Dynamic Pair Selection with Machine Learning Clustering, Cointegration Testing, a Stop-Loss Policy and Partial Pair Order Buffering

Matthew Trachevski, Anaf Andalib and Andrew Naughton

University of Melbourne

25 October 2019

Base Strategy Description

Our base strategy is a Quantopian implementation of one of Ernest Chan's pairs trading strategies from his books on algorithmic trading (Quantopian, 2014). The strategy relies on two pairs that both contain stocks from the alternative energy sector that have similar fundamentals and product lines. The pairs consist of Abengoa (ABGB) and First Solar (FSLR), and China Sunergy (CSUN) and Ascent Solar Technologies (ASCI) respectively. The strategy assumes that both pairs of stocks are sufficiently cointegrated to perform well in a mean-reverting trading strategy. Specifically, it assumes that a linear combination of the two stocks in each pair will produce a stationary time series that hovers around a horizontal line (i.e. that mean reverts).

For each pair, the algorithm combines two stock price series in the following way:

```
new_spreads[i, :] = Y[-1] - hedge * X[-1]
```

where `Y[-1]` is the close price of stock Y, `X[-1]` is the close price of stock X, and `hedge` is the coefficient estimate on stock X in the OLS regression of stock Y on stock X and a constant. The algorithm is therefore not simply computing a spread as a straight difference between two stocks in a pair, but as a modified difference equal to one stock price minus a coefficient multiplied by the other stock price. We believe this decision was made to adjust for different scales of stock prices and thereby help express one stock in terms of the other more equally.

The algorithm successfully avoids look-ahead bias by using moving statistics. Rather than computing statistics for the entire sample space, for each point in time, the algorithm looks 20 sample points (days) back and computes statistics over that set of days.

```
context.lookback = 20
```

The algorithm's avoidance of look-ahead bias is crucial. If the spread was computed without a lookback parameter, the spread would estimate the coefficient hedge on all the data, including values in the future that are not available at the time of price checking. This would make the strategy appear a lot more predictive than it actually is. The algorithm makes one final adjustment to the spread by converting it into a z-score using the following formula:

```
zscore = (spreads[-1] - spreads.mean()) / spreads.std()
```

We believe this conversion is performed to preserve the shape of the spread originally computed while making the spread more meaningful in a pairs trading context where thresholds to long or short the spread are generally set one standard deviation above or below the spread's mean. Before sending orders to trade, the algorithm checks that it is not already in a trade. If it is not currently trading and the z-score computed is less than -1 (i.e. the spread is one standard deviation below its mean), the algorithm longs the spread. Taking the spread as `Y[-1] - hedge * X[-1]`, longing the spread means buying 1 unit of stock Y and buying `-hedge` units of stock X (i.e. shorting `hedge` units of stock X). Once the z-score rises above 0, the algorithm closes out its position by setting the target weight for each stock to 0 and becomes available to open a new position. Similarly, if the algorithm is not currently trading and the z-score computed is greater than 1 (i.e. the spread is one standard deviation above its mean), the algorithm shorts the spread. Shorting the spread means selling 1 unit of stock Y and selling `-hedge` units of stock X (i.e. buying `hedge` units of stock X). Once the z-score falls below 0, the algorithm closes out its position by setting the target weight for each stock to 0 and becomes available to open a new position.

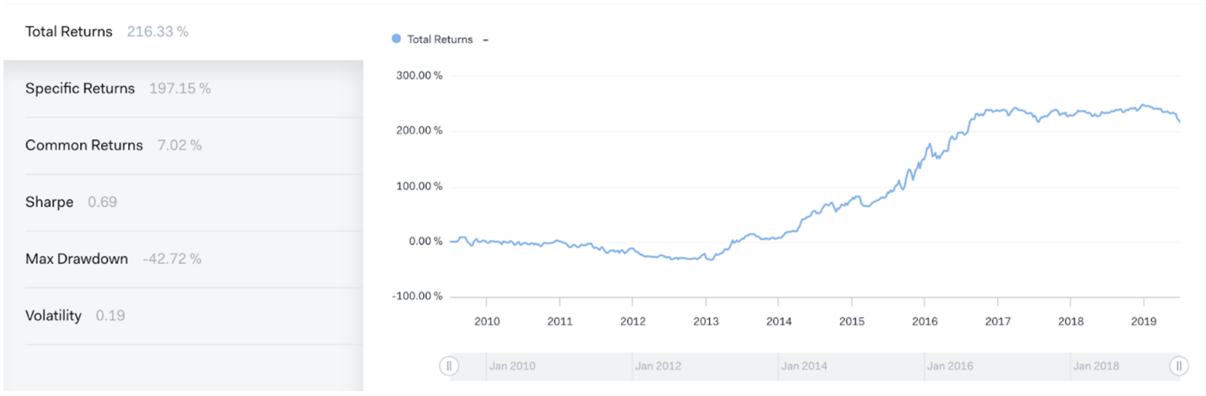
The algorithm constructs portfolios with the aid of Quantopian's Optimize API, which requires an objective and a constraint. The objective tells the Optimize API what the strategy wants from its new portfolio and the constraint gives a condition that the new portfolio must satisfy. The algorithm pursues a `TargetWeights` objective subject to a `MaxGrossExposure(1.0)` constraint. `MaxGrossExposure(1.0)` constrains the total value of the portfolio longs and shorts to be no more than 1 times the current portfolio value. If the target violates this constraint, the optimizer figures out how to adjust the target portfolio as little as possible while ensuring that the constraint is satisfied.

