07_backtesting_with_zipline

September 29, 2021

1 Backtesting with Zipline - Using the Pipeline API with ML-driven Signals

1.1 Imports & Settings

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

[2]:
```
%matplotlib inline

from pathlib import Path
from collections import defaultdict
from time import time

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

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,
                         get_open_orders, cancel_order,
                         order_target, order_target_percent)
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 JP_EQUITIES
from zipline.pipeline.filters import StaticAssets
from zipline.pipeline.loaders.frame import DataFrameLoader

import pyfolio as pf
```

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

[3]:
```
idx = pd.IndexSlice
sns.set_style('whitegrid')
np.random.seed(42)
```

[4]:
```
results_path = Path('results', 'return_predictions')
if not results_path.exists():
    results_path.mkdir(parents=True)
```

1.1.1 Load zipline extensions

Only need this in notebook to find bundle.

[5]:
```
load_extensions(default=True,
                extensions=[],
                strict=True,
                environ=None)
```

[6]:
```
log_handler = StderrHandler(format_string='[{record.time:%Y-%m-%d %H:%M:%S.%f}]:
 ↪ ' +
                            '{record.level_name}: {record.func_name}: {record.
 ↪message}',
                            level=INFO)
log_handler.push_application()
log = Logger('Algorithm')
```

1.2 Algo Params

The strategy will hold the 25 stocks with the highest positive and lowest negative predictions each as long as there are at least 15 on each side.

[7]:
```
N_LONGS = 25
N_SHORTS = 25
MIN_POSITIONS = 15
```

1.3 Load Data

1.3.1 Quandl Wiki Bundel

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

1.3.2 ML Predictions

We generate the `train` predictions in the notebook `alphalens_signal_quality` and the `test` predictions in the notebook `random_forest_return_signals`.

[9]:
```
def load_predictions(bundle):
    t = 1
    df = pd.concat([pd.read_hdf(results_path / 'predictions.h5', 'train/{:02}'.
 ↪format(t)),
                    pd.read_hdf(results_path / 'predictions.h5', 'test/{:02}'.
 ↪format(t))])
    df = df[~df.index.duplicated()].drop('y_test', axis=1)
    predictions = df.iloc[:, :5].mean(1).to_frame('predictions')

    tickers = predictions.index.get_level_values('ticker').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('ticker')
            .rename(columns=ticker_map)
            .predictions
            .tz_localize('UTC')), assets
```

[10]:
```
predictions, assets = load_predictions(bundle_data)
```

[11]:
```
predictions.info()
```

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 504 entries, 2017-12-05 00:00:00+00:00 to 2019-12-27
00:00:00+00:00
Columns: 946 entries, 1 to 2857
dtypes: float64(946)
memory usage: 3.6 MB
```

1.3.3 Define Custom Dataset

[12]:
```
class SignalData(DataSet):
    predictions = Column(dtype=float)
    domain = JP_EQUITIES
```

1.3.4 Define Pipeline Loaders

[13]:
```
signal_loader = {SignalData.predictions:
                 DataFrameLoader(SignalData.predictions, predictions)}
```

1.4 Pipeline Setup

1.4.1 Custom ML Factor

[14]:
```
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, preds):
        out[:] = preds
```

1.4.2 Create Pipeline

[15]:
```
def compute_signals():
    signals = MLSignal()
    predictions = SignalData.predictions.latest
    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

[16]:
```
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()

    set_slippage(slippage.FixedSlippage(spread=0.00))
    set_commission(commission.PerShare(cost=0.05, min_trade_cost=1))

    schedule_function(rebalance,
                      date_rules.every_day(),
                      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

[17]:
```
def before_trading_start(context, data):
    """
    Called every day before market open.
    """
    output = pipeline_output('signals')
    context.trades = (output['longs'].astype(int)
                      .append(output['shorts'].astype(int).mul(-1))
                      .reset_index()
                      .drop_duplicates()
                      .set_index('index')
                      .squeeze())
```

1.6 Define Rebalancing Logic

[18]:
```
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])
    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

[19]:
```
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

[20]:
```
dates = predictions.index.get_level_values('date')
start_date = dates.min() + pd.DateOffset(day=1)
end_date = dates.max()
```

[21]:
```
print('Start:\t{}\nEnd:\t{}'.format(start_date.date(), end_date.date()))
```

