03_backtesting_with_backtrader

September 29, 2021

1 Backtesting an ML strategy with Backtrader

1.1 Imports & Settings

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

[2]:
```
%matplotlib inline

from pathlib import Path
import csv
from time import time
import datetime
import numpy as np
import pandas as pd
import pandas_datareader.data as web
import matplotlib.pyplot as plt
import seaborn as sns

import backtrader as bt
from backtrader.feeds import PandasData

import pyfolio as pf
```

[3]:
```
pd.set_option('display.expand_frame_repr', False)
np.random.seed(42)
sns.set_style('darkgrid')
```

[4]:
```
def format_time(t):
    m_, s = divmod(t, 60)
    h, m = divmod(m_, 60)
    return f'{h:>02.0f}:{m:>02.0f}:{s:>02.0f}'
```

1.2 Backtrader Setup

1.2.1 Custom Commission Scheme

[5]:
```
class FixedCommisionScheme(bt.CommInfoBase):
    """
    Simple fixed commission scheme for demo
    """
    params = (
        ('commission', .02),
        ('stocklike', True),
        ('commtype', bt.CommInfoBase.COMM_FIXED),
    )

    def _getcommission(self, size, price, pseudoexec):
        return abs(size) * self.p.commission
```

1.2.2 DataFrame Loader

[6]:
```
OHLCV = ['open', 'high', 'low', 'close', 'volume']
```

[7]:
```
class SignalData(PandasData):
    """
    Define pandas DataFrame structure
    """
    cols = OHLCV + ['predicted']

    # create lines
    lines = tuple(cols)

    # define parameters
    params = {c: -1 for c in cols}
    params.update({'datetime': None})
    params = tuple(params.items())
```

1.2.3 Strategy

Includes an option to only trade on certain weekdays in lines 39/40.

[8]:
```
class MLStrategy(bt.Strategy):
    params = (('n_positions', 10),
              ('min_positions', 5),
              ('verbose', False),
              ('log_file', 'backtest.csv'))

    def log(self, txt, dt=None):
        """ Logger for the strategy"""
        dt = dt or self.datas[0].datetime.datetime(0)
```

```
        with Path(self.p.log_file).open('a') as f:
            log_writer = csv.writer(f)
            log_writer.writerow([dt.isoformat()] + txt.split(','))

    def notify_order(self, order):
        if order.status in [order.Submitted, order.Accepted]:
            return

        # Check if an order has been completed
        # broker could reject order if not enough cash
        if self.p.verbose:
            if order.status in [order.Completed]:
                p = order.executed.price
                if order.isbuy():
                    self.log(f'{order.data._name},BUY executed,{p:.2f}')
                elif order.issell():
                    self.log(f'{order.data._name},SELL executed,{p:.2f}')

            elif order.status in [order.Canceled, order.Margin, order.Rejected]:
                self.log(f'{order.data._name},Order Canceled/Margin/Rejected')

    # bt calls prenext instead of next unless
    # all datafeeds have current values
    # => call next to avoid duplicating logic
    def prenext(self):
        self.next()

    def next(self):
        today = self.datas[0].datetime.date()
#         if today.weekday() not in [0, 3]: # only trade on Mondays;
#             return
        positions = [d._name for d, pos in self.getpositions().items() if pos]
        up, down = {}, {}
        missing = not_missing = 0
        for data in self.datas:
            if data.datetime.date() == today:
                if data.predicted[0] > 0:
                    up[data._name] = data.predicted[0]
                elif data.predicted[0] < 0:
                    down[data._name] = data.predicted[0]

        # sort dictionaries ascending/descending by value
        # returns list of tuples
        shorts = sorted(down, key=down.get)[:self.p.n_positions]
        longs = sorted(up, key=up.get, reverse=True)[:self.p.n_positions]
        n_shorts, n_longs = len(shorts), len(longs)
```

```
        # only take positions if at least min_n longs and shorts
        if n_shorts < self.p.min_positions or n_longs < self.p.min_positions:
            longs, shorts = [], []
        for ticker in positions:
            if ticker not in longs + shorts:
                self.order_target_percent(data=ticker, target=0)
                self.log(f'{ticker},CLOSING ORDER CREATED')

        short_target = -1 / max(self.p.n_positions, n_shorts)
        long_target = 1 / max(self.p.n_positions, n_longs)
        for ticker in shorts:
            self.order_target_percent(data=ticker, target=short_target)
            self.log('{ticker},SHORT ORDER CREATED')
        for ticker in longs:
            self.order_target_percent(data=ticker, target=long_target)
            self.log('{ticker},LONG ORDER CREATED')
```

