Deep Portfolio Theory – analyze the ability of a Neural Network based approach to uncover market factors

Haoran Su, Ruijing Yang, Bokai Xiang, Wanting Chen
Instructor: Prof Igor Halperin

Abstract

In dealing with high-dimensional data sets, factor models are often useful for dimension reduction. The estimation of factor models has been actively studied in various fields. After seeing that the residuals taken from real data using factor models always have some patterns, we change to the Neural Network, a nonlinear model and find the results improve. We first generate synthetic data with autoregressive residuals and do linear model, Principal Component Analysis and Neural Network methods, and we find that Neural Network really captures more information, compared with others. Increasing the number of factors also show Neural Network's improvement. Further we want to uncover significant factors from market. We apply auto-encoder together with SVM (support vector machine) to predict the trend of S&P 500 index from 1/3/2007 to 11/2/2017 with 23 technical indicators. We find that the extracted factors from auto-encoder performs better than the raw factors.

Keywords: principal component analysis; residuals; Auto-Encoder; market factors

I Introduction:

The increasing accessibility of `big data' occurs also in economics and finance. To deal with such high-dimensional data sets, factor models are often used, since they can reduce the dimension and effectively extract relevant information and can also reduce related dimensionality problem. The estimation of high-dimensional factors has been actively studied extensively in statistics and econometrics.

Other linear models such as Principal Component Analysis have been used popularly. Principal Component Analysis is a statistical procedure that uses an orthogonal transformation to convert a set of observations of possibly correlated variables into a set of linearly uncorrelated variables. These variables are principal components, and we will find variables that can explain most of original property. We assume the residuals are all i.i.d. normal distributed when dealing with linear PCA. However, financial data does not follow i.i.d. normal distribution. In factor model, we assume the data follows:

$$\mathrm{R} = \mathrm{LF} + \mathrm{U}$$

Where R is synthetic return, L is factor loading, F is factor, U is synthetic residual.

Motivated by this, we seek for Neural Network method to deal with the problem with the hope that we can gather more information. To demonstrate the effectiveness of Neural Network methods, we first generate synthetic stock return data with one market factor and autoregressive residuals. Here we assume residuals follows AR(1) process with time-invariant decay rate b, that is $\mathrm{U}_{\mathrm{t}+1} = bU_t + \zeta$. b also denotes the mean-reversion and volatility of residuals. After generating the data, we compare the ability of uncovering factors and model prediction ability among Linear Regression, PCA and Neural Network methods. The effectiveness of different methods is determined by the estimated b. We find that Neural Network Method would best mitigate the autoregressive coefficient b. In other words, it extracts more useful information from data than linear model and PCA.

Furthermore, we increase the number of factors, then we use Auto-Encoder for feature dimension reduction. Compared with original Neural Network method, the prediction ability of Auto-Encoder is better than pure Neural Network method.

All these works show that Neural Network Method is effective dealing with synthetic data and also confirm that this method is powerful for capturing patterns in data. Then we switch to real data and try to uncover market factors using auto-encoder technique. We get the S&P 500 index from 3/1/2007 to 11/2/2017. The factors we choose here are 23 technical factors for all S&P 500 index. We turn these factors into indicators, which are all -1/0/+1, and we want to check their performance to predict the trend of the S&P 500 index rather than the real-valued log return. We use SVM to predict the S&P 500 Index trend. We find that although encoded factors do not give us a very good prediction accuracy, they can give pretty good trading signals. We would long the index If the prediction is up-trend (+1) and go short otherwise.

II Auto-regressive residual analysis

1.Generating synthetic data

First, we want to compare the PCA with neural network method in terms of the ability to uncover market factors with ‘cleaner’ data, because there may exist too much noise in real data, and it might be hard to judge whether a model is good enough for the fitting. Hence, we initially use synthetic data to test the quality of fitting concerning different models.

