Two Machine Learning Approaches for Statistical Arbitrage Pairs Selection

Brian Chan (bchan17), Nhi Truong (ntruongv), Carina Zhang (carinaz)

1 Introduction

Statistical arbitrage is an algorithmic trading approach based on the assumption that there exists inefficiency in pricing in the financial markets. We will focus on a simple but effective statistical arbitrage strategy called pairs trading [1]. If A and B are two stocks that have similar characteristics, then we can expect that their returns have similar temporal trend. Based on this assumption, one can model their relationship linearly as follows:

$$\frac{dA_t}{A_t} = \alpha dt + \beta \frac{dB_t}{B_t} + dX_t, \tag{1}$$

where the residual process X_t can be interpreted as the return spread between stock A and B. This process is referred as the OU Process. When X_t is large positive, then stock A is overpriced, and when it is large negative, stock B is overpriced. Since the market is assumed to drift towards the efficient equilibrium, X_t is suggested to be a (mean-reverting) Ornstein - Uhlenback process. Moreover, this suggests a long-short arbitrage strategy: when $X_t \gg 0$, we long one dollar in stock A and short β in stock B, and vice versa when $X_t \ll 0$. This strategy does not depend on absolute stock price because it is market neutral. Rather, the burden of the strategy is on: (1) finding a good pair of stock, and (2) modeling X_t well to open/close position with precision.

There have been multiple researches as well as CS229 projects on modeling X_t once fixing two stocks, often from the same industry (for e.g.: APPL- GOOGL) [3, 11, 9]. There's also a related paper on clustering but on stocks and ETF [7]. In this paper, we deployed two different approaches using machine learning methods in order to identify optimal stock pairs.

2 Method

2.1 Approach 1: Factor Model

An important insight from arbitrage pricing theory is that a stock's return is driven by its exposure to risk factors in the market. One way to extract these risk factors purely from return data is to use a latent factor model:

$$\underbrace{X}_{N\times T} = \underbrace{\Lambda}_{N\times L}\underbrace{F}_{L\times T} + \underbrace{e}_{N\times T}. \tag{2}$$

Here, X is a matrix of returns of N stocks through a period of T days. F is the matrix of returns of K latent risk factors, Λ is the matrix of loading and e the residual.

Under the model (2), each stock in the matrix X is now associated to an $\mathbb{R}^L$ vector that represents its correlation with the latent factors. Two stocks that have similar loadings are likely to have similar return movements. Further, when choosing $L \ll N$, we can achieve dimensionality reduction.

2.1.1 PCA

To extract the loadings Λ and factors F in model (2), we apply PCA to the sample covariance matrix XX^T.[1] The loadings Λ is constructed from the eigenvectors associated with the L largest eigenvalues. The factor F is then given by regressing the data X on the estimated loading Λ.

[1] It is proven that as $N, T \to \infty$, PCA chosen factors converge, up to a rotation, the real factors. Since we use the loadings for doing K-means clustering, a rotation in space should not affect final result.

2.1.2 K-means Clustering

Given that each stock is now represented by a vector in L-dimensional space, we then apply K-means clustering method learned in class in order to get groups of stocks that have similar characteristics.

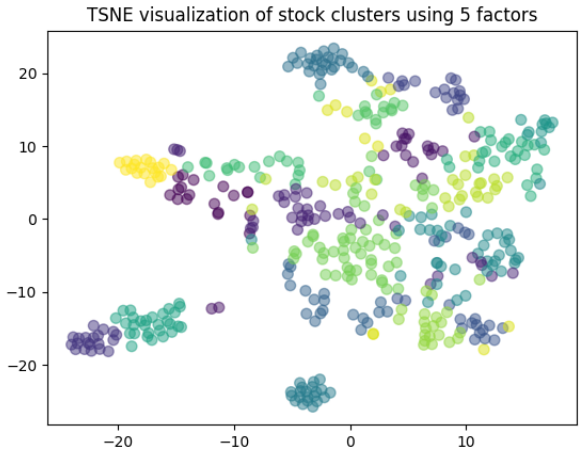

Figure 1: Stock clusters using Factor Models

2.2 Approach 2: Convolutional AutoEncoder (CAE)

Recently, there are a number of researches focusing on representing time series data as images in order to exploit advances in computer vision [5, 6, 10]. Many of these aim to classify time series data or to forecast. Inspired by this, we converted time series to images and use a CAE to learn features for clustering.