1.2.4 Create and Configure Cerebro Instance

[9]:
```
cerebro = bt.Cerebro()  # create a "Cerebro" instance
cash = 10000
# comminfo = FixedCommisionScheme()
# cerebro.broker.addcommissioninfo(comminfo)
cerebro.broker.setcash(cash)
```

1.2.5 Add input data

[10]:
```
idx = pd.IndexSlice
data = pd.read_hdf('00_data/backtest.h5', 'data').sort_index()
tickers = data.index.get_level_values(0).unique()

for ticker in tickers:
    df = data.loc[idx[ticker, :], :].droplevel('ticker', axis=0)
    df.index.name = 'datetime'
    bt_data = SignalData(dataname=df)
    cerebro.adddata(bt_data, name=ticker)
```

1.2.6 Run Strategy Backtest

[11]:
```
cerebro.addanalyzer(bt.analyzers.PyFolio, _name='pyfolio')
cerebro.addstrategy(MLStrategy, n_positions=25, min_positions=20,
                    verbose=True, log_file='bt_log.csv')
start = time()
results = cerebro.run()
ending_value = cerebro.broker.getvalue()
duration = time() - start
```

```
print(f'Final Portfolio Value: {ending_value:,.2f}')
print(f'Duration: {format_time(duration)}')
```

```
Final Portfolio Value: 10,078.17
Duration: 00:00:56
```

1.2.7 Plot result

Disabled because large number of datafeeds all plot separately.

[12]:
```
# cerebro.plot()  # plot the results
# figure = cerebro.plot(style='candlebars')[0][0]
# figure.savefig(f'backtrader.png')
```

1.2.8 Get pyfolio inputs

[13]:
```
# prepare pyfolio inputs
pyfolio_analyzer = results[0].analyzers.getbyname('pyfolio')
returns, positions, transactions, gross_lev = pyfolio_analyzer.get_pf_items()

returns.to_hdf('backtrader.h5', 'returns')
positions.to_hdf('backtrader.h5', 'positions')
transactions.to_hdf('backtrader.h5', 'transactions/')
gross_lev.to_hdf('backtrader.h5', 'gross_lev')
```

1.3 Run pyfolio analysis

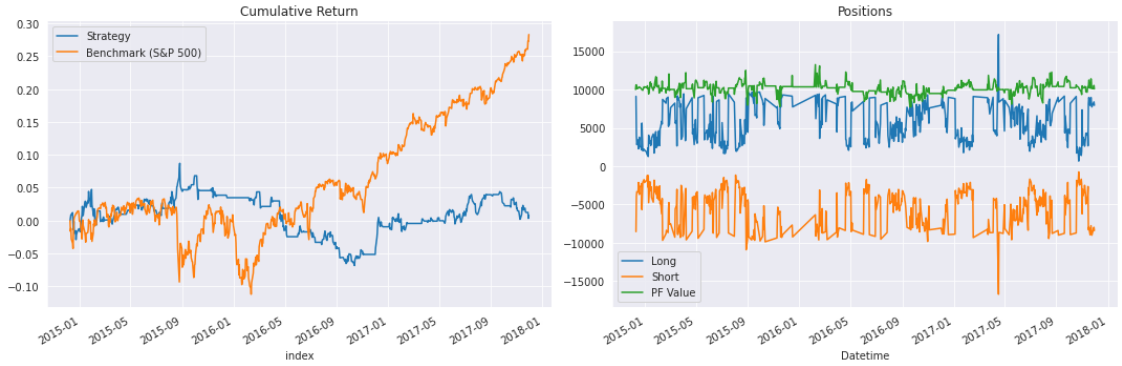
[14]:
```
returns = pd.read_hdf('backtrader.h5', 'returns')
positions = pd.read_hdf('backtrader.h5', 'positions')
transactions = pd.read_hdf('backtrader.h5', 'transactions/')
gross_lev = pd.read_hdf('backtrader.h5', 'gross_lev')
```