Interestingly, the algorithm only trades during the last 30 minutes of trading each day.

```
schedule_function(func=check_pair_status,
date_rule=date_rules.every_day(),
time_rule=time_rules.market_close(minutes=30))
```

We believe this decision was motivated by the empirical observation that stock markets are most volatile during the first and last 30 minutes of each trading day. Ozenbas et al. (2010) show that the cause of volatility accentuation at the open is driven by the difficulty of translating new information into prices following an extended period of non-trading, and that volatility accentuation at the close is mainly driven by traders who wish to complete their position transactions in an attempt to avoid possible adverse price fluctuations overnight. Similarly, Brock and Kleidon (1992), and Lee et al. (1993) document that the intraday width of bid-ask spreads for New York Stock Exchange stocks follows a U-shaped pattern, where spreads are widest immediately after the open and immediately preceding the close. By trading just before market close each day, this strategy is seeking to profit off larger and more frequent oscillations of its pairs' spreads around their mean.

It was necessary for us to replace two of the base strategy's stocks with close substitutes due to a ticker change and delisting. The substitute stocks were chosen in a way that preserves the intention of the strategy - to generate alpha on the assumption that pairs of stocks in the renewable energy industry will have cointegrating relationships. We replaced Abengoa, whose ticker changed in 2016, with one of its main competitors, Acciona (ANA), a Spanish multinational service provider of renewable energy solutions, and China Sunergy, which delisted in 2016, with one of its main competitors, SunPower (SPWR), an American energy company that designs and manufactures photovoltaic cells and solar panels. In order to perform a compelling analysis, we change the backtest period of our base strategy from July 30, 2012 – January 1, 2014 to July 1, 2009 – July 1, 2019. The more backtesting data that exists for an algorithm, the more confident we can be that the backtest results are a reasonable representation of the algorithm's performance in all market conditions. With these amendments, we backtested the base strategy and generated results shown in Figure 1.



Total Returns	216.33 %
Specific Returns	197.15 %
Common Returns	7.02 %
Sharpe	0.69
Max Drawdown	-42.72 %
Volatility	0.19

Figure 1: Base Strategy Backtest Performance

Potential Weaknesses

We identify five main weaknesses in our selected base strategy.

1. By only trading the two pairs of stocks selected, the strategy ignores other potential pairs of stocks that might be more cointegrated or mean revert with greater frequency. The strategy is, in this sense, lazy, and would likely benefit by identifying the most promising pairs possible.
2. The strategy only trades stocks from the same industry. Although pairs trading is often thought of as a market-neutral and risk-free strategy, it is not completely so, as two or more stocks that are perfectly correlated do not exist. Despite the fact that pairs trading strategies only consider stocks that demonstrate strong co-movements or are highly and positively correlated, they can still benefit from diversification effects as the number of stocks increases, especially when the existing number of stocks is significantly low. Thus, multi-dimensional pairs trading could reduce risk through diversification (Chzee et al., 2016). Gatev et al. (2006), for instance, find that diversification benefits exist from combining multiple pairs in a portfolio. As the number of pairs in a portfolio increases, the portfolio standard deviation falls. The diversification benefits are apparent from the range of realized returns. As the number of pairs in the strategy increases, the minimum realized return also increases, while the maximum realized excess return remains relatively stable.
3. The strategy has a ‘set and forget’ approach to pair selection. The cost of this is that it does not dynamically reassess whether the initial pairs are still strongly cointegrated, or cointegrated at a statistically significant level at all, after a period of time. The strategy will not account for deteriorating cointegration relationships, delistings, or ticker changes, and does not seek to update the pairs it trades accordingly.

4. By entering the market only in the last 30 minutes of trading each day, the strategy misses out on volatile market activity during the first 30 minutes of trading, a period that is potentially more profitable to a mean-reverting strategy. A portion of the intra-day volatility literature suggests that volatility is higher at the opening of trading than the closing. Stoll, H., & Whaley, R. (1990), for example, find that the ratio of variance of open-to-open returns to close-to-close returns is consistently greater than one for NYSE common stocks. They conclude that the greater volatility at the open is attributable to private information revealed in trading and to temporary price deviations induced by specialist traders. Similarly, reverse-J intraday patterns, where volume or volatility ahead of the close remains substantially lower than at the open but higher than for the middle of the trading day, have been reported by Hussain (2011) in DAX index return volatility, and Harju and Hussain (2011) in other European equity indices. McInish and Wood (1992) also provide evidence of a reverse-J pattern in New York Stock Exchange bid-ask spreads.
5. The strategy does not exit positions that fail to revert after a significant time. Cointegration is a long-run relationship, and holds even if two series don't mean revert straight away. However, in a pairs trading context, there is an opportunity cost associated with waiting for this long-run relationship to materialise. If the long-run equilibrium mean-reverting relationship on which profitability is contingent upon does not hold into the future, possibly due to a certain structural change in one of the stocks, the portfolio will face both systematic and firm-specific risks (Alexander and Dimitriu, 2002).