```
Start:  2017-12-01
End:    2019-12-27
```

[22]:
```
start = time()
results = run_algorithm(start=start_date,
                        end=end_date,
                        initialize=initialize,
                        before_trading_start=before_trading_start,
                        capital_base=1e6,
                        data_frequency='daily',
                        bundle='stooq',
                        custom_loader=signal_loader)# need to modify zipline

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

```
[2021-04-16 00:45:11.957893]: INFO: handle_simulation_end: Simulated 521 trading
days
first open: 2017-12-01 14:31:00+00:00
last close: 2019-12-27 21:00:00+00:00

Duration: 19.67s
```

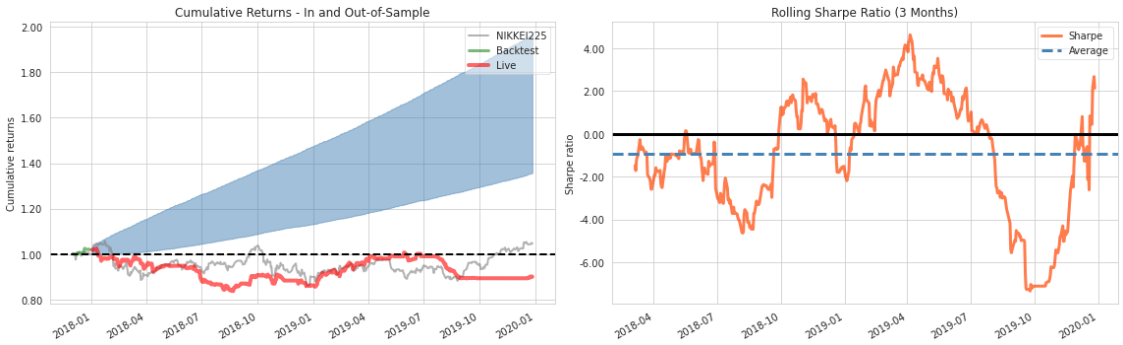
1.9 PyFolio Analysis

[23]:
```
returns, positions, transactions = pf.utils.
 ↪extract_rets_pos_txn_from_zipline(results)
```

[27]:
```
benchmark = web.DataReader('NIKKEI225',
                           'fred',
                           start='2015',
                           end='2020').squeeze()
benchmark = benchmark.pct_change().tz_localize('UTC')
```

1.9.1 Custom Plots

[25]:
```
fig, axes = plt.subplots(ncols=2, figsize=(16, 5))
plot_rolling_returns(returns,
                     factor_returns=benchmark,
                     live_start_date='2018-01-01',
                     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()
fig.savefig((results_path / 'pyfolio_out_of_sample').as_posix(), dpi=300);
```

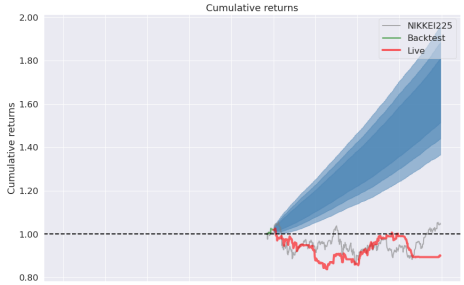


1.9.2 Tear Sheets

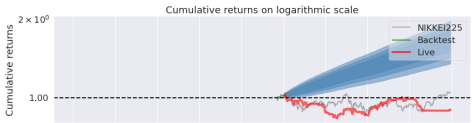
[26]:
```
pf.create_full_tear_sheet(returns,
                          positions=positions,
                          transactions=transactions,
```

```
                            benchmark_rets=benchmark,
                            live_start_date='2018-01-01',
                            round_trips=True)
```