[15]:
```
benchmark = web.DataReader('SP500', 'fred', '2014', '2018').squeeze()
benchmark = benchmark.pct_change().tz_localize('UTC')
```

[16]:
```
daily_tx = transactions.groupby(level=0)
longs = daily_tx.value.apply(lambda x: x.where(x>0).sum())
shorts = daily_tx.value.apply(lambda x: x.where(x<0).sum())
```

[17]:
```
fig, axes = plt.subplots(ncols=2, figsize=(15, 5))

df = returns.to_frame('Strategy').join(benchmark.to_frame('Benchmark (S&P␣
↪500)'))
df.add(1).cumprod().sub(1).plot(ax=axes[0], title='Cumulative Return')

longs.plot(label='Long',ax=axes[1], title='Positions')
```

```
shorts.plot(ax=axes[1], label='Short')
positions.cash.plot(ax=axes[1], label='PF Value')
axes[1].legend()
sns.despine()
fig.tight_layout();
```



[18]:
```
pf.create_full_tear_sheet(returns,
                          transactions=transactions,
                          positions=positions,
                          benchmark_rets=benchmark.dropna())
```

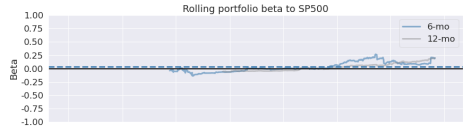
```
<IPython.core.display.HTML object>

<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>
```

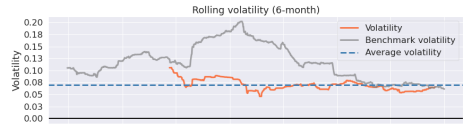
Cumulative returns
SP500
Backtest
Cumulative returns

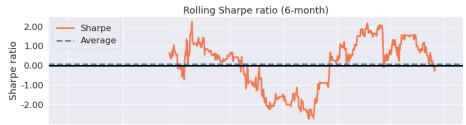
Cumulative returns volatility matched to benchmark
Cumulative returns

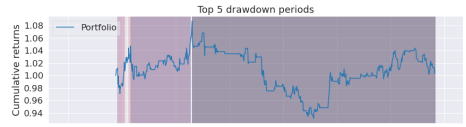
Cumulative returns on logarithmic scale
SP500
Backtest
Cumulative returns

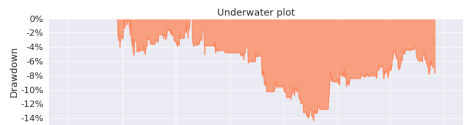
Returns
Returns

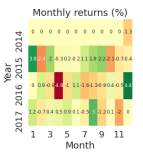

Rolling portfolio beta to SP500
6-mo
12-mo
Beta

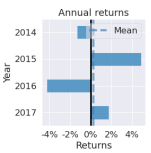

Rolling volatility (6-month)
Volatility
Benchmark volatility
Average volatility
Volatility

