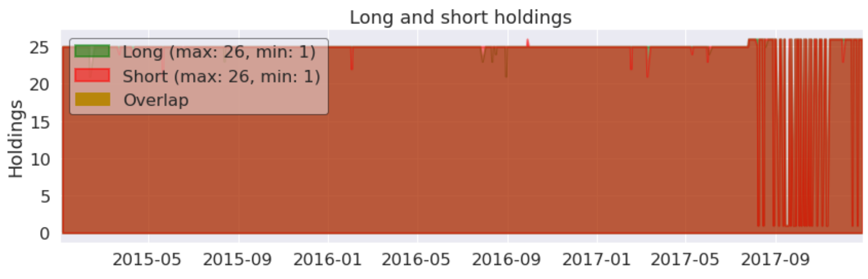

New Normal
Algo
benchmark
Returns
0.8
0.6
0.4
0.2
0.0
2014-01
2014-07
2015-01
2015-07
2016-01
2016-07
2017-01
2017-07
2018-01

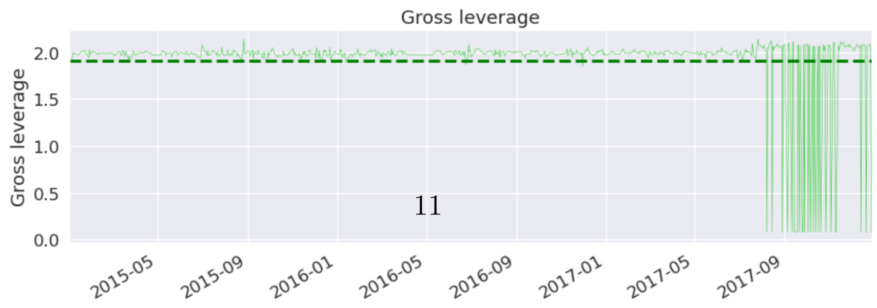

Exposure
Exposure
1.0
0.5
0.0
−0.5
−1.0
Net
Long
Short

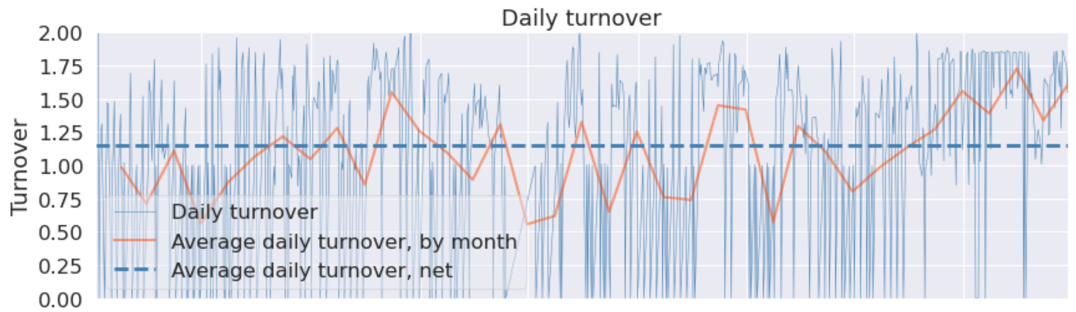

Portfolio allocation over time, only top 10 holdings
Exposure by holding
0.06
0.04
0.02
0.00
−0.02
−0.04
ALNY
NKTR
MDR
FCX
ATI
TROX
VRX
FIZZ
NVDA
SM

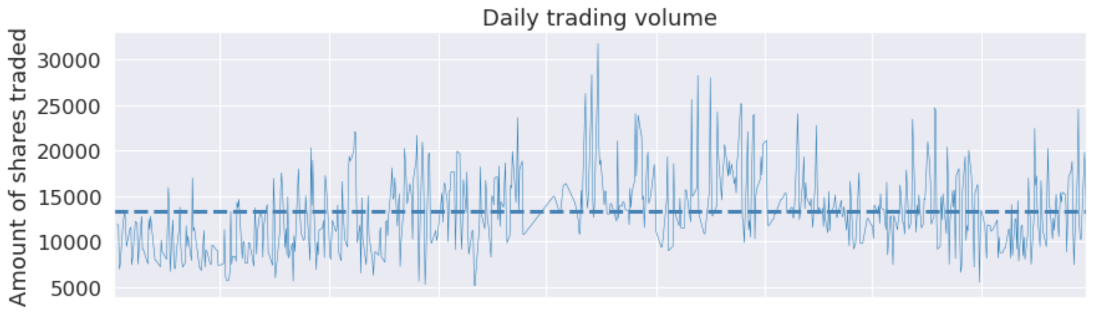

Long/short max and median position concentration
Exposure
0.06
0.04
0.02
0.00
−0.02
−0.04
max_long
median_long
median_short
max_short