Improvisations

To address our base strategy's weaknesses and enhance performance, we implemented three processes: dynamic pair selection, a stop-loss policy and a partial pair order buffer.

Dynamic pair selection

We select optimal pairs at each predetermined interval according to the following procedure: 1. screen a universe of stocks, 2. cluster stocks based on common underlying factors, and 3. determine the pair in each cluster whose cointegration test produces the lowest p-value.

1. Screening

a) Screen out stocks with a low financial health grade or no grade at all. This is an initial buffer against default risk which could break down a pair's cointegration relationship. We screened out stocks with credit ratings of D and lower.

b) Screen out stocks in the industry group 'Conglomerates' as these companies have no clear dominant revenue and income stream and are not expected to be good members of any pair in the future.

c) Screen out stocks that do not have a full price series over the backtest.

2. Clustering stocks using machine learning methods

Rather than specifying common factors a priori to well-known factors, such as market cap, industry, or P/E ratio, we use principal component analysis (PCA) to reduce the dimensionality of the returns data and extract the historical latent common factor loadings for each stock. We then take these features, add fundamental values 'Market Cap' and 'Financial Health' to make the model more robust, and use the DBSCAN unsupervised clustering algorithm to cluster stocks based on common underlying factors. We choose DBSCAN because unlike KMeans, Agglomerative Clustering and other clustering methods, DBSCAN does not require us to specify the number of clusters and not all samples get clustered. The latter feature is attractive because it makes sense that not every stock will be so closely related to at least one other stock that a viable pair relationship would sustain. It is likely that most stocks are ‘noise’ with respect to this analysis and DBSCAN handles that well.

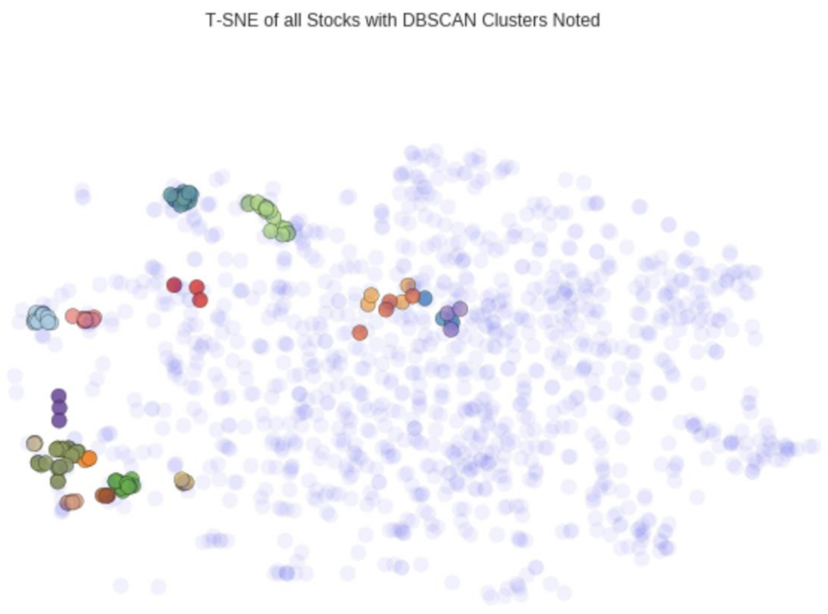


Figure 2: Clusters of similar stocks (August, 2019)

The clustering searches for stocks that have properties which are likely to make them similar and may increase our likelihood of finding pairs within the same sector. The alternative of just looking through all pairs in our screened universe would yield a significant amount of multiple comparison bias which would produce spurious results. The advantage of a clustering technique is that it allows us to perform a first pass that can suggest related baskets, effectively doing dimensionality reduction on features.

3. Cointegration testing

For each cluster, we ascertain each combination of stocks (i.e. each candidate pair), set a significance level of 5%, and run a cointegration test for each pair in each cluster. The cointegration test imported from `Statsmodels` uses the augmented Engle-Granger two-step cointegration test which includes a constant and trend in the first stage. This feature saves us from first detrending our non-stationary price series. Once we have determined the p-value of each pair, we rank the pairs based on lowest significant p-value and determine each cluster's `best_pairs`.

We found it beneficial to only include a limited number of pairs in the algorithm, as the gains and losses of many pairs of stocks offset one another and squashed total returns. The algorithm took `best_pairs` and found the two sets of stocks, `top2_pairs`, that were the most highly co-integrated. This was based on having the smallest p-value amongst all the top pairs from each industry.