2.2.1 Gramian Angular Field

First, we scaled the stock time series data so that it lies in $[-1, 1]$:

$$\tilde{S}_t = \frac{2S_t - \max_i S_i - \min_i S_i}{\max_i S_i - \min_i S_i}$$

Then, we use the embed the 1D time series into 2D polar coordinate space by

$$\phi_t = \arccos(\tilde{S}_t), \quad r_t = \frac{t}{T},$$

where T is the length of the time series. The GAF is defined as

$$G = \begin{pmatrix} \cos(\phi_1 + \phi_1) & \cdots & \cos(\phi_1 + \phi_T) \\ \vdots & \ddots & \vdots \\ \cos(\phi_T + \phi_1) & \cdots & \cos(\phi_T + \phi_T). \end{pmatrix}$$

We can see that this matrix contains information about temporal correlation of our series since it can be viewed as a Gramian matrix.

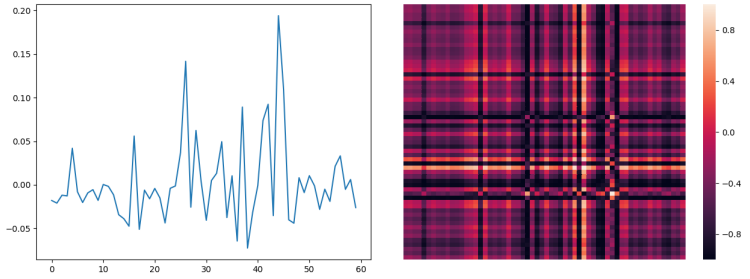

Figure 2: GAF of a return series

2.2.2 CAE architecture

We applied the CAE proposed by [4], as the authors showed that their proposed architecture excels in identifying clusters on the MNIST dataset. The encoders involves 3 convolutional layers to extract hierarchical features, ending with a flattened vector, which is inputted into a dense layer that outputs to a low-dimensional space. The decoders involve tranposes of the encoders.

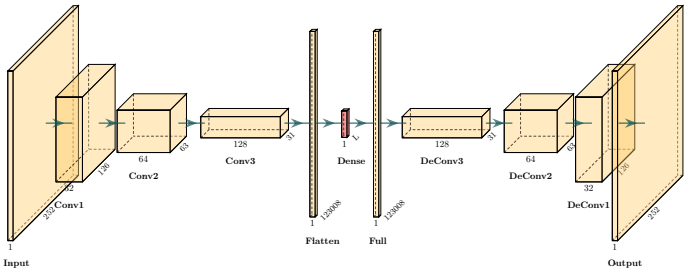

Figure 3: CAE Architecture

Note that the red layer compresses our data to $\mathbb{R}^L$ dimensional space. The original paper [4] suggested using the number of features $L = K$, the number of clusters. In our case, we treat this as a hyperparameter to tune. We use the features obtained from this layer to do K-means clustering.

2.3 Stock Screening with ARIMA

After clustering stocks into groups using either approach 1 or 2, we would like to choose the top pairs from each group. Given a pair of stock, we use linear regression on return data to fit α, β and find the residual dX_t in equation (1). Note that a pair will do well in statistical arbitrage if the residual can be modelled as an OU process:

$$dX_t = \kappa(m - X_t)dt + \sigma dW_t. \tag{3}$$

This can be discretized and modelled as an AR(1) process:

$$X_{n+1} = a + bX_n + \zeta_{n+1}$$

$$b = e^{-\kappa\Delta_t},\ a = m(1-b),\ \text{var}(\zeta) = \sigma^2\frac{1-b^2}{2\kappa}$$