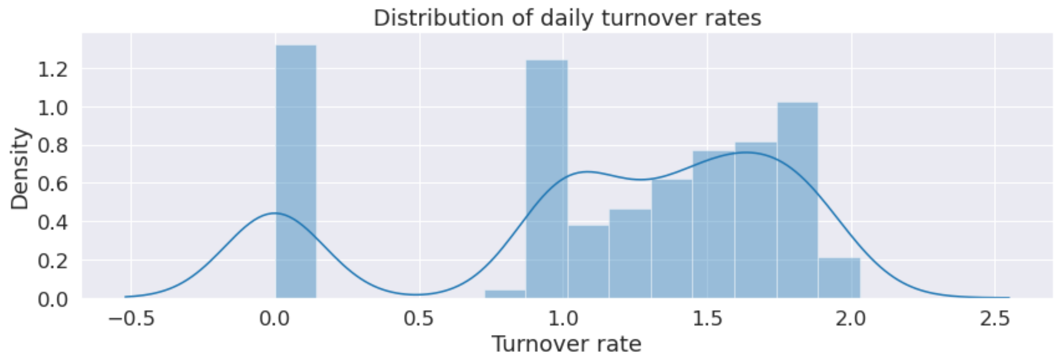

Total holdings
Holdings
50
40
30
20
10
0
Daily holdings
Average daily holdings, by month
Average daily holdings, overall

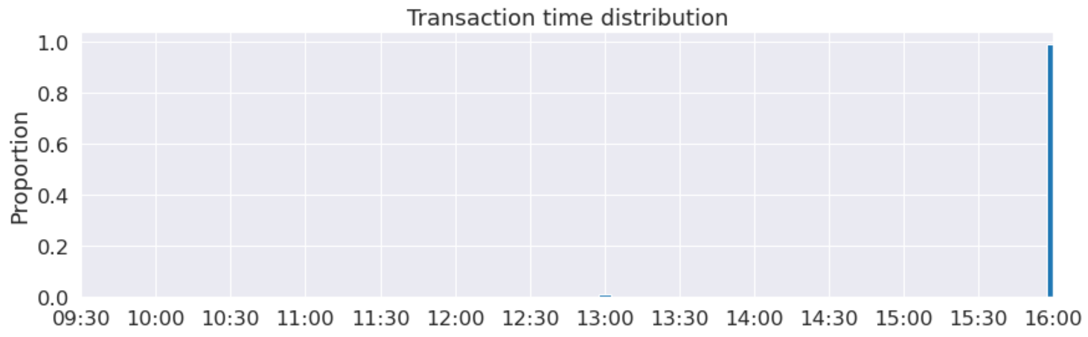

Long and short holdings
Holdings
25
20
15
10
5
0
Long (max: 26, min: 1)
Short (max: 26, min: 1)
Overlap
2015-05
2015-09
2016-01
2016-05
2016-09
2017-01
2017-05
2017-09

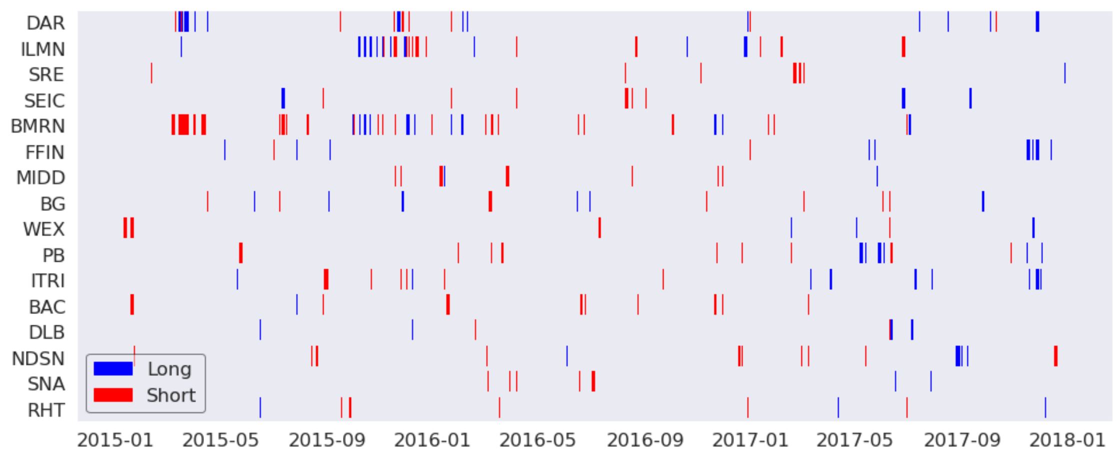

Gross leverage
Gross leverage
2.0
1.5
1.0
0.5
0.0
2015-05
2015-09
2016-01
2016-05
2016-09
2017-01
2017-05
2017-09