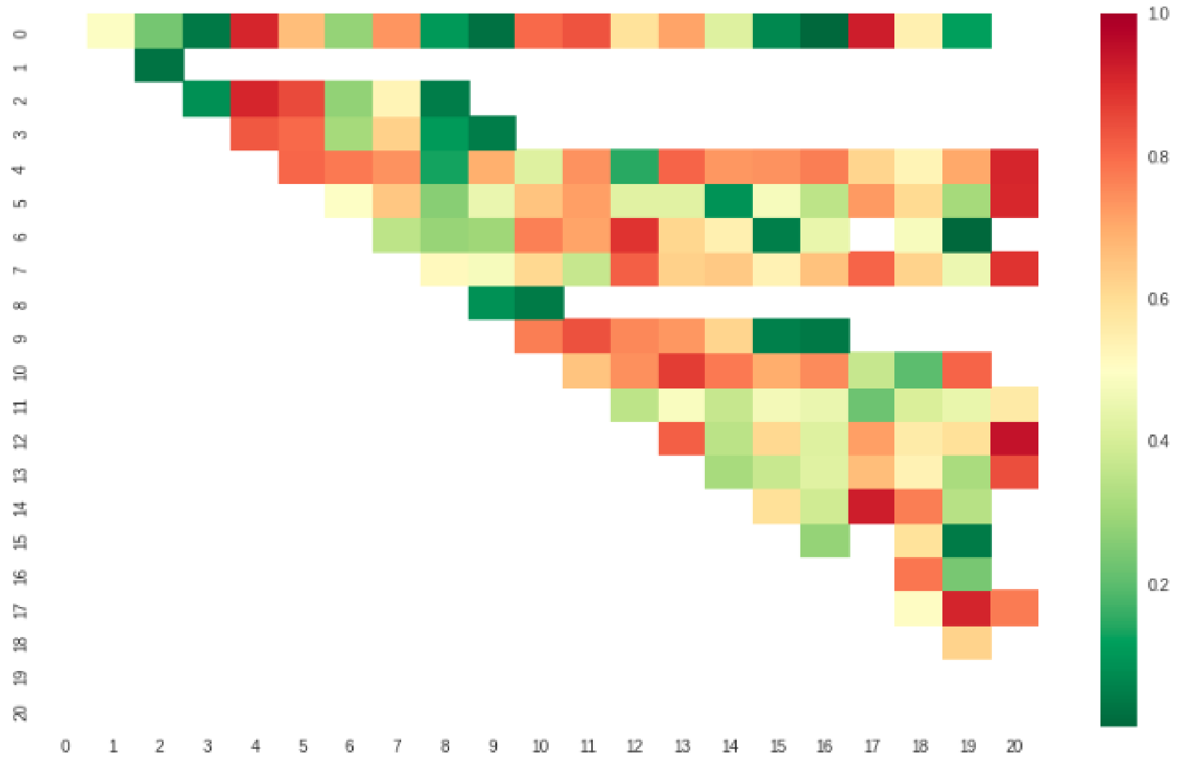


Figure 3: Cointegration heat map for a cluster

Rather than trading these best pairs for the duration of the backtest, we re-clustered every month, and then every year. In doing so, we were able to determine the most cointegrated pair in each sector on a dynamic basis, rather than ‘setting and forgetting’. This was one way we protected against deteriorating cointegration relationships between pairs. If no pairs were found in a given period, our algorithm took no position. During the Great Financial Crisis, for instance, no clusters were formed, suggesting that stocks in our screened universe did not share enough common underlying factors to form clusters. However, by taking no position, our algorithm did, in a sense, still take a position. In fact, by not trading immediately after this crisis period, our algorithm may have avoided negative returns incurred by many traders and investors around the recession.

Stop-loss policy

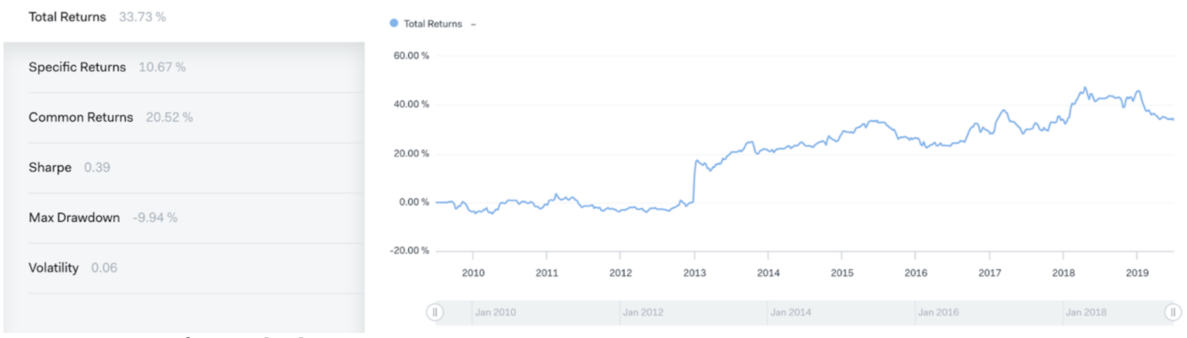
We implemented a stop-loss function that is called each time the algorithm re-clustered. Our stop-loss function aimed to address one of the weaknesses in the base strategy; namely, a lack of a safety measure to stop an (unrealised) loss that a stock pair incurs when it fails to mean revert after an extended period of time. If the opportunity cost of not trading a more profitable pair outweighs the cost of realizing a loss from an existing non-mean-reverting pair, then it might be optimal to close out the position early. Thus, we specified a ‘degree of impatience’ which could be altered to gauge the average time required by pairs to mean-revert. This initially took the form of a separate function; however, we found it efficient to equate our degree of impatience with our re-clustering period. By allowing best pairs to trade for a year, we found that spreads were able to mean revert at least once, which was not always the case when the degree of impatience was higher (i.e. with monthly re-clustering). This approach was largely improvisatory, owing to the lack of empirical understanding of the average cointegration breakdown time of equity pairs.