Given the clusters of stocks, we fit their residual dX_t (gotten from linear regression) with an AR(1) process using Kalman filter [3]. Notice that $b \geq 0$ for OU processes, and since we want the process to be stationary, we only accept pairs with $0 < b < 1$, and p-value less than 0.05.[2]

2.4 Trading Phase

We applied the standard pairs trading procedure established in [1]. We use a rolling window approach, where we use 60 days of historical data to estimate the AR(1) parameters for a spread between any pair. Then, we use that to generate a trading signal:

$$s = \frac{X_{60} - m}{\sigma}. \tag{4}$$

This is the number of standard deviation away from the long-term mean of the OU-process of the price we observe today. We use the rule: $s_{\text{long, open}} = -1.25$, $s_{\text{long, close}} = -0.5$, $s_{\text{short open}} = 1.25$, $s_{\text{short,close}} = 0.75$.

Here, long, open means that we buy a unit of stock A and short β units of stock B. Short, open means the opposite. Close means closing any open positions.

3 Experiments and Results

3.1 Data

We obtained stock price data for the 500 companies appearing in the S&P500 index during the years of 2010 and 2011. We converted daily price data into daily return data, and only include the stocks with no missing value in both years (467 stocks). We split the data into 2010 dataset and 2011 dataset. We used the 2010 dataset to train our model and the 2011 dataset to test our model.

3.2 Metrics

Since we have an unsupervised problem and because our overall goal is to create a profitable trading strategy, the Sharpe ratio (SR) (mean return over standard deviation) is chosen as the evaluation metrics. Our null hypothesis is that pairs chosen from our strategy has the same SR compared to pairs chosen at random. The goal is to produce a p-value that rejects this null hypothesis at the 5% level.

To compute p-value, we generated 10,000 random pairs of stocks and ran our strategy on them to get SR. We then do 100,000 samplings, each time of a number of stocks equal to the number of stocks picked by our screening. For each sample, we computed the mean SR. This gives us an empirical distribution for the mean SR of a random group of pairs of stocks (size equal to our screened group), from which we compute p-value.

3.3 Factor Model Approach

3.3.1 Hyperparameter Tuning

The first obstacle is in choosing the number of latent factors when performing PCA. We tested 3 different methods:

1. **Threshold**: In economics literature, the benchmark is to choose enough factors so as to explain 55% of variance. This gives **11 factors**.

[2] A popular method used in screen pairs is called the cointegration test (two times series are cointegrated if they are not stationary but some linear combination is). This is a sound strategy on price data, but return data is often stationary, so this test is not reliable.

2. **Information Criterion(IC)**: [2] developed a type of IC for picking the number of factors. This gives **2 factors**.
3. **Random Matrix Theory** : [8] proposed using the fact that the population distribution of eigenvalues $\lambda_1 \geq \cdots \geq \lambda_N$ of the covariance matrix in a factor model will be such that $\lambda_{K+1}, \ldots, \lambda_N$ are small and cluster together. This gives **5 factors**.

We also need to choose K, the number of clusters. For this, we ran our procedure for a number of different values of K and obtain Figure 4.