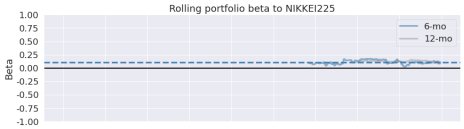
```
<IPython.core.display.HTML object>
<IPython.core.display.HTML object>
<IPython.core.display.HTML object>
<IPython.core.display.HTML object>
<IPython.core.display.HTML object>
<IPython.core.display.HTML object>
<IPython.core.display.HTML object>
<IPython.core.display.HTML object>
<IPython.core.display.HTML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

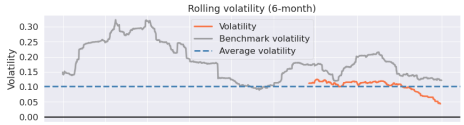

Cumulative returns
NIKKEI225
Backtest
Live
Cumulative returns

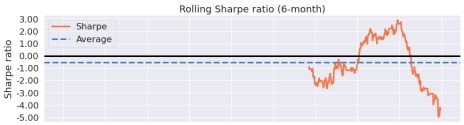

Cumulative returns volatility matched to benchmark
Cumulative returns

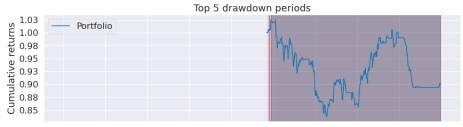

Cumulative returns on logarithmic scale
NIKKEI225
Backtest