Partial Pair Order Buffering

One of the key shortcomings of our base strategy is that it fails to account for cases where an order to buy or sell a pair is only partially filled (i.e. where only one order for a stock in a pair is filled). Our solution to this problem was to "catch" all open positions where at least one of the stocks, which we thought we had bought or sold successfully, in reality, had a portfolio 'weight' equal to zero. This improvisation enabled the algorithm to avoid being left with a single long or short position by immediately closing out un-paired positions, and preserve its focus on trading pairs.

Results

We received mixed results from our implementation of our different improvisations. The spillover effects of one improvisation into another proved difficult to track, as the intention of these enhancements were often unrelated. For instance, when we increased our stop-loss/re-clustering period from one month to one year, our total returns and Sharpe ratio increased significantly. However, when we allowed the algorithm to trade during the first 30 minutes of trading rather than the last 30, these improvements were washed away. We found that by adjusting the z-score of a pair's spread relationship up from 1 to 2, performance diminished. This might have been due to the algorithm failing to trade frequently enough to profit off volatile market conditions over the backtest. When we lowered the z-score to 0.5, kept our stop-loss/re-clustering period at 1 year, and implemented our Partial Pair Order Buffer, we generated our best performing backtest, which produced total returns of 33.73% over the 10 year sample and a Sharpe ratio of 0.39, as shown in Figure 4.



Total Returns	33.73 %
Specific Returns	10.67 %
Common Returns	20.52 %
Sharpe	0.39
Max Drawdown	-9.94 %
Volatility	0.06

Figure 4: Best performing backtest

Interestingly, increasing volatility by including stocks with low credit ratings worsened backtest performance. One possible explanation for this is that the benefits of more frequent stock oscillations were offset by the costs of weaker cointegration relationships.

Conclusions and Implications

In this report, we have introduced a pairs trading algorithm inspired by a well-known figure in algorithm trading, identified its shortcomings, and offered possible enhancements. Our results suggest that the interaction of our different improvisations have mixed effects on backtest performance, which indicates that these enhancements are likely to lead to unpredictable performance out-of-sample. Future studies might build on our approach by screening out low-volatility and momentum stocks that do not mean revert reliably, trading during both market open and close, and implementing a secondary verification step in our dynamic pair selection process to ensure clusters include stocks in the same industry.

References

Alexander, C. and A. Dimitriu, 2002, The Cointegration Alpha: Enhanced Index Tracking and Long-Short Equity Market Neutral Strategies, working paper (ISMA Center of the University of Reading, UK)

Brock, W., and Kleidon, A. 1992. Periodic market closure and trading volume: A model of intraday bids and asks. Journal of Economic Dynamics and Control 16:451-89.

Chzee, L., Wenjun, X., & Yuan, W. (2016). Multi-Dimensional Pairs Trading Using Copulas. EFM Classification Codes & Research Area: 370 - Portfolio Management and Asset Allocation.

Gatev, Evan, Goetzmann, William N. and Rouwenhorst, K. Geert (2006), "Pairs Trading: Performance of a Relative Value Arbitrage Rule", The Review of Financial Studies, Vol. 19, No. 3, pp. 797-827

Harju, K., Hussain, S.M., 2011. Intraday Seasonalities and Macroeconomic news announcements. European Financial Management, 17, 2, 367-390.

Hussain, S.M., 2011. The Intraday Behaviour of Bid-Ask Spreads, Trading Volume and Return Volatility: Evidence from DAX30. International Journal of Economics and Finance, 3, 1, 23-34.

Lee, C., Mucklow, B., and Ready, M. 1993. Spreads, depths, and the impact of earnings information: An intraday analysis. Review of Financial Studies 6, no. 2:345-74.

McInish, T.H., Wood, R.A., 1992. An analysis of intraday patterns in Bid-Ask spreads for NYSE stocks. Journal of Finance, 47, 753-764.

Ozenbas, D., Pagano, M. S., & Schwartz, R. A. (2010). Accentuated Intraday Stock Price Volatility: What Is the Cause? Journal of Portfolio Management, (3), 45.

Quantopian Lecture 46: Pairs Trading Algorithm [Blog post]. Retrieved October 8, 2019, from https://www.quantopian.com/lectures/example-pairs-trading-algorithm

Stoll, H., & Whaley, R. (1990). Stock Market Structure and Volatility. Review of Financial Studies, 3(1), 37–71.

.