In the traditional linear factor model, the residual is assumed to be independently identically distributed. However, this is not always the case in the real world especially when we try to predictor financial data (i.e. returns of stocks) in linear fashion models. Those residuals always demonstrate some features which cannot be explained by the simple linear function. J. Yeo and G. Papanicolaou’s method [1] of generating synthetic data actually capture this characteristic, here is what the model looks like:

$$\mathrm{R} = \mathrm{LF} + \mathrm{U}$$

Here R is synthetic return, a N×T matrix, L is factor loading, a N×p matrix, F is factor, a p×T matrix, U is synthetic residual, a N×T matrix. Here N is number of stocks (or other financial products), T is time. $\mathrm{L_{ij}}, \mathrm{F_{ij}}$ follow

Instead of typical assumption that the residuals are i.i.d. normal distributed, this model set the residuals follow an autoregressive process as following:

$$\mathrm{U_{NT}} = \mathrm{I}_{\mathrm{NN}}^{\frac{1}{2}} \epsilon_{\mathrm{NT}} B_{TT}^{\frac{1}{2}}$$

$$\epsilon_{\mathrm{ij}} \sim N(0,1)$$

$$\mathrm{B_{ij}} = b^{|i-j|}$$

Here b denotes the mean-reversion and volatility of residuals, and we simplify treat it as a time independent constant. We estimate b using an AR(1) model, $\mathrm{U_{t+1}} = bU_t + \zeta$, where ζ is independent from U. Thus $\hat{b} = E[U_t U_{t+1}]/E[U_t^2]$

2.Fitting synthetic data with 3 models

After generating synthetic data with one factor (p=1), we used linear model, Principal Components Analysis and Neural Network model to fit the data. We compare the estimated decay rates b with the initial value we set for these three methods. Figure1 demonstrates the result:

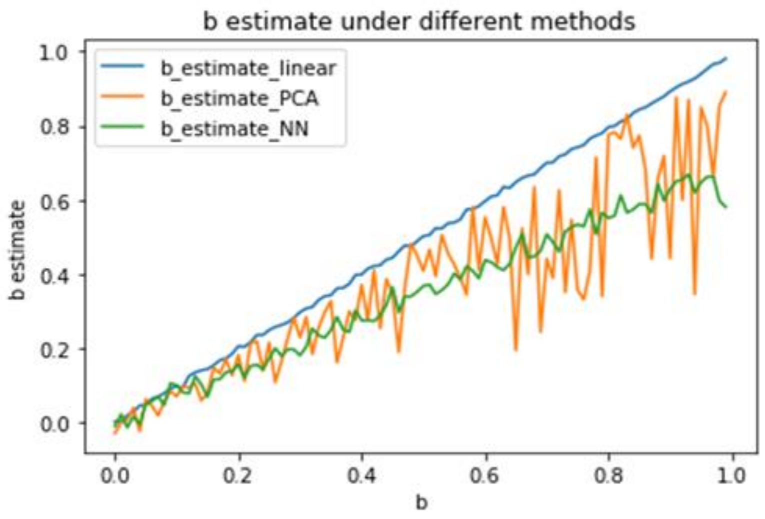


Figure 1: estimated b for different methods, original b is from 0 to 1

The horizontal axis denotes the initial value of b that we set to generate the data, and the vertical axis represents the decay rates of the residuals that the fitting models actually captured.

We can see the residuals in the linear model demonstrate almost the same autoregressive feature with the initial 'residuals' we set. It is as expected as the data was generated through a linear fashion.

For the PCA model, because we set p=1 here, we only use the first eigenvalue, which nearly accounts for 50% of total variance of returns, to do the regression. The result shows that in general, the residuals captured by the PCA have lower auto-regressiveness than the initial model setting. However, it's somehow volatile in capturing this feature, making it unreliable to some extent. Furthermore, we have to extract about first 50 eigenvalues to catch about 90% of total variance of returns. This indicates that PCA model cannot distinguish factors very well.

The green line reflects the results of neural network model. Figure 2 shows roughly how our neural network model works [2].

