rank_features_solution

April 24, 2023

1 Rank Features (Solution)

The creator of Shapley Additive Explanations, Scott Lundberg, has written an efficient implementation that we can install and use. We'll be able to use this to determine both local feature importance (for a single observation) and global feature importance (for all training samples as a whole). To aggregate local feature importance into global feature importance, we take the absolute values of the local feature importances, and then average them.

We can calculate the feature importance using sklearn and using the Shap library.

Based on the feature importances, we can think about modifying features to improve them. Then we can re-train the model on the modified features.

Finally, we can prune the feature set to just use the most relevant features.

In [ ]:
```
# Note, this will install zipline and alphalens, which will take some time
import sys
!{sys.executable} -m pip install --quiet -r requirements.txt
```

In [ ]:
```
import numpy as np
import pandas as pd
import time

import matplotlib.pyplot as plt
%matplotlib inline
```

In [ ]:
```
plt.style.use('ggplot')
plt.rcParams['figure.figsize'] = (14, 8)
```

In [ ]:
```
import os
import project_helper
from zipline.data import bundles

os.environ['ZIPLINE_ROOT'] = os.path.join(os.getcwd(), '..', '..', 'data', 'module_4_qui

ingest_func = bundles.csvdir.csvdir_equities(['daily'], project_helper.EOD_BUNDLE_NAME)
bundles.register(project_helper.EOD_BUNDLE_NAME, ingest_func)

print('Data Registered')
```

```
In [ ]: from zipline.pipeline import Pipeline
        from zipline.pipeline.factors import AverageDollarVolume
        from zipline.utils.calendars import get_calendar


        universe = AverageDollarVolume(window_length=120).top(500)
        trading_calendar = get_calendar('NYSE')
        bundle_data = bundles.load(project_helper.EOD_BUNDLE_NAME)
        engine = project_helper.build_pipeline_engine(bundle_data, trading_calendar)
```

```
In [ ]: # Test
        universe_end_date = pd.Timestamp('2016-01-05', tz='UTC')

        universe_tickers = engine\
            .run_pipeline(
                Pipeline(screen=universe),
                universe_end_date,
                universe_end_date)\
            .index.get_level_values(1)\
            .values.tolist()
```

```
In [ ]: from zipline.data.data_portal import DataPortal

        data_portal = DataPortal(
            bundle_data.asset_finder,
            trading_calendar=trading_calendar,
            first_trading_day=bundle_data.equity_daily_bar_reader.first_trading_day,
            equity_minute_reader=None,
            equity_daily_reader=bundle_data.equity_daily_bar_reader,
            adjustment_reader=bundle_data.adjustment_reader)

        def get_pricing(data_portal, trading_calendar, assets, start_date, end_date, field='clos
            end_dt = pd.Timestamp(end_date.strftime('%Y-%m-%d'), tz='UTC', offset='C')
            start_dt = pd.Timestamp(start_date.strftime('%Y-%m-%d'), tz='UTC', offset='C')

            end_loc = trading_calendar.closes.index.get_loc(end_dt)
            start_loc = trading_calendar.closes.index.get_loc(start_dt)

            return data_portal.get_history_window(
                assets=assets,
                end_dt=end_dt,
                bar_count=end_loc - start_loc,
                frequency='1d',
                field=field,
                data_frequency='daily')
```

1.1 Make Factors

- Take the same factors we have been using:

```
In [ ]: from zipline.pipeline.factors import CustomFactor, DailyReturns, Returns, SimpleMovingAv
        from zipline.pipeline.data import USEquityPricing

        factor_start_date = universe_end_date - pd.DateOffset(years=3, days=2)
        sector = project_helper.Sector()

        def momentum_1yr(window_length, universe, sector):
            return Returns(window_length=window_length, mask=universe) \
                .demean(groupby=sector) \
                .rank() \
                .zscore()

        def mean_reversion_5day_sector_neutral(window_length, universe, sector):
            return -Returns(window_length=window_length, mask=universe) \
                .demean(groupby=sector) \
                .rank() \
                .zscore()

        def mean_reversion_5day_sector_neutral_smoothed(window_length, universe, sector):
            unsmoothed_factor = mean_reversion_5day_sector_neutral(window_length, universe, sect
            return SimpleMovingAverage(inputs=[unsmoothed_factor], window_length=window_length)
                .rank() \
                .zscore()

        class CTO(Returns):
            """
            Computes the overnight return, per hypothesis from
            https://papers.ssrn.com/sol3/papers.cfm?abstract_id=2554010
            """
            inputs = [USEquityPricing.open, USEquityPricing.close]

            def compute(self, today, assets, out, opens, closes):
                """
                The opens and closes matrix is 2 rows x N assets, with the most recent at the bo
                As such, opens[-1] is the most recent open, and closes[0] is the earlier close
                """
                out[:] = (opens[-1] - closes[0]) / closes[0]


        class TrailingOvernightReturns(Returns):
            """
            Sum of trailing 1m O/N returns
            """
            window_safe = True

            def compute(self, today, asset_ids, out, cto):
                out[:] = np.nansum(cto, axis=0)
```

```
def overnight_sentiment(cto_window_length, trail_overnight_returns_window_length, univer
    cto_out = CTO(mask=universe, window_length=cto_window_length)
    return TrailingOvernightReturns(inputs=[cto_out], window_length=trail_overnight_retu
        .rank() \
        .zscore()

def overnight_sentiment_smoothed(cto_window_length, trail_overnight_returns_window_lengt
    unsmoothed_factor = overnight_sentiment(cto_window_length, trail_overnight_returns_w
    return SimpleMovingAverage(inputs=[unsmoothed_factor], window_length=trail_overnight
        .rank() \
        .zscore()

universe = AverageDollarVolume(window_length=120).top(500)
sector = project_helper.Sector()

pipeline = Pipeline(screen=universe)
pipeline.add(
    momentum_1yr(252, universe, sector),
    'Momentum_1YR')
pipeline.add(
    mean_reversion_5day_sector_neutral_smoothed(20, universe, sector),
    'Mean_Reversion_Sector_Neutral_Smoothed')
pipeline.add(
    overnight_sentiment_smoothed(2, 10, universe),
    'Overnight_Sentiment_Smoothed')

all_factors = engine.run_pipeline(pipeline, factor_start_date, universe_end_date)

all_factors.head()
```

1.2 Add sector code

```
In [ ]: pipeline.add(sector, 'sector_code')
```

1.3 Universal Quant Features

- stock volatility: zipline has a custom factor called AnnualizedVolatility. The source code is here and also pasted below:

Annualized volatility. Create `AnnualizedVolatility` objects for 20 day and 120 day (one month and six-month) time windows. Remember to set the `mask` parameter to the `universe` object created earlier (this filters the stocks to match the list in the `universe`). Convert these to ranks, and then convert the ranks to zscores.

```
In [ ]: from zipline.pipeline.factors import AnnualizedVolatility
        volatility_20d = AnnualizedVolatility(window_length=20, mask=universe).rank().zscore()
        volatility_120d = AnnualizedVolatility(window_length=120, mask=universe).rank().zscore()
```