Live
Cumulative returns

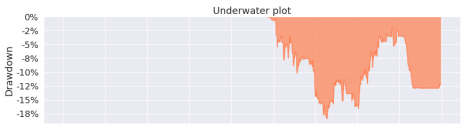

Returns
Returns

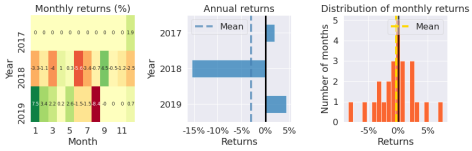

Rolling portfolio beta to NIKKEI225
6-mo
12-mo
Beta

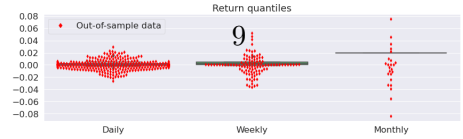

Rolling volatility (6-month)
Volatility
Benchmark volatility
Average volatility
Volatility

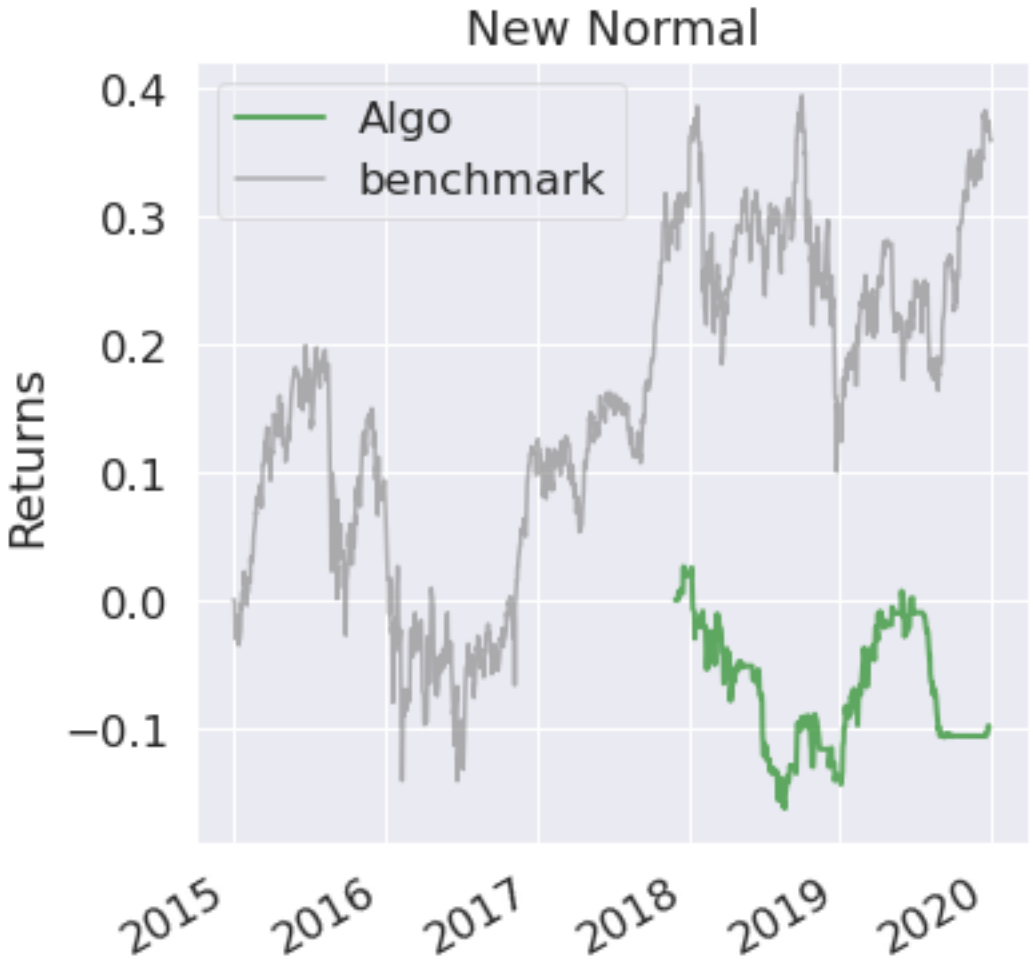

Rolling Sharpe ratio (6-month)
Sharpe
Average
Sharpe ratio

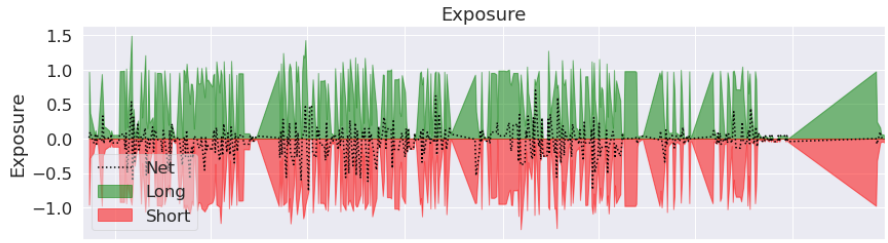

Top 5 drawdown periods
Portfolio
Cumulative returns


Underwater plot
Drawdown


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

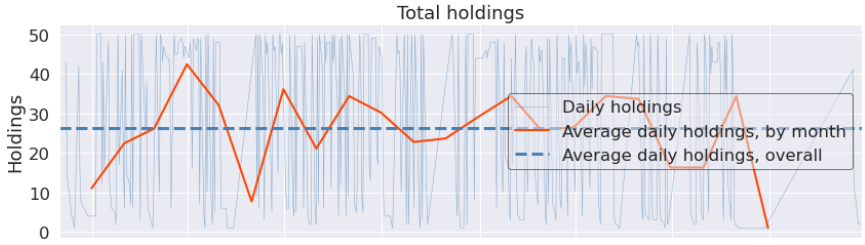

Return quantiles
Out-of-sample data
Daily
Weekly
Monthly

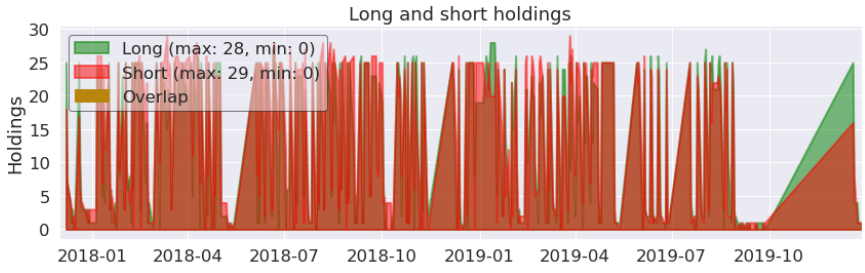