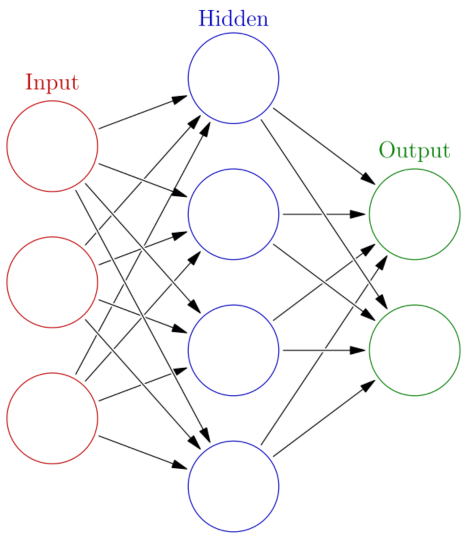


Figure 2: an outline for our neural network model, there is 1 node, 50 nodes and 1 node in the input, hidden, and output layer, we use Relu as our activation function

We set one hidden layer and 50 neurons in the hidden layer, although we actually have no intuition about choosing the hidden layer's shape. We use the mean square error as loss function:

$$\text{loss} = \frac{1}{\text{n}}\sum(y_i - f(\omega x)_i)^2$$

here ω is the weight to optimize, and the function f is the activation function, we use gradient descent method to optimize:

$$\Delta\omega = \nabla\text{loss}$$

$$\omega_{\text{i}+1} = \omega_i - \alpha\Delta\omega$$

Here α is learning rate, and we set it 0.01.

Figure 3 shows loss of our neural network, we train it for 1000 times, and the loss converges after about 100 steps and become stable.

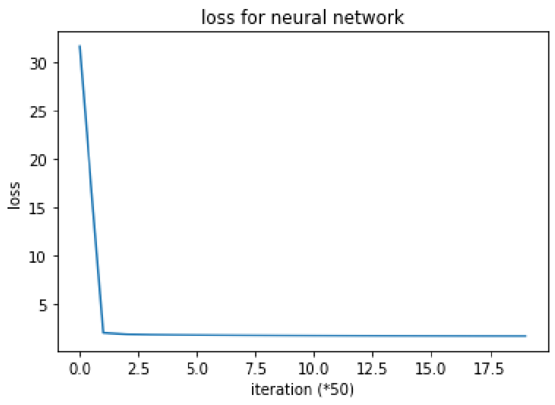


Figure 3: loss of our neural network, please notice that xlabel should modified by 50 times, and the loss converges quickly

From figure 1, we can see that neural network model demonstrates a stable characteristic comparatively. As the decay rate of the initial setting increases, the estimation value deviates downward, indicating the model's ability to capture the residuals' feature which the simple linear model fail to capture.

This result indicates that the neural network models has better quality in predicting the returns to some extent. Next step we are going to test deep learning method accompany with dimension reduction techniques which are derived from neural network.

II Uncover factors with Auto-Encoder

We now choose 50 factors (p=50), to see if we can extract features and use these extracted features to make prediction using auto-encoder.

An auto-encoder is an artificial neural network used for unsupervised learning of efficient codings. The aim of an auto-encoder is to learn a representation (encoding) for a set of data, typically for the purpose of dimension reduction.

There is two main part in auto-encoder: one encoder part and one decoder part. In the encoder part, we extract features and make dimension reduction; in the decoder part, we reconstruct original input with extracted features. Figure 4 shows roughly how auto-encoder works:

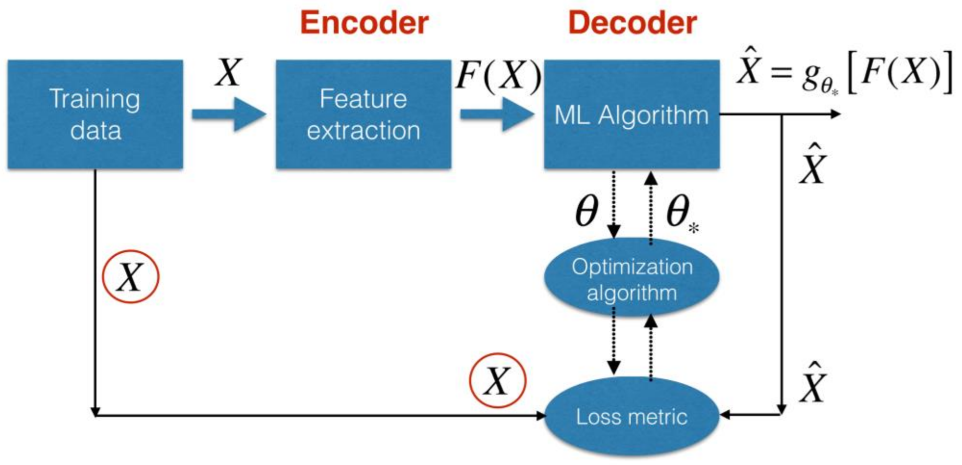


Figure 4:an outline for our auto-encoder method, we first encoder the input X into F(X), then decoder it to get an estimated $\hat{X} = g(f(X))$