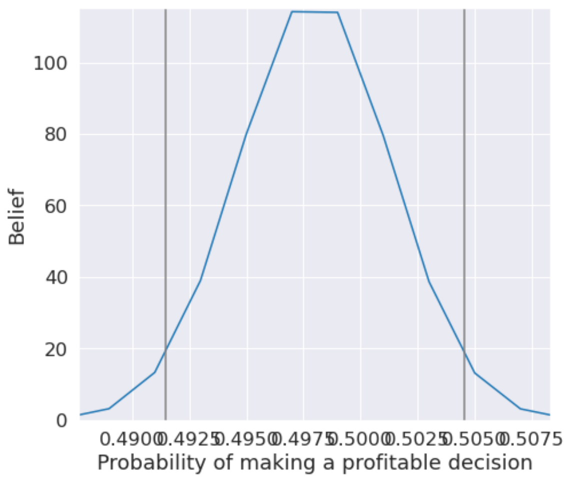

Daily turnover
Turnover
2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
Daily turnover
Average daily turnover, by month
Average daily turnover, net

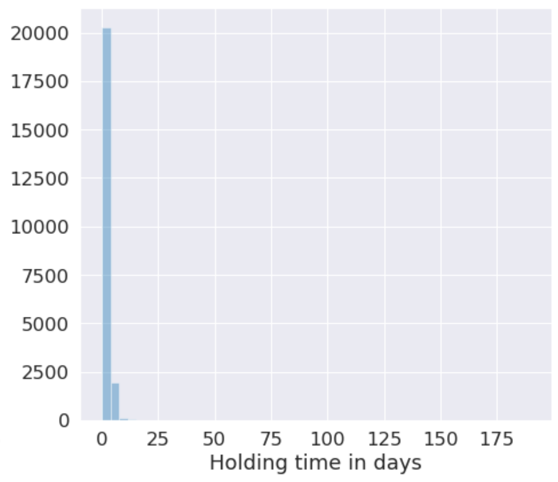

Daily trading volume
Amount of shares traded
30000
25000
20000
15000
10000
5000

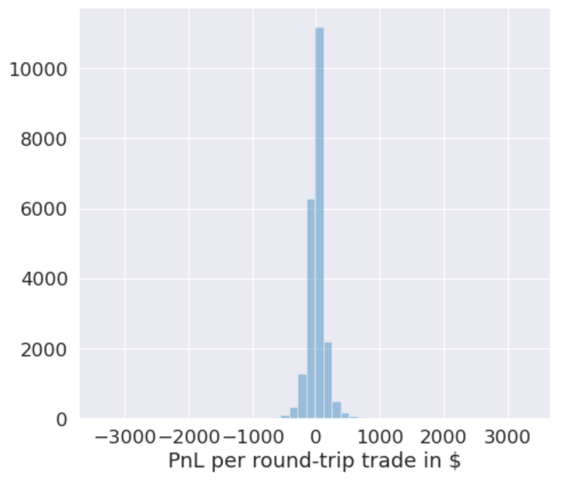

Distribution of daily turnover rates
Density
1.2
1.0
0.8
0.6
0.4
0.2
0.0
−0.5
0.0
0.5
1.0
1.5
2.0
2.5
Turnover rate

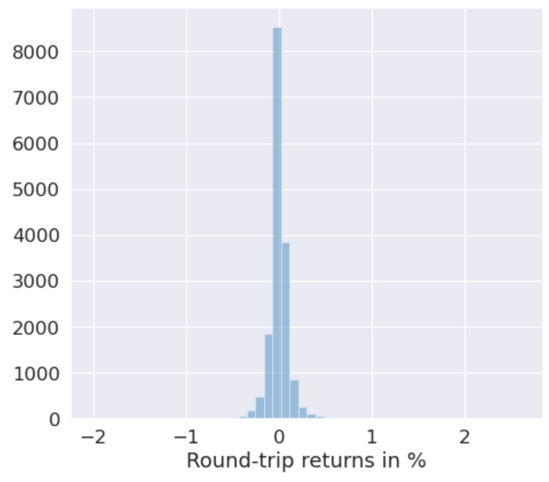

Transaction time distribution
Proportion
1.0
0.8
0.6
0.4
0.2
0.0
09:30
10:00
10:30
11:00
11:30
12:00
12:30
13:00
13:30
14:00
14:30
15:00
15:30
16:00


DAR
ILMN
SRE
SEIC
BMRN
FFIN
MIDD
BG
WEX
PB
ITRI
BAC
DLB
NDSN
SNA
RHT
Long
Short
2015-01
2015-05
2015-09
2016-01
2016-05
2016-09
2017-01
2017-05
2017-09
2018-01


Belief
0
20
40
60
80
100
0.4900
0.4925
0.4950
0.4975
0.5000
0.5025
0.5050
0.5075
Probability of making a profitable decision


0
2500
5000
7500
10000
12500
15000
17500
20000
0
25
50
75
100
125
150
175
Holding time in days


0
2000
4000
6000
8000
10000
−3000
−2000
−1000
0
1000
2000
3000
PnL per round-trip trade in $


0
1000
2000
3000
4000
5000
6000
7000
8000
−2
−1
0
1
2
Round-trip returns in %