New Normal
Algo
benchmark
Returns
0.4
0.3
0.2
0.1
0.0
−0.1
2015
2016
2017
2018
2019
2020

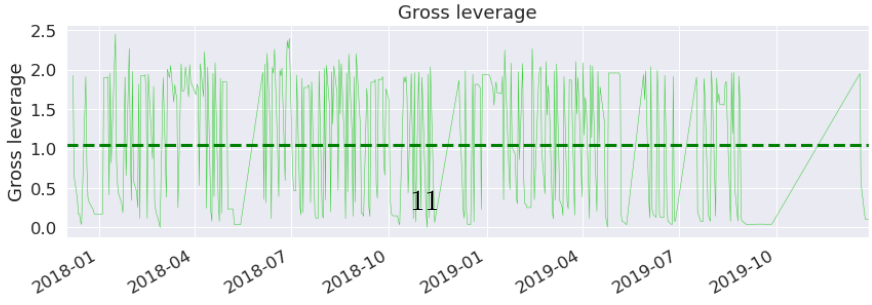

Exposure
Exposure
1.5
1.0
0.5
0.0
−0.5
−1.0
Net
Long
Short

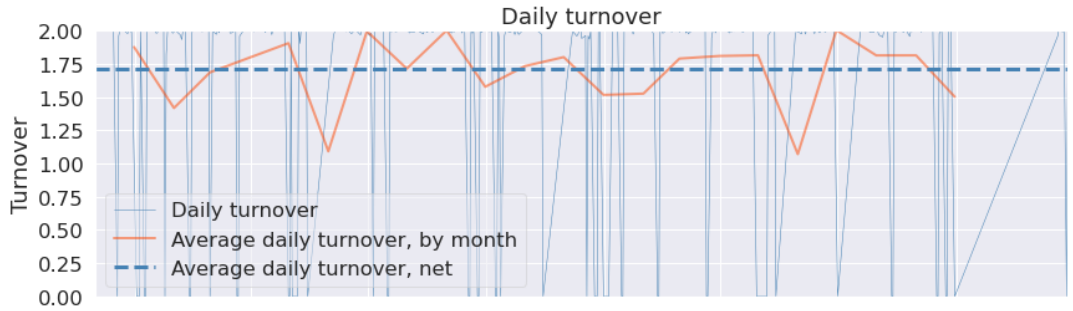

Portfolio allocation over time, only top 10 holdings
Exposure by holding
0.20
0.15
0.10
0.05
0.00
−0.05
−0.10
−0.15

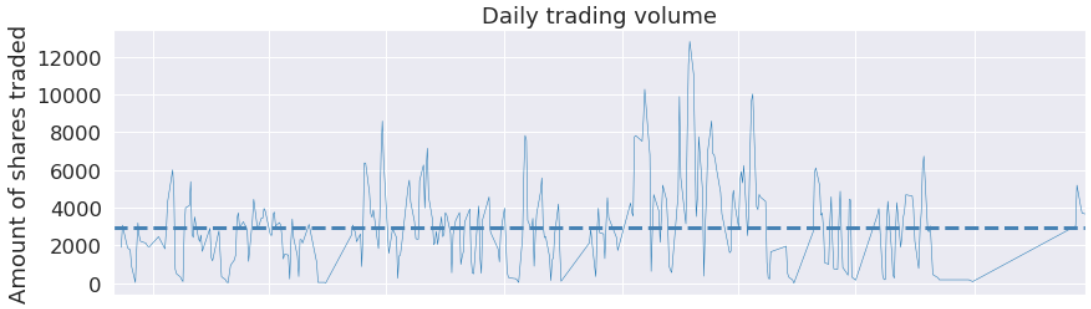

4078.JP
8558.JP
7013.JP
9507.JP
6445.JP
6753.JP
6779.JP
4331.JP
8107.JP
6905.JP
Long/short max and median position concentration
Exposure
0.20
0.15
0.10
0.05
0.00
−0.05
−0.10
−0.15
max_long
median_long
median_short
max_short

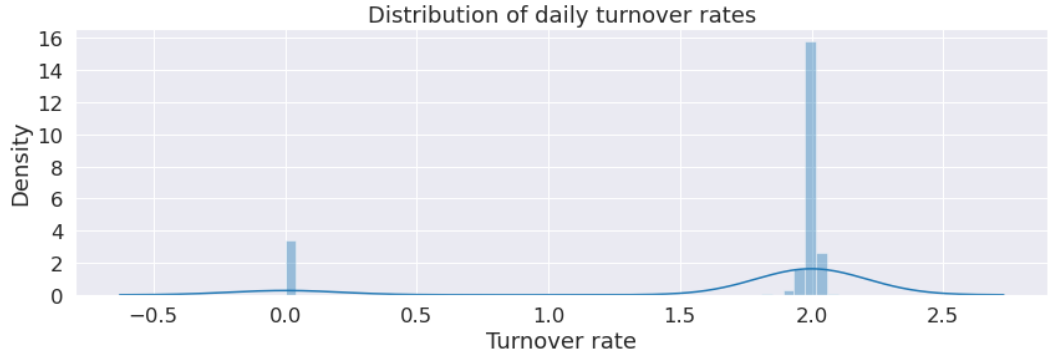

Total holdings
Holdings
50
40
30
20
10
0
Daily holdings