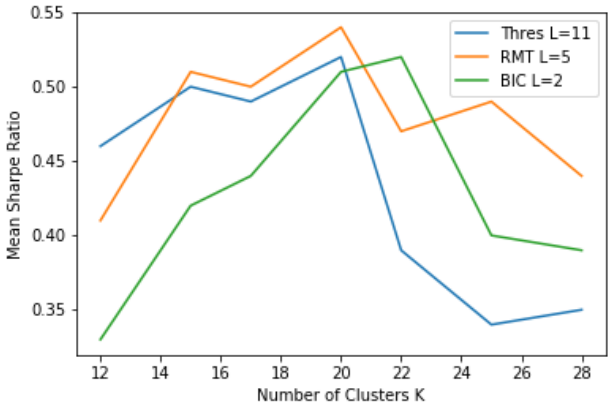

Figure 4: Hyperparameter tuning in Factor Model

In the clustering stage, the number of factors based on threshold, RMT, and IC methods are 11, 5, and 2 respectively. 11 factors result in overfitting while 2 factors result in underfitting. Note that when overfitting occurs in the clustering stage, it means that we have looser rules to group stocks into one cluster. When stocks in one cluster are less similar to each other, the SR in the trading strategy should be lower, which is what we have observed. Same reasoning goes for underfitting. Hence, across the board, we see that the mean SR from RMT method is always higher, both in training and in testing.

Similar reasoning applies for the number of clusters, and we can see here that the optimal number of clusters K is about 20-22. Higher and lower K leads to lower mean SR.

3.3.2 Performance

As suggested by Figure 4, we chose to use $L = 5$ factors (given by RMT approach) and $K = 20$ clusters.

Method	Metrics	Train	Test
Factor Model and PCA	Mean SR	0.54	0.33
	p-value	10^{-6}	0.009
	Beta	0.015	0.009
	Alpha	0.091	0.079
Baseline (Industry)	Mean SR	0.18	0.17
	p-value	0.34	0.31
	Beta	0.012	0.010
	Alpha	0.023	0.011

Table 1: Performance of Factor Model Approach

In comparison to randomly picking stock pairs but with the only constraint that stocks in the same pair have to be in the same industry, our method clearly performs way better in terms of both mean SR and p-value. Further, we reject the null hypothesis that our model picks pairs similar to random selection at the 1% level, while we do not reject such hypothesis for the baseline.

Since the statistical arbitrage strategy is market neutral, the beta values are all low, approximately 0.01. On the other hand, we have relatively high alpha of about 8% for the factor model approach (which also outperforms the baseline).

3.4 CAE

3.4.1 Hyperparameter tuning

We need to choose the number of features we want to use for clustering, which is the number of neurons in the dense layer (red layer in Figure 3). We also denote this by L to parallel with the previous approach. Similarly, we have to determine the number of clusters K as well. We ran our procedures for different values of L and K and obtain Figure 5

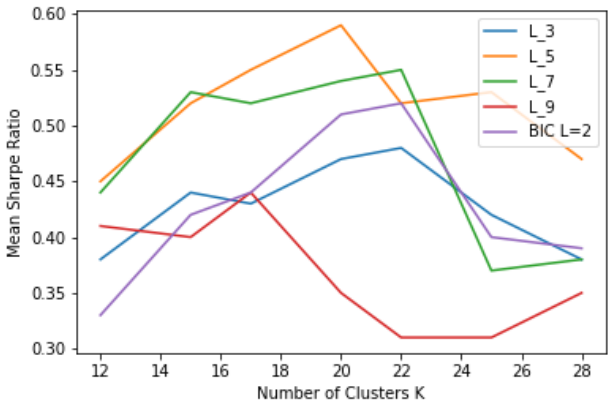

Figure 5: Hyperparameter tuning in CAE

We again have that (quite surprisingly), the optimal number of factors is still $L = 5$ and the optimal number of clusters $K = 20$.

There are other hyperparameters that we have not found time to tune, for example the number of epochs and batch sizes for the CAE, which we set by default at 20 and 200.

3.4.2 Performance

Method	Metrics	Train	Test
CAE	Mean SR	0.59	0.39
	p-value	$< 10^{-6}$	0.003
	Beta	0.013	0.014
	Alpha	0.15	0.091
Baseline (Industry)	Mean SR	0.18	0.17
	p-value	0.34	0.31
	Beta	0.012	0.010
	Alpha	0.023	0.011

Table 2: Performance of CAE Approach

Again, we can see that the CAE approach to picking pair outperforms the baseline of picking stocks by industry in all aspects: mean SR, p-value and alpha. As previously, the strategy is market neutral so beta is very low.