Appendix

Final Algorithm Code

```
# Algorithmic Trading Project 2
# Pairs Trading with Machine Learning

# Code by Anaf Andalib (ID: 916696), Andrew Naughton (ID: 910691), and Matt Trachevski
(ID: 913128)

# referenced from https://www.quantopian.com/posts/pairs-trading-with-machine-learning
# Contains all code in order to make relevant diagrams and graphs


import statsmodels.api as sm
import numpy as np
import pandas as pd
import datetime

import quantopian.optimize as opt
import quantopian.algorithm as algo

from sklearn.cluster import KMeans, DBSCAN
from sklearn.decomposition import PCA
from sklearn.manifold import TSNE
from sklearn import preprocessing

from statsmodels.tsa.stattools import coint

from scipy import stats

from quantopian.pipeline.data import morningstar
from quantopian.pipeline.filters.morningstar import Q500US, Q1500US, Q3000US
from quantopian.pipeline import Pipeline
from quantopian.algorithm import attach_pipeline, pipeline_output

UNIVERSE = Q1500US()
INITIAL_TIME = '2009-07-01'
N_PRIN_COMPONENTS = 50
CLUSTER_SIZE_LIMIT = 9999
N_TOP_PAIRS = 1

def initialize(context):
    # Quantopian backtester specific variables
    set_slippage(slippage.FixedSlippage(spread=0))
    set_commission(commission.PerTrade(cost=1))
    set_symbol_lookup_date('2009-07-01')

    context.stock_pairs = []
    context.stocks = []

    context.num_pairs = len(context.stock_pairs)

    schedule_function(func=get_clusters, date_rule= date_rules.month_start(),
time_rule=time_rules.market_open(minutes=1))


    attach_pipeline(make_pipeline(), 'pipeline')

    # strategy specific variables
    context.lookback = 20 # used for regression
    context.z_window = 20 # used for zscore calculation, must be <= lookback

    context.target_weights = pd.Series(index=context.stocks, data=0.25)

    context.spread = np.ndarray((context.num_pairs, 0))
    context.inLong = [False] * context.num_pairs
    context.inShort = [False] * context.num_pairs

    # Only do work 30 minutes before close
    schedule_function(func=check_pair_status, date_rule=date_rules.every_day(),
time_rule=time_rules.market_close(minutes=30))


# Will be called on every trade event for the securities you specify.
def handle_data(context, data):
    # Our work is now scheduled in check_pair_status
    pass

def check_pair_status(context, data):

    prices = data.history(context.stocks, 'price', 20, '1d').iloc[-context.lookback::]

    new_spreads = np.ndarray((context.num_pairs, 1))

    for i in range(context.num_pairs):

        (stock_y, stock_x) = context.stock_pairs[i]
        Y = prices[stock_y]
        X = prices[stock_x]

        # Comment explaining try block
        try:
            hedge = hedge_ratio(Y, X, add_const=True)
        except ValueError as e:
            log.debug(e)
            return
        context.target_weights = get_current_portfolio_weights(context, data)

        new_spreads[i, :] = Y[-1] - X[-1]

        if context.spread.shape[1] > context.z_window:
            # Keep only the z-score lookback period
            spreads = context.spread[i, -context.z_window:]

            zscore = (spreads[-1] - spreads.mean()) / spreads.std()

            # liquidate/close out position if there's been a partially/fully unfilled order
            if context.inShort[i] and (not context.target_weights[stock_x] or not
context.target_weights[stock_y]):
                context.target_weights[stock_y] = 0
                context.target_weights[stock_x] = 0

                context.inShort[i] = False
                context.inLong[i] = False

                record(X_pct=0, Y_pct=0)
                allocate(context, data)
                return

            # liquidate/close out position if there's been a partially/fully unfilled order
            if context.inLong[i] and (not context.target_weights[stock_x] or not
context.target_weights[stock_y]):
                context.target_weights[stock_y] = 0
                context.target_weights[stock_x] = 0

                context.inShort[i] = False
                context.inLong[i] = False

                record(X_pct=0, Y_pct=0)
                allocate(context, data)
                return

            if context.inShort[i] and zscore < 0.0:
                context.target_weights[stock_y] = 0
                context.target_weights[stock_x] = 0

                context.inShort[i] = False
                context.inLong[i] = False

                record(X_pct=0, Y_pct=0)
                allocate(context, data)
                return

            if context.inLong[i] and zscore > 0.0:
                context.target_weights[stock_y] = 0
                context.target_weights[stock_x] = 0
                context.inShort[i] = False
                context.inLong[i] = False

                record(X_pct=0, Y_pct=0)
                allocate(context, data)
                return

            if zscore < -0.5 and (not context.inLong[i]):
                # Only trade if NOT already in a trade
                y_target_shares = 1
                X_target_shares = -1
                context.inLong[i] = True
                context.inShort[i] = False

                (y_target_pct, x_target_pct) =
computeHoldingsPct(y_target_shares,X_target_shares, Y[-1], X[-1])

                context.target_weights[stock_y] = y_target_pct * (1.0/context.num_pairs)