This auto-encoder method is built on neural network framework. The input layer has 50 neurons, and hidden layers have 15,10,8,10,15 neurons to encode and decode. To predict, we also implement one neural network inside this auto-encoder after encoding to regress, and this neural network has two layers with 8 neurons. There are two parts of the loss function, one is from the neural network part, which is same with the mean square error above; another is from the auto-encoder part, which is equal to:

$$\mathrm{loss}_{\mathrm{auto-encoder}} = \frac{1}{n}\sum\left(X_i - g\big(f(X)\big)_i\right)^2$$

Besides, we add a L1 regularization here [3]:

$$\mathrm{L}_1 = \lambda\sum|\omega_i|$$

Here λ is our coefficient for L1 penalty, we set it as 0.02, and ω includes the weight for both encoder part and decoder part.

With all these losses, we also use gradient descent method to optimize, and this is same with that in neural network method. Figure 5 shows the loss graph:

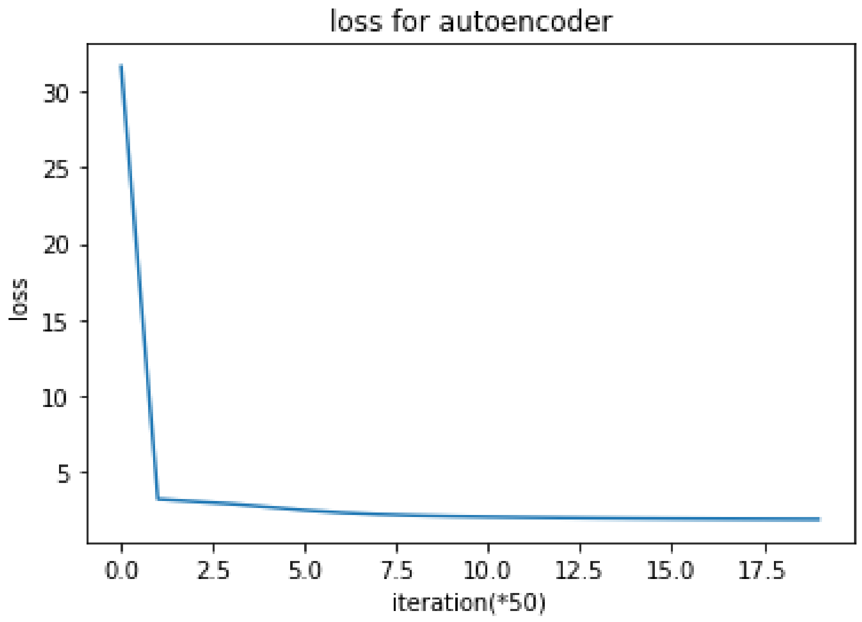
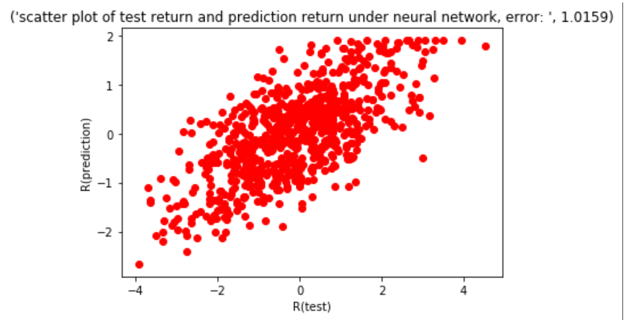


Figure 5: loss for auto-encoder, we optimize for 1000 times, and this loss graph is similar to that for neural network

To test the feasibility of this approach, we still use synthetic data, with increased factors. Now we split our data into a training set and a testing set. Here we just compare these two models and do not do any optimization, so we do not use a validation set here. We compare the predictions through auto-encoder approach and that through neural network method without dimensionality reduction. The results are shown in figure 6.