3.5 Approach Comparison

Although the CAE approach is much more complicated, its performance is only slightly better than the factor model with PCA approach, as can be seen by comparing Table 1 and Table 2. This could be because we do not have enough of samples (only using 467 stocks) to train our neural network. For example, in the original paper by [4], they used the MNIST dataset with 10000 samples.

4 Conclusion

We have obtained promising results as both of our approaches (Factor Model with PCA and CAE) outperforms the baseline method of randomly choosing stocks from the same industry. Interestingly, the hyperparameter tuning procedures lead to choosing $L = 5$ factors/features for both methods, and around $K = 20$ clusters. Qualitatively speaking, the former number of $L = 5$ factors or features is interesting in the sense that it lies up with other researches in financial economics literature, notably the Fama-French 5 model. The CAE approach is more complicated and produces better results than the factor model both in and out of sample, but the margin is very small.

5 Extension

5.1 In progress

Notice that the mean SR we get during test period is much lower in comparison with the result in the train period. We suspected that this is because of non-stationarity in stock data, and we implemented a rolling window approach, where we have a 60 day training to select stocks followed by a 60 day trade over 2010-2011. The mean SR for this is much higher, ≈ 1.2 for factor model approach. Given time, we will implement this for the CAE approach and also to do hypothesis testing.

5.2 Future Work

Given the discussion in Section 3.5, it would be worthwhile to look at a larger stock universe and see how CAE performs. Further, we would also like to spend more time investigating other hyperparameter for this approach such as epochs and batch sizes. It would also be interesting to see how this CAE approach on 2D GAF data generated by time series actually compares to just applying some 1D convolutional/recurrent neural network to the original time series.

Contribution

Project conceiving: N.T.

Data processing: B.C, C.Z, N.T

K-means in factor model: B.C, C.Z

Factor choice procedure: N.T

ARIMA model: N.T

Trading simulation: N.T

Milestone write-up: B.C, N.T, C.Z

CAE implementation: N.T

Hyperparameter tuning: N.T

Rolling window train-test: B.C, N.T, C.Z

Poster: B.C, N.T, C.Z

Final write-up: N.T

References

[1] Marco Avellaneda and Jeong-Hyun Lee. Statistical arbitrage in the US equities market. *Quant. Finance*, 10(7):761–782, 2010.

[2] Jushan Bai and Serena Ng. Determining the number of factors in approximate factor models. 2003.

[3] Yuxing Chen, Weiluo Ren, and Xiaoxiong Lu. Machine learning in pairs trading strategies. 2018.

[4] Xifeng Guo, Xinwang Liu, En Zhu, and Jianping Yin. Deep clustering with convolutional autoencoders. In *ICONIP*, 2017.

[5] Hassan Ismail Fawaz, Germain Forestier, Jonathan Weber, Lhassane Idoumghar, and Pierre-Alain Muller. Deep learning for time series classification: a review. *Data Mining and Knowledge Discovery*, Mar 2019.

[6] Taewook Kim and Ha Young Kim. Forecasting stock prices with a feature fusion lstm-cnn model using different representations of the same data. In *PloS one*, 2019.

[7] Anran Lu, Atharva Parulekar, and Huanzhong Xu. Cluster-based statistical arbitrage strategy. 2018.

[8] Alexei Onatski. Determining the number of factors from empirical distribution of eigenvalues. *The Review of Economics and Statistics*, 92(4):1004–1016, 2010.

[9] Johannes Stübinger, Benedikt Mangold, and Christopher Krauss. Statistical arbitrage with vine copulas. *Quant. Finance*, 18(11):1831–1849, 2018.

[10] Zhiguang Wang and Tim Oates. Encoding time series as images for visual inspection and classification using tiled convolutional neural networks. In *Workshops at the Twenty-Ninth AAAI Conference on Artificial Intelligence*, 01 2015.

[11] Jiayu Wu. A pairs trading strategy for goog/googl using machine learning. 2015.

Github Link: https://github.com/brianjychan/statarb.git