                context.target_weights[stock_x] = x_target_pct * (1.0/context.num_pairs)

                record(Y_pct=y_target_pct, X_pct=x_target_pct)
                allocate(context, data)
                return

            if zscore > 0.5 and (not context.inShort[i]):
                # Only trade if NOT already in a trade
                y_target_shares = -1
                X_target_shares = 1
                context.inShort[i] = True
                context.inLong[i] = False

                (y_target_pct, x_target_pct) = computeHoldingsPct( y_target_shares,
X_target_shares, Y[-1], X[-1] )

                context.target_weights[stock_y] = y_target_pct * (1.0/context.num_pairs)
                context.target_weights[stock_x] = x_target_pct * (1.0/context.num_pairs)

                record(Y_pct=y_target_pct, X_pct=x_target_pct)
                allocate(context, data)
                return

    context.spread = np.hstack([context.spread, new_spreads])

def hedge_ratio(Y, X, add_const=True):
    if add_const:
        X = sm.add_constant(X)
        model = sm.OLS(Y, X).fit()
        return model.params[1]
    model = sm.OLS(Y, X).fit()
    return model.params.values

def computeHoldingsPct(yShares, xShares, yPrice, xPrice):
    yDol = yShares * yPrice
    xDol = xShares * xPrice
    notionalDol =  abs(yDol) + abs(xDol)
    y_target_pct = yDol / notionalDol
    x_target_pct = xDol / notionalDol
    return (y_target_pct, x_target_pct)

def get_current_portfolio_weights(context, data):
    positions = context.portfolio.positions
    positions_index = pd.Index(positions)
    share_counts = pd.Series(
        index=positions_index,
        data=[positions[asset].amount for asset in positions]
    )

    current_prices = data.current(positions_index, 'price')
    current_weights = share_counts * current_prices / context.portfolio.portfolio_value
    return current_weights.reindex(positions_index.union(context.stocks), fill_value=0.0)

def allocate(context, data):
    # Set objective to match target weights as closely as possible, given constraints
    objective = opt.TargetWeights(context.target_weights)

    # Define constraints
    constraints = []
    constraints.append(opt.MaxGrossExposure(1.0))


    algo.order_optimal_portfolio(
        objective=objective,
        constraints=constraints,
    )

# Primary function for ML clustering, takes current data and sets the trading
# algorithms focus onto the top two pairs which are highly co-integrated
def get_clusters(context, data):

    sim_date = get_datetime()
    months = [1, 2, 3, 4, 5, 6, 8, 9, 10, 11, 12]

    # Only run if no pairs have been found/ if a year cycle has passed (in July)
    if sim_date.month not in months or not context.stock_pairs:
        print("I AM FINDING THE BEST PAIRS")
    else:
        print("IM NOT RUNNING")
        return
    # Close all open positions before finding potentially new stocks
    close_all_open(context, data)

    results = pipeline_output("pipeline")
    res = manipulate_results(results)

    price_history = data.history(res.index,
fields="close",                                     bar_count=year_to_day(2), frequency="1d")

    # print("R/C: {0}".format(price_history.shape))

    returns = price_history.pct_change()
    # print(returns.shape)

    # we can only work with stocks that have the full return series
    returns = returns.iloc[1:,:].dropna(axis=1)
    # print(returns.shape)

    # PCA on returns
    pca = PCA(n_components=N_PRIN_COMPONENTS)
    pca.fit(returns)

    X = np.hstack(
        (pca.components_.T,
         res['Market Cap'][returns.columns].values[:, np.newaxis],
         res['Financial Health'][returns.columns].values[:, np.newaxis])
    )

    # As euclidean is sensitive to magnitude, we need to scale the data
    # such that the algorithm works optimally
    X = preprocessing.StandardScaler().fit_transform(X)

    clf = DBSCAN(eps=1.9, min_samples=3)

    clf.fit(X)
    labels = clf.labels_
    n_clusters_ = len(set(labels)) - (1 if -1 in labels else 0)
    print "\nClusters discovered: %d" % n_clusters_

    clustered = clf.labels_

    clustered_series =
pd.Series(index=returns.columns,                                    data=clustered.flatten()
)
    clustered_series_all =
pd.Series(index=returns.columns,                                    data=clustered.flatten()
)
    clustered_series = clustered_series[clustered_series != -1]
    # the initial dimensionality of the search was
    ticker_count = len(returns.columns)
    # print "Total pairs possible in universe: %d " % (ticker_count*(ticker_count-1)/2)

    counts = clustered_series.value_counts()
    ticker_count_reduced = counts[(counts>1) & (counts<=CLUSTER_SIZE_LIMIT)]
    print "Clusters formed: %d" % len(ticker_count_reduced)
    print "Pairs to evaluate: %d" % ((ticker_count_reduced*(ticker_count_reduced-1)).sum() /
2)

    cluster_dict = {}
    for i, which_clust in enumerate(ticker_count_reduced.index):
        tickers = clustered_series[clustered_series == which_clust].index
        score_matrix, pvalue_matrix, pairs, top_pairs = find_cointegrated_pairs(
            price_history[tickers]
        )
        cluster_dict[which_clust] = {}
        cluster_dict[which_clust]['score_matrix'] = score_matrix
        cluster_dict[which_clust]['pvalue_matrix'] = pvalue_matrix
        cluster_dict[which_clust]['pairs'] = pairs