Average daily holdings, by month
Average daily holdings, overall

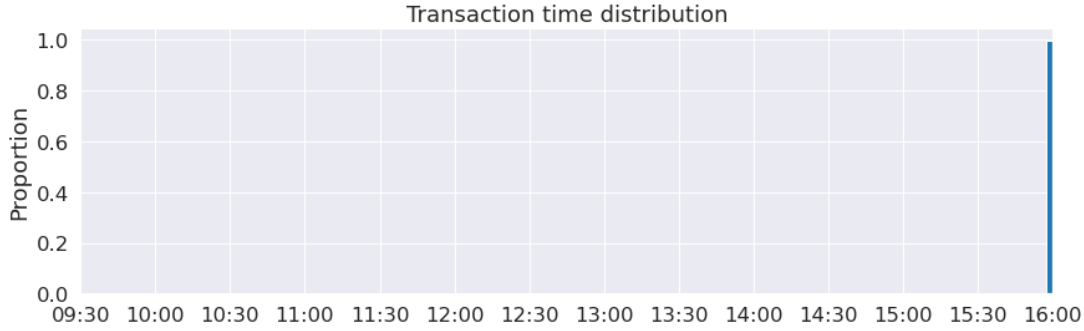

Long and short holdings
Holdings
30
25
20
15
10
5
0
Long (max: 28, min: 0)
Short (max: 29, min: 0)
Overlap
2018-01
2018-04
2018-07
2018-10
2019-01
2019-04
2019-07
2019-10

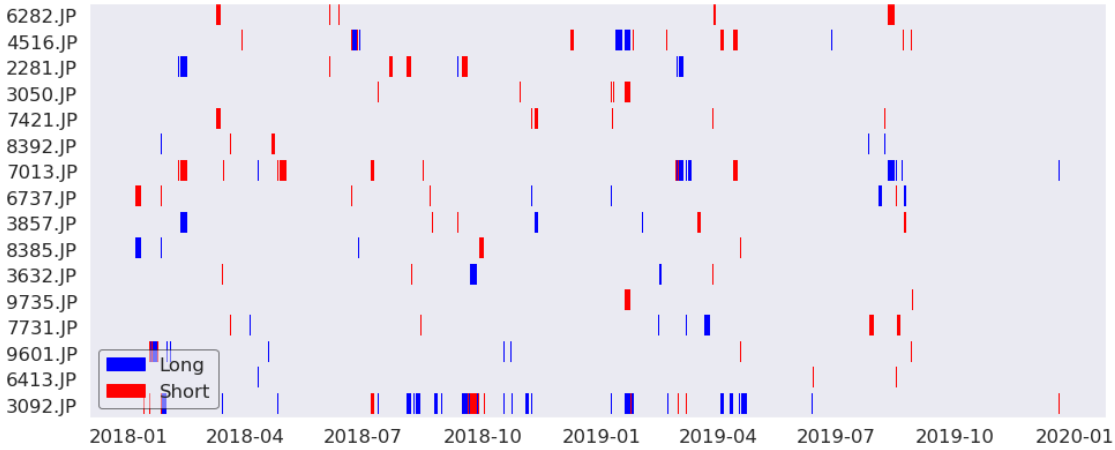

Gross leverage
Gross leverage
2.5
2.0
1.5
1.0
0.5
0.0
2018-01
2018-04
2018-07
2018-10
2019-01
2019-04
2019-07
2019-10

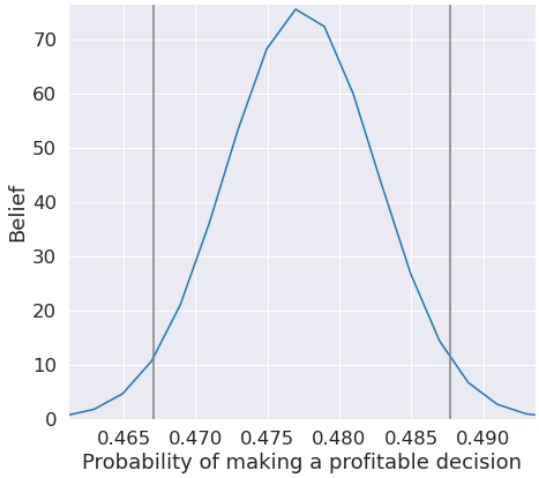

Daily turnover
Turnover
2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
Daily turnover
Average daily turnover, by month
Average daily turnover, net

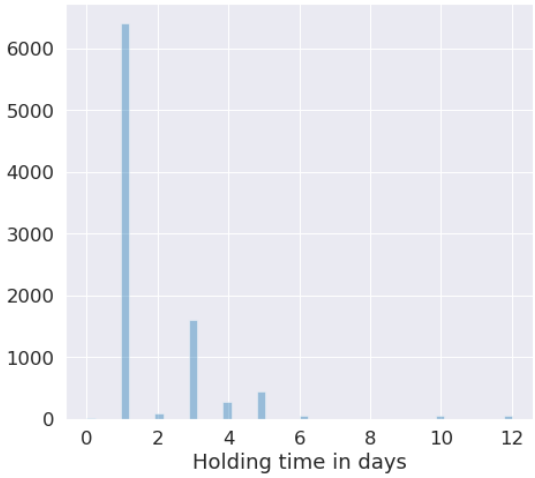

Daily trading volume
Amount of shares traded
12000