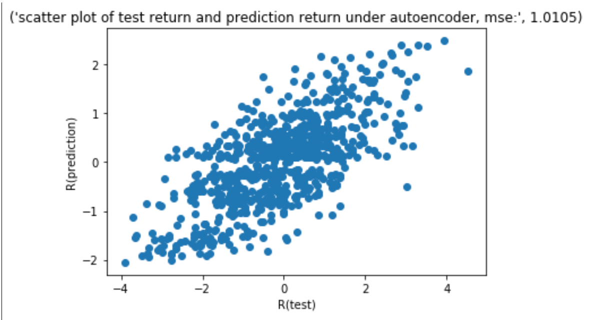


Figure 6: comparison between scatter plot of predicted return and actual return, the prediction from neural network is shown in the left and that from auto-encoder is shown in the left

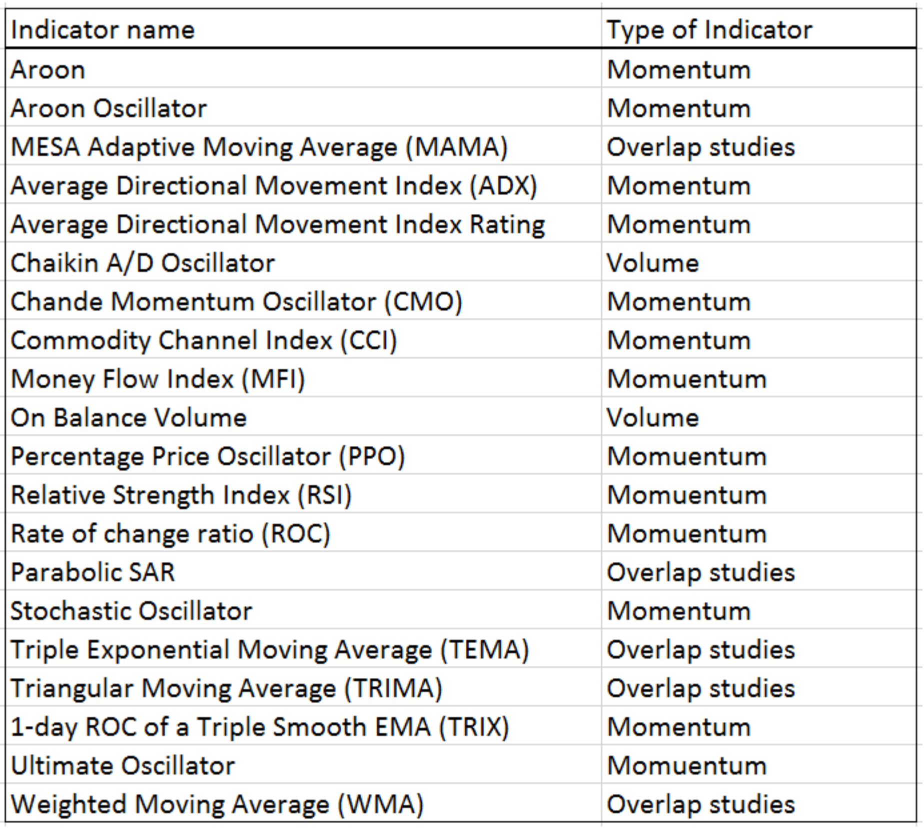
It seems no significant differences in terms of the prediction results even though the mean square error of the auto-encoder is slightly lower. This is mainly due to the pre-assumption of the synthetic data factors that they are i.i.d, indicating no relationship with each other which result in little influence of the dimensionality reduction. Nevertheless, factors in the real world are somehow related which might make the auto-encoder algorithm more effective when deal with those data.

III Deal with real data

Further, we want to uncover factors from market. We apply Support Vector Machine (SVM) and Auto-Encoders with SVM [4] to predict the trend of S&P 500 index. Auto-Encoders as an alternative means to perform a feature reduction. In this paper we investigate the application of both SVM and AE in more general terms, attempting to outline how architectural and input space characteristics can affect the quality of prediction. We were intended to predict the log return of S&P 500 index at first, but the results were not very good. This is understandable as most indicators we use are momentum indicators. These momentum indicators may perform better to predict the trend than to predict the return.