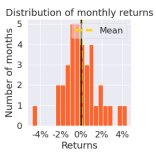

Rolling Sharpe ratio (6-month)
Sharpe
Average
Sharpe ratio

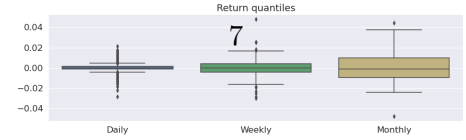

Top 5 drawdown periods
Portfolio
Cumulative returns

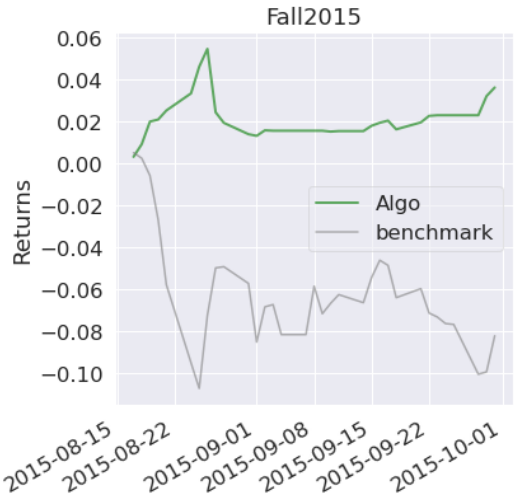

Underwater plot
Drawdown

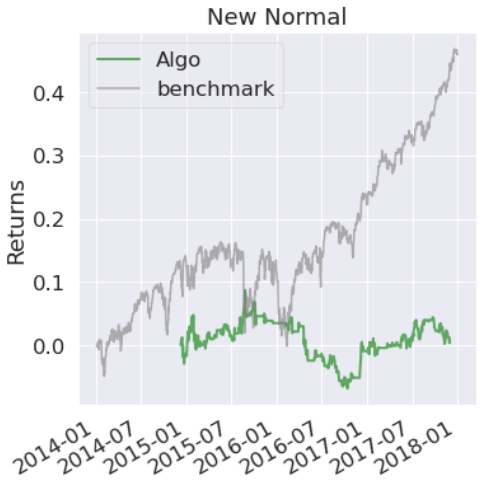

Monthly returns (%)
Year
Month

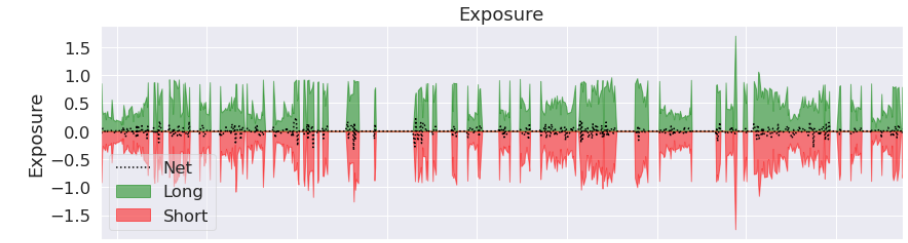

Annual returns
Mean
Year
Returns


Distribution of monthly returns
Mean
Number of months
Returns

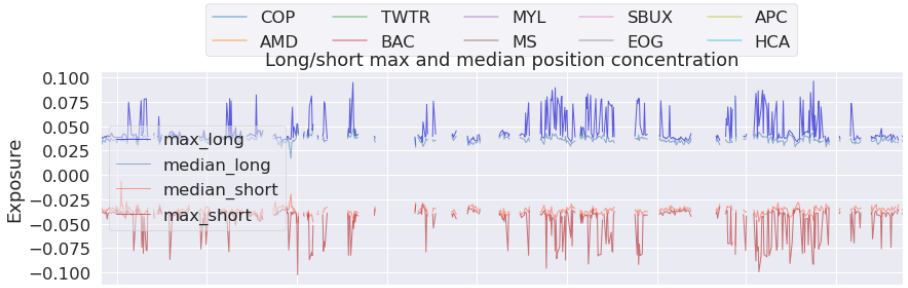

Return quantiles
Daily
Weekly
Monthly

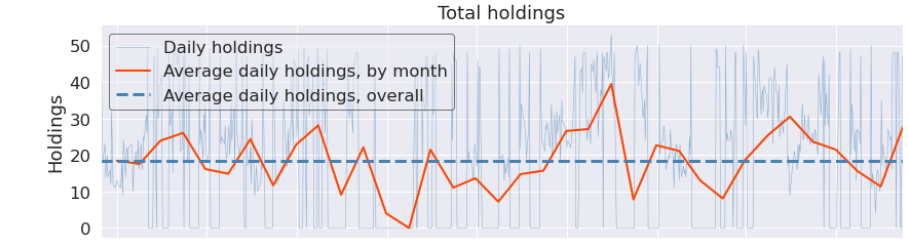

Fall2015
Returns
0.06
0.04
0.02
0.00
−0.02
−0.04
−0.06
−0.08
−0.10
Algo
benchmark
2015-08-15
2015-08-22
2015-09-01
2015-09-08
2015-09-15
2015-09-22
2015-10-01

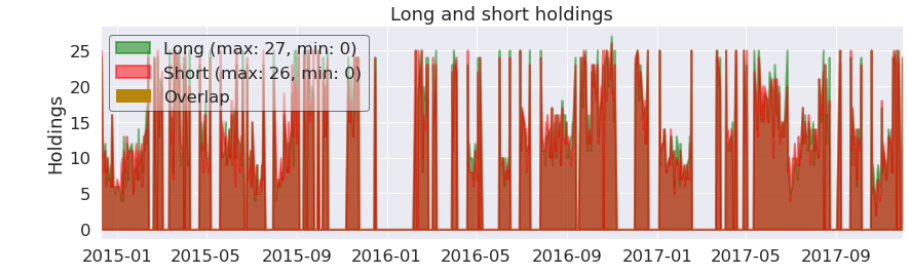

New Normal
Returns
0.4
0.3
0.2
0.1
0.0
Algo
benchmark
2014-01
2014-07
2015-01
2015-07
2016-01
2016-07
2017-01
2017-07
2018-01

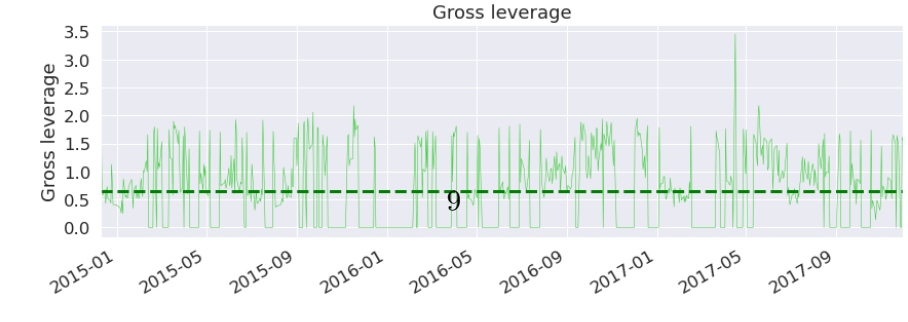

Exposure
Exposure
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
Net
Long
Short

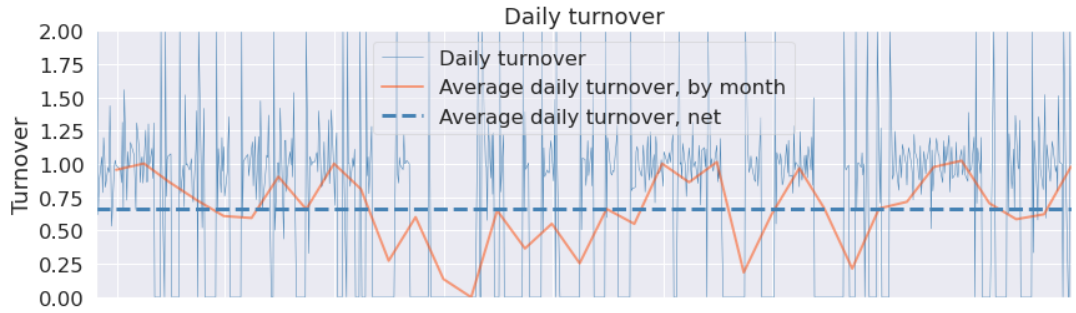
Portfolio allocation over time, only top 10 holdings
Exposure by holding
0.100
0.075
0.050
0.025
0.000
−0.025
−0.050
−0.075
−0.100

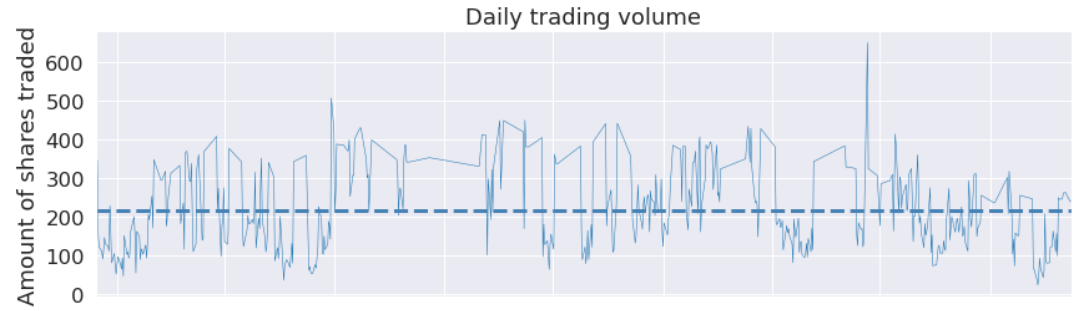

COP
AMD
TWTR
BAC
MYL
MS
SBUX
EOG
APC
HCA
Long/short max and median position concentration
Exposure
0.100
0.075
0.050
0.025
0.000
−0.025
−0.050
−0.075
−0.100
max_long
median_long
median_short
max_short

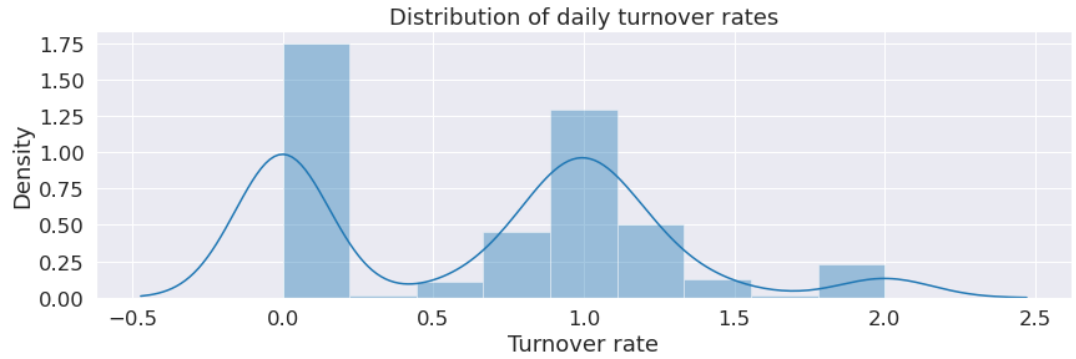

Total holdings
Holdings
50
40
30
20
10
0
Daily holdings
Average daily holdings, by month
Average daily holdings, overall

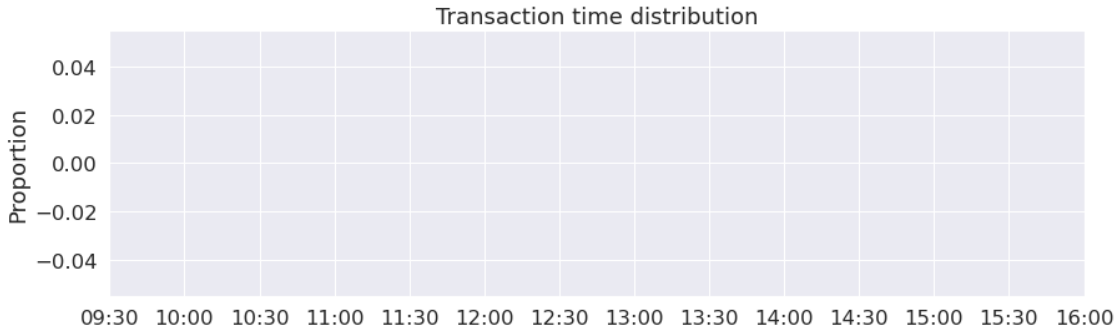

Long and short holdings
Holdings
25
20
15
10
5
0
Long (max: 27, min: 0)
Short (max: 26, min: 0)
Overlap
2015-01
2015-05
2015-09
2016-01
2016-05
2016-09
2017-01
2017-05
2017-09


Gross leverage
Gross leverage
3.5
3.0
2.5
2.0
1.5
1.0
0.5
0.0
2015-01
2015-05
2015-09
2016-01
2016-05
2016-09
2017-01
2017-05
2017-09


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


Daily trading volume
Amount of shares traded
600
500
400
300
200
100
0


Distribution of daily turnover rates
Density
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
−0.5
0.0
0.5
1.0
1.5
2.0
2.5
Turnover rate


Transaction time distribution
Proportion
0.04
0.02
0.00
−0.02
−0.04
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

[ ]: