feature_engineering_solution

April 24, 2023

1 Feature Engineering and Labeling

We'll use the price-volume data and generate features that we can feed into a model. We'll use this notebook for all the coding exercises of this lesson, so please open this notebook in a separate tab of your browser.

Please run the following code up to and including "Make Factors." Then continue on with the lesson.

In [ ]:
```
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)
```

Registering data

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 [ ]: 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')
```

2 Make Factors

- We'll use the same factors we have been using in the lessons about alpha factor research. Factors can be features that we feed into the model.

```
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()
```

Stop here and continue with the lesson section titled "Features".

3 Universal Quant Features

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

```
class AnnualizedVolatility(CustomFactor):
    """
    Volatility. The degree of variation of a series over time as measured by
    the standard deviation of daily returns.
    https://en.wikipedia.org/wiki/Volatility_(finance)
    **Default Inputs:** :data:`zipline.pipeline.factors.Returns(window_length=2)`  # noqa
    Parameters
    ----------
    annualization_factor : float, optional
        The number of time units per year. Defaults is 252, the number of NYSE
        trading days in a normal year.
```

```
    """
    inputs = [Returns(window_length=2)]
    params = {'annualization_factor': 252.0}
    window_length = 252

    def compute(self, today, assets, out, returns, annualization_factor):
        out[:] = nanstd(returns, axis=0) * (annualization_factor ** .5)
```

```
In [ ]: from zipline.pipeline.factors import AnnualizedVolatility
        AnnualizedVolatility()
```

Quiz We can see that the returns `window_length` is 2, because we're dealing with daily returns, which are calculated as the percent change from one day to the following day (2 days). The `AnnualizedVolatility` `window_length` is 252 by default, because it's the one-year volatility. Try to adjust the call to the constructor of `AnnualizedVolatility` so that this represents one-month volatility (still annualized, but calculated over a time window of 20 trading days)

Answer

```
In [ ]: # TODO
        AnnualizedVolatility(window_length=20)
```

Quiz: Create one-month and six-month 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 [ ]: # TODO
        volatility_20d = AnnualizedVolatility(window_length=20, mask=universe).rank().zscore()
        volatility_120d = AnnualizedVolatility(window_length=120, mask=universe).rank().zscore()
```

Add to the pipeline

```
In [ ]: pipeline.add(volatility_20d, 'volatility_20d')
        pipeline.add(volatility_120d, 'volatility_120d')
```

Quiz: Average Dollar Volume feature We've been using AverageDollarVolume to choose the stock universe based on stocks that have the highest dollar volume. We can also use it as a feature that is input into a predictive model.
Use 20 day and 120 day `window_length` for average dollar volume. Then rank it and convert to a zscore.

```
In [ ]: """already imported earlier, but shown here for reference"""
        #from zipline.pipeline.factors import AverageDollarVolume

        # TODO: 20-day and 120 day average dollar volume
        adv_20d = AverageDollarVolume(window_length=20, mask=universe).rank().zscore()
        adv_120d = AverageDollarVolume(window_length=120, mask=universe).rank().zscore()
```

Add average dollar volume features to pipeline

```
In [ ]: pipeline.add(adv_20d, 'adv_20d')
        pipeline.add(adv_120d, 'adv_120d')
```

3.0.1 Market Regime Features

We are going to try to capture market-wide regimes: Market-wide means we'll look at the aggregate movement of the universe of stocks.

High and low dispersion: dispersion is looking at the dispersion (standard deviation) of the cross section of all stocks at each period of time (on each day). We'll inherit from CustomFactor. We'll feed in DailyReturns as the `inputs`.

Quiz If the `inputs` to our market dispersion factor are the daily returns, and we plan to calculate the market dispersion on each day, what should be the `window_length` of the market dispersion class?

Answer window_length = 1, because each row of the input data represents returns (not stock prices).

Quiz: market dispersion feature Create a class that inherits from `CustomFactor`. Override the `compute` function to calculate the population standard deviation of all the stocks over a specified window of time.

Calculate the mean returns
$\mu = \sum_{t=0}^{T} \sum_{i=1}^{N} r_{i,t}$
$\sqrt{\frac{1}{T} \sum_{t=0}^{T} \frac{1}{N} \sum_{i=1}^{N} (r_{i,t} - \mu)^2}$
Use numpy.nanmean to calculate the average market return μ and to calculate the average of the squared differences.

```
In [ ]: class MarketDispersion(CustomFactor):
            inputs = [DailyReturns()]
            window_length = 1
            window_safe = True

            def compute(self, today, assets, out, returns):

                # TODO: calculate average returns
                mean_returns = np.nanmean(returns)

                #TODO: calculate standard deviation of returns
                out[:] = np.sqrt(np.nanmean((returns - mean_returns)**2))
```

Quiz Create the MarketDispersion object. Apply two separate smoothing operations using SimpleMovingAverage. One with a one-month window, and another with a 6-month window. Add both to the pipeline.