        cluster_dict[which_clust]['top_pairs'] = top_pairs

    pairs = []
    best_pairs = []
    for clust in cluster_dict.keys():
        pairs.extend(cluster_dict[clust]['pairs'])
        best_pairs.extend(cluster_dict[clust]['top_pairs'])

    print("We found {0} cointegrated pairs and {1} pairs from different
clusters".format(len(pairs), len(best_pairs)))

    if len(best_pairs) >= 2:
        top2_pairs = sorted(best_pairs)[:2]
    else:
         top2_pairs = best_pairs

    # New stocks to trade
    pair_symbols = [item[1] for item in top2_pairs]
    stocks_list = [stock for pair in pair_symbols for stock in pair]

    context.stock_pairs = pair_symbols
    context.stocks = stocks_list
    context.num_pairs = len(context.stock_pairs)
    num_stocks = len(stocks_list)
    if num_stocks:
        weight = 1.0/num_stocks
        context.target_weights = pd.Series(index=context.stocks, data=weight)

    context.spread = np.ndarray((context.num_pairs, 0))
    context.inLong = [False] * context.num_pairs
    context.inShort = [False] * context.num_pairs

def year_to_day(year):
    #year_to_day(3)
    return year * 365


def make_pipeline():
    """
    A pipeline to create our dynamic stock selector (pipeline). Documentation
    on pipeline can be found here:
    https://www.quantopian.com/help#pipeline-title
    """

    pipe = Pipeline(
        columns= {
            'Market Cap': morningstar.valuation.market_cap.latest.quantiles(5),
            'Industry':                                  morningstar.asset_classification.morningstar
_industry_group_code.latest,
            'Financial Health': morningstar.asset_classification.financial_health_grade.latest
        },
        screen=UNIVERSE
    )

    print("Pipe has been made")
    return pipe


def manipulate_results(results):
    # remove stocks in Industry "Conglomerates"
    res = results[results['Industry']!=31055]

    # remove stocks without a Financial Health grade
    res = res[res['Financial Health']!= None]

    # Removes stocks which have a highly imminent default risk
    res = res[res['Financial Health'] != 'D']
    res = res[res['Financial Health'] != 'F']

    print ("new dimensions of results: {0}".format(res.shape))

    # replace the categorical data with numerical scores per the docs
    # Leaving in VALUES: D and F in case algorithm needs to be changed
    res['Financial Health'] = res['Financial Health'].astype('object')
    health_dict = {u'A': 0.1,
                   u'B': 0.3,
                   u'C': 0.7,
                   u'D': 0.9,
                   u'F': 1.0}
    res = res.replace({'Financial Health': health_dict})

    return res

def find_cointegrated_pairs(data, significance=0.05):
    # This function is from https://www.quantopian.com/lectures/introduction-to-pairs-trading
    # Has been altered to inclue the top N pairs in each cluster, we only need 1
    n = data.shape[1]
    score_matrix = np.zeros((n, n))
    pvalue_matrix = np.ones((n, n))
    keys = data.keys()
    pairs = []
    top_pairs = []
    for i in range(n):
        for j in range(i+1, n):
            S1 = data[keys[i]]
            S2 = data[keys[j]]
            result = coint(S1, S2)
            score = result[0]
            pvalue = result[1]
            score_matrix[i, j] = score
            pvalue_matrix[i, j] = pvalue
            if pvalue < significance:
                pairs.append((keys[i], keys[j]))
                if len(top_pairs) == N_TOP_PAIRS:
                    for i in range(len(top_pairs)):
                        # if new pair has a higher coint test e.g. a lower pval
                        # then replace the old pair
                        if top_pairs[i][0] > pvalue:
                            top_pairs.pop(i)
                            top_pairs.append((pvalue, (keys[i], keys[j])))
                            # sort so highest coint is always first (lowest p)
                            top_pairs.sort()

                elif len(top_pairs) < N_TOP_PAIRS:
                    top_pairs.append((pvalue, (keys[i], keys[j])))
                    top_pairs.sort()

    return score_matrix, pvalue_matrix, pairs, top_pairs

# Closes all current open positions, wiping the slate
def close_all_open(context, data):

    print("CURRENTLY CLOSING ALL OPEN POSITIONS FOR PREV STOCKS")

    for i in range(context.num_pairs):
        (stock_y, stock_x) = context.stock_pairs[i]

        if context.inShort[i]:
            print("I HAVE OPEN SHORT")
            context.target_weights[stock_y] = 0
            context.target_weights[stock_x] = 0

            context.inShort[i] = False
            context.inLong[i] = False

            record(X_pct=0, Y_pct=0)
            allocate(context, data)

        if context.inLong[i]:
            print("I HAVE OPEN LONG")
            context.target_weights[stock_y] = 0
            context.target_weights[stock_x] = 0

            context.inShort[i] = False
            context.inLong[i] = False

            record(X_pct=0, Y_pct=0)
            allocate(context, data)

    # Checks to see if above iterations worked
    for i in range(context.num_pairs):
        (stock_y, stock_x) = context.stock_pairs[i]

        if context.inShort[i]:
            print("I HAVE OPEN SHORT BUT I SHOULD NOT")

        if context.inLong[i]:
            print("I HAVE OPEN LONG BUT SHOULD NOT")
```