10000
8000
6000
4000
2000
0

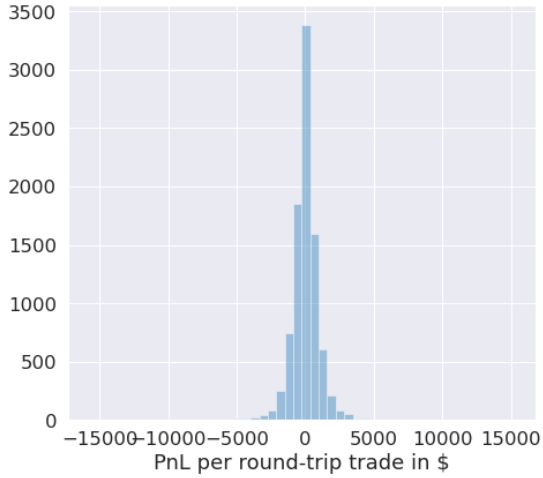

Distribution of daily turnover rates
Density
16
14
12
10
8
6
4
2
0
−0.5
0.0
0.5
1.0
1.5
2.0
2.5
Turnover rate

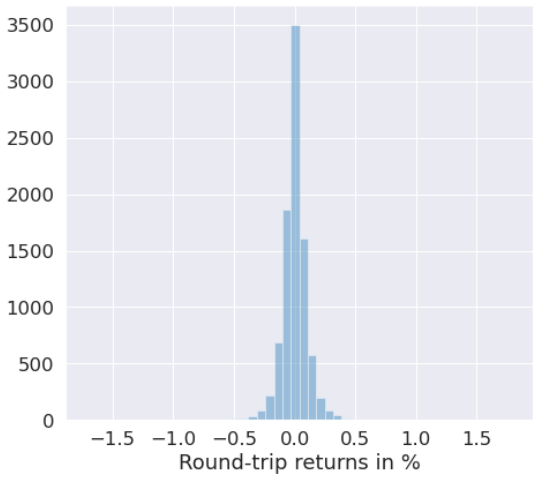

Transaction time distribution
Proportion
1.0
0.8
0.6
0.4
0.2
0.0
09:30
10:00
10:30
11:00
11:30
12:00
12:30
13:00
13:30
14:00
14:30
15:00
15:30
16:00


6282.JP
4516.JP
2281.JP
3050.JP
7421.JP
8392.JP
7013.JP
6737.JP
3857.JP
8385.JP
3632.JP
9735.JP
7731.JP
9601.JP
6413.JP
3092.JP
Long
Short
2018-01
2018-04
2018-07
2018-10
2019-01
2019-04
2019-07
2019-10
2020-01


Belief
Probability of making a profitable decision


Holding time in days


PnL per round-trip trade in $


Round-trip returns in %

[ ]: