alpha_factor_zipline_with_trades

September 29, 2021

1 zipline MeanReversion Backtest

In the chapter 04, we introduced `zipline` to simulate the computation of alpha factors from trailing cross-sectional market, fundamental, and alternative data.

Now we will exploit the alpha factors to derive and act on buy and sell signals using the custom MeanReversion factor developed in the last chapter.

Run the following from the command line to create a `conda` environment with `zipline` and `pyfolio`:

`conda env create -f environment.yml`

This assumes you have miniconda3 installed.

1.1 Imports

[4]:
```
import sys
import pandas as pd
from pytz import UTC
from zipline import run_algorithm
from zipline.api import (attach_pipeline, date_rules, time_rules,␣
 ↪order_target_percent,
                         pipeline_output, record, schedule_function,␣
 ↪get_open_orders, calendars,
                         set_commission, set_slippage)
from zipline.finance import commission, slippage
from zipline.pipeline import Pipeline, CustomFactor
from zipline.pipeline.factors import Returns, AverageDollarVolume
import logbook
import matplotlib.pyplot as plt
import seaborn as sns

from pyfolio.utils import extract_rets_pos_txn_from_zipline
```

```
WARNING (theano.tensor.blas): Using NumPy C-API based implementation for BLAS
functions.
```

[5]:
```
sns.set_style('darkgrid')
```

1.2 Logging Setup

[6]:
```
# setup stdout logging
zipline_logging = logbook.NestedSetup([
    logbook.NullHandler(level=logbook.DEBUG),
    logbook.StreamHandler(sys.stdout, level=logbook.INFO),
    logbook.StreamHandler(sys.stderr, level=logbook.ERROR),
])
zipline_logging.push_application()
```

1.3 Algo Settings

[7]:
```
# Settings
MONTH = 21
YEAR = 12 * MONTH
N_LONGS = 200
N_SHORTS = 0
VOL_SCREEN = 1000
```

[8]:
```
start = pd.Timestamp('2010-01-01', tz=UTC)
end = pd.Timestamp('2018-01-01', tz=UTC)
capital_base = 1e7
```

1.4 Mean Reversion Factor

[9]:
```
class MeanReversion(CustomFactor):
    """Compute ratio of latest monthly return to 12m average,
       normalized by std dev of monthly returns"""
    inputs = [Returns(window_length=MONTH)]
    window_length = YEAR

    def compute(self, today, assets, out, monthly_returns):
        df = pd.DataFrame(monthly_returns)
        out[:] = df.iloc[-1].sub(df.mean()).div(df.std())
```

1.5 Create Pipeline

The Pipeline created by the `compute_factors()` method returns a table with a long and a short column for the 25 stocks with the largest negative and positive deviations of their last monthly return from its annual average, normalized by the standard deviation. It also limited the universe to the 500 stocks with the highest average trading volume over the last 30 trading days.

[10]:
```
def compute_factors():
    """Create factor pipeline incl. mean reversion,
        filtered by 30d Dollar Volume; capture factor ranks"""
    mean_reversion = MeanReversion()
    dollar_volume = AverageDollarVolume(window_length=30)
```

```
    return Pipeline(columns={'longs'  : mean_reversion.bottom(N_LONGS),
                             'shorts' : mean_reversion.top(N_SHORTS),
                             'ranking': mean_reversion.rank(ascending=False)},
                    screen=dollar_volume.top(VOL_SCREEN))
```

Before_trading_start() ensures the daily execution of the pipeline and the recording of the results, including the current prices.

[11]:
```
def before_trading_start(context, data):
    """Run factor pipeline"""
    context.factor_data = pipeline_output('factor_pipeline')
    record(factor_data=context.factor_data.ranking)
    assets = context.factor_data.index
    record(prices=data.current(assets, 'price'))
```

1.6 Set up Rebalancing

The new rebalance() method submits trade orders to the exec_trades() method for the assets flagged for long and short positions by the pipeline with equal positive and negative weights. It also divests any current holdings that are no longer included in the factor signals:

[12]:
```
def rebalance(context, data):
    """Compute long, short and obsolete holdings; place trade orders"""
    factor_data = context.factor_data
    assets = factor_data.index

    longs = assets[factor_data.longs]
    shorts = assets[factor_data.shorts]
    divest = context.portfolio.positions.keys() - longs.union(shorts)

    exec_trades(data, assets=divest, target_percent=0)
    exec_trades(data, assets=longs, target_percent=1 / N_LONGS if N_LONGS else␣
↪0)
    exec_trades(data, assets=shorts, target_percent=-1 / N_SHORTS if N_SHORTS␣
↪else 0)
```

[13]:
```
def exec_trades(data, assets, target_percent):
    """Place orders for assets using target portfolio percentage"""
    for asset in assets:
        if data.can_trade(asset) and not get_open_orders(asset):
            order_target_percent(asset, target_percent)
```

1.7 Initialize Backtest

The `rebalance()` method runs according to `date_rules` and `time_rules` set by the `schedule_function()` utility at the beginning of the week, right after market_open as stipulated by the built-in US_EQUITIES calendar (see docs for details on rules).

You can also specify a trade commission both in relative terms and as a minimum amount. There is also an option to define slippage, which is the cost of an adverse change in price between trade decision and execution

[14]:
```
def initialize(context):
    """Setup: register pipeline, schedule rebalancing,
        and set trading params"""
    attach_pipeline(compute_factors(), 'factor_pipeline')
    schedule_function(rebalance,
                      date_rules.week_start(),
                      time_rules.market_open(),
                      calendar=calendars.US_EQUITIES)

    set_commission(us_equities=commission.PerShare(cost=0.00075,␣
↪min_trade_cost=.01))
    set_slippage(us_equities=slippage.VolumeShareSlippage(volume_limit=0.0025,␣
↪price_impact=0.01))
```

1.8 Run Algorithm

The algorithm executes upon calling the run_algorithm() function and returns the backtest performance DataFrame.

[15]:
```
backtest = run_algorithm(start=start,
                         end=end,
                         initialize=initialize,
                         before_trading_start=before_trading_start,
                         capital_base=capital_base)
```

```
[2019-06-27 21:04:59.129108] INFO: Loader: Cache at
/home/stefan/.zipline/data/SPY_benchmark.csv does not have data from 2010-01-04
00:00:00+00:00 to 2017-12-29 00:00:00+00:00.

[2019-06-27 21:04:59.130378] INFO: Loader: Downloading benchmark data for 'SPY'
from 2009-12-31 00:00:00+00:00 to 2017-12-29 00:00:00+00:00
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-15-e2ef7ab499a9> in <module>()
      3                          initialize=initialize,
      4                          before_trading_start=before_trading_start,
----> 5                          capital_base=capital_base)

/home/stefan/.pyenv/versions/miniconda3-latest/envs/backtesting/lib/python2.7/
↪site-packages/zipline/utils/run_algo.pyc in run_algorithm(start, end,␣
↪initialize, capital_base, handle_data, before_trading_start, analyze,␣
↪data_frequency, data, bundle, bundle_timestamp, trading_calendar, metrics_set␣
↪default_extension, extensions, strict_extensions, environ, blotter)
    428         local_namespace=False,
```

```
    429         environ=environ,
--> 430         blotter=blotter,
    431     )

/home/stefan/.pyenv/versions/miniconda3-latest/envs/backtesting/lib/python2.7/
 ↪site-packages/zipline/utils/run_algo.pyc in _run(handle_data, initialize,␣
 ↪before_trading_start, analyze, algofile, algotext, defines, data_frequency,␣
 ↪capital_base, data, bundle, bundle_timestamp, start, end, output,␣
 ↪trading_calendar, print_algo, metrics_set, local_namespace, environ, blotter)
    157             trading_calendar=trading_calendar,
    158             trading_day=trading_calendar.day,
--> 159             trading_days=trading_calendar.schedule[start:end].index,
    160         )
    161         first_trading_day =\

/home/stefan/.pyenv/versions/miniconda3-latest/envs/backtesting/lib/python2.7/
 ↪site-packages/zipline/finance/trading.pyc in __init__(self, load, bm_symbol,␣
 ↪exchange_tz, trading_calendar, trading_day, trading_days, asset_db_path,␣
 ↪future_chain_predicates, environ)
    101             trading_day,
    102             trading_days,
--> 103             self.bm_symbol,
    104         )
    105

/home/stefan/.pyenv/versions/miniconda3-latest/envs/backtesting/lib/python2.7/
 ↪site-packages/zipline/data/loader.pyc in load_market_data(trading_day,␣
 ↪trading_days, bm_symbol, environ)
    147         # date so that we can compute returns for the first date.
    148         trading_day,
--> 149         environ,
    150     )
    151     tc = ensure_treasury_data(

/home/stefan/.pyenv/versions/miniconda3-latest/envs/backtesting/lib/python2.7/
 ↪site-packages/zipline/data/loader.pyc in ensure_benchmark_data(symbol,␣
 ↪first_date, last_date, now, trading_day, environ)
    214
    215     try:
--> 216         data = get_benchmark_returns(symbol)
    217         data.to_csv(get_data_filepath(filename, environ))
    218     except (OSError, IOError, HTTPError):

/home/stefan/.pyenv/versions/miniconda3-latest/envs/backtesting/lib/python2.7/
 ↪site-packages/zipline/data/benchmarks.pyc in get_benchmark_returns(symbol)
     33         'https://api.iextrading.com/1.0/stock/{}/chart/5y'.format(symbol)
     34     )
---> 35     data = r.json()
     36
     37     df = pd.DataFrame(data)
```

```
/home/stefan/.local/lib/python2.7/site-packages/requests/models.pyc in␣
↪json(self, **kwargs)
    890                     # used.
    891                     pass
--> 892         return complexjson.loads(self.text, **kwargs)
    893
    894     @property

/home/stefan/.pyenv/versions/miniconda3-latest/envs/backtesting/lib/python2.7/
↪json/__init__.pyc in loads(s, encoding, cls, object_hook, parse_float,␣
↪parse_int, parse_constant, object_pairs_hook, **kw)
    337             parse_int is None and parse_float is None and
    338             parse_constant is None and object_pairs_hook is None and no␣
↪kw):
--> 339         return _default_decoder.decode(s)
    340     if cls is None:
    341         cls = JSONDecoder

/home/stefan/.pyenv/versions/miniconda3-latest/envs/backtesting/lib/python2.7/
↪json/decoder.pyc in decode(self, s, _w)
    362
    363         """
--> 364         obj, end = self.raw_decode(s, idx=_w(s, 0).end())
    365         end = _w(s, end).end()
    366         if end != len(s):

/home/stefan/.pyenv/versions/miniconda3-latest/envs/backtesting/lib/python2.7/
↪json/decoder.pyc in raw_decode(self, s, idx)
    380             obj, end = self.scan_once(s, idx)
    381         except StopIteration:
--> 382             raise ValueError("No JSON object could be decoded")
    383         return obj, end

ValueError: No JSON object could be decoded
```

1.9 Extract pyfolio Inputs

The `extract_rets_pos_txn_from_zipline` utility provided by `pyfolio` extracts the data used to compute performance metrics.

[13]:
```
returns, positions, transactions = extract_rets_pos_txn_from_zipline(backtest)
```

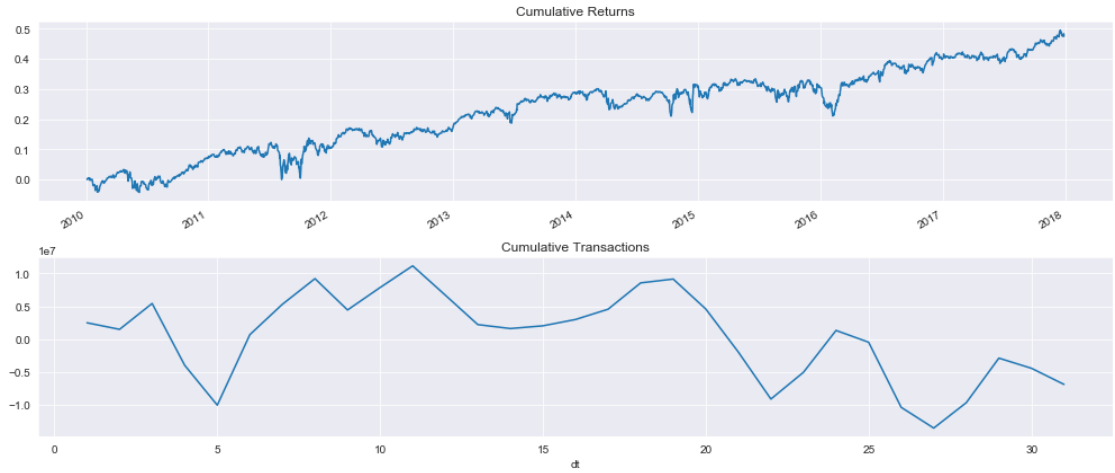
1.10 Persist Results for use with pyfolio

[14]:
```
with pd.HDFStore('backtests.h5') as store:
    store.put('backtest', backtest)
    store.put('returns', returns)
    store.put('positions', positions)
    store.put('transactions', transactions)
```

```
/home/stefan/.pyenv/versions/miniconda3-latest/envs/env_zipline/lib/python3.5/si
te-packages/IPython/core/interactiveshell.py:2901: PerformanceWarning:
your performance may suffer as PyTables will pickle object types that it cannot
map directly to c-types [inferred_type->mixed,key->block4_values]
[items->['factor_data', 'orders', 'period_label', 'positions', 'prices',
'transactions']]

  if self.run_code(code, result):
/home/stefan/.pyenv/versions/miniconda3-latest/envs/env_zipline/lib/python3.5/si
te-packages/IPython/core/interactiveshell.py:2961: PerformanceWarning:
your performance may suffer as PyTables will pickle object types that it cannot
map directly to c-types [inferred_type->mixed,key->axis0] [items->None]

  exec(code_obj, self.user_global_ns, self.user_ns)
/home/stefan/.pyenv/versions/miniconda3-latest/envs/env_zipline/lib/python3.5/si
te-packages/IPython/core/interactiveshell.py:2961: PerformanceWarning:
your performance may suffer as PyTables will pickle object types that it cannot
map directly to c-types [inferred_type->mixed,key->block0_items] [items->None]

  exec(code_obj, self.user_global_ns, self.user_ns)
/home/stefan/.pyenv/versions/miniconda3-latest/envs/env_zipline/lib/python3.5/si
te-packages/IPython/core/interactiveshell.py:2901: PerformanceWarning:
your performance may suffer as PyTables will pickle object types that it cannot
map directly to c-types [inferred_type->mixed,key->block3_values]
[items->['commission', 'order_id', 'sid', 'symbol']]

  if self.run_code(code, result):
```

1.11 Plot Results

[42]:
```
fig, axes= plt.subplots(nrows=2, figsize=(14,6))
returns.add(1).cumprod().sub(1).plot(ax=axes[0], title='Cumulative Returns')
transactions.groupby(transactions.dt.dt.day).txn_dollars.sum().cumsum().
 ↪plot(ax=axes[1], title='Cumulative Transactions')
fig.tight_layout();
```



[43]:
```
positions.info()
```

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 2012 entries, 2010-01-05 to 2017-12-29
Columns: 1717 entries, Equity(0 [A]) to cash
dtypes: float64(1717)
memory usage: 26.5 MB
```

[26]:
```
transactions.describe()
```

[26]:

	amount	price	txn_dollars
count	54596.000000	54596.000000	54596.000000
mean	6.639039	54.526810	-125.974292
std	3164.246839	70.158189	46190.665141
min	-193113.000000	0.260000	-260289.136266
25%	-798.000000	23.870000	-50045.888868
50%	7.000000	40.440000	-401.450000
75%	790.250000	65.459000	48254.080538
max	312268.000000	2846.059938	110457.269868