1.Data preparation

Historical data consist of the price of S&P 500 index from 1/3/2007 to 11/2/2017. We separate this data into a training set, a validation set, and a testing set with a ratio of 5:1:2, no overlapping. We use python package TA-Lib to calculate 23 technical indicators. Table 1 lists some indicators used in our experiments:



Indicator name	Type of Indicator
Aroon	Momentum
Aroon Oscillator	Momentum
MESA Adaptive Moving Average (MAMA)	Overlap studies
Average Directional Movement Index (ADX)	Momentum
Average Directional Movement Index Rating	Momentum
Chaikin A/D Oscillator	Volume
Chande Momentum Oscillator (CMO)	Momentum
Commodity Channel Index (CCI)	Momentum
Money Flow Index (MFI)	Momuentum
On Balance Volume	Volume
Percentage Price Oscillator (PPO)	Momuentum
Relative Strength Index (RSI)	Momuentum
Rate of change ratio (ROC)	Momuentum
Parabolic SAR	Overlap studies
Stochastic Oscillator	Momentum
Triple Exponential Moving Average (TEMA)	Overlap studies
Triangular Moving Average (TRIMA)	Overlap studies
1-day ROC of a Triple Smooth EMA (TRIX)	Momentum
Ultimate Oscillator	Momuentum
Weighted Moving Average (WMA)	Overlap studies

Table 1: some indicators we use, most of which are momentum indicators

Then we discrete these indicators into set {-1;0;1} or {-1,1}. Most indicators denote the trend and the original numbers themselves do not tell us much information. For example, The Aroon indicator is used for identifying trends and the likelihood that the trends will reverse. There are two types, a bullish one, which is calculated as [(# of periods) - (# of periods since highest high)] / (# of periods)] x 100, and a bearish one, which is [(# of periods) - (# of periods since lowest low)] / (# of periods)] x 100. We set it 1 if the bullish Aroon indicator is between 70 and 100, -1 if the bearish Aroon indicator is between 70 and 100, 0 otherwise. We do similar labelling for other indicators, and rules are different for every indicator.

2.Support Vector Machine

We first use Support Vector Machine (SVM) to predict the trend, Figure 7 shows briefly how SVM works.

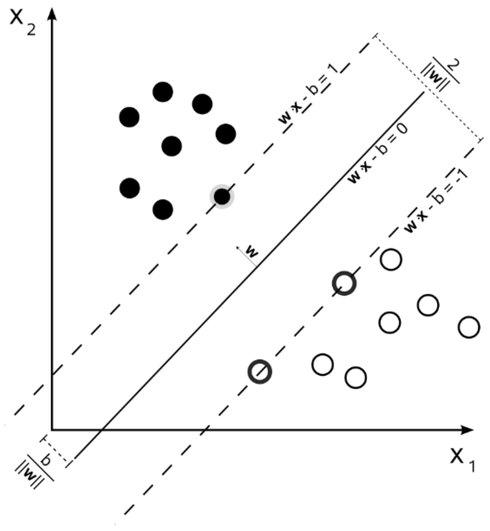


Figure 7: a brief introduction of SVM, the black dots (+1) and the white dots (-1) are separated by the hyperplane

Our data are:

$$(x_1, y_1), \ldots, (x_n, y_n)$$

where the y_i are the class to which the x_i belongs. They are either 1 or -1, indicating a uptrend or a down trend. Each x_i is a 23-dimensional real vector. We want to find the "maximum-margin hyperplane" that divides the group of points x_i for which $y_i = 1$ from the group of points for which $y_i = -1$, which is defined so that the distance between the hyperplane and the nearest point x_i from either group is maximized.

The region bounded by these two hyperplanes is called the "margin", and the maximum-margin hyperplane is the hyperplane that lies halfway between them. These hyperplanes can be described by the equations:

$$\mathrm{f(x)} = \omega x - b = \pm 1$$

The loss function is

$$\frac{1}{n}\sum|1 - y_i f(x_i)| + \lambda\sum|\omega|^2$$

Here we add a L2 regularization with $\lambda = 0.01$.

We do not apply any soft margin or kernel function in our SVM method. We use gradient descent method to optimize, and Figure 8 shows the loss:

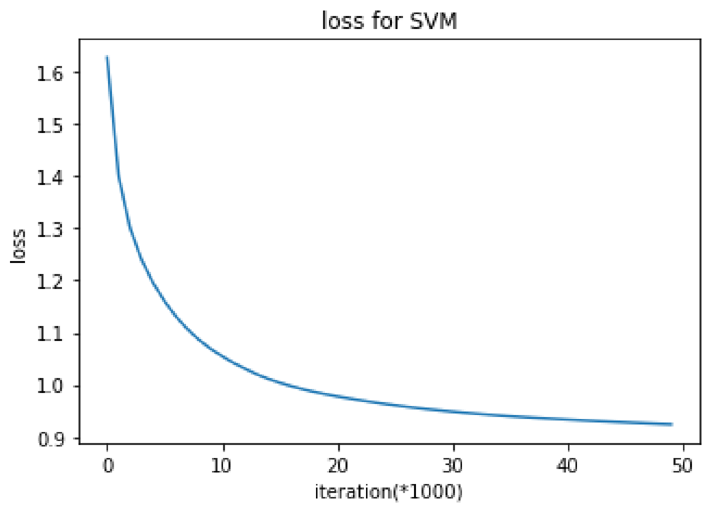


Figure 8: loss of our SVM method, it converges after about 20 thousand iterations, not as quick as neural network method

3.Auto-encoder

Then, we try to combine Auto-encoder (AE) together with SVM to do the prediction and compare the result to see if AE really works. The auto-encoder we choose has 5 layers. The input layer has 23 neurons for 23 indicators. The first, second and third hidden layers respectively have 23, 15, 8 neurons, and then decode them back into 23 outputs symmetrically, then the decoded variable will be compared with the original inputs.

The innermost layer has 8 nodes, which represent the key features of all inputs. We use sigmoid function as our activation function. They are non-linear combination of the original inputs and have no clear meaning. After auto-encoder, we use SVM to do the classification using 8 extracted feature, and compare the result with SVM without auto-encoder. Figure 9 shows how our auto-encoder model works.

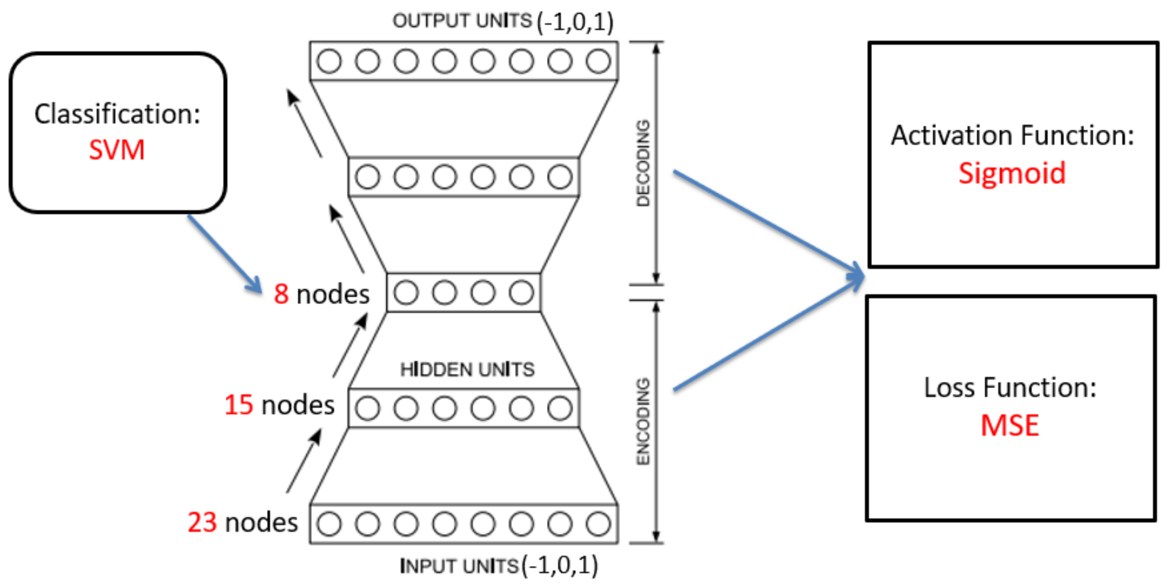


Figure 9: an outline of our auto-encoder method

The loss function also contains three parts, one for auto-encoder part, another for SVM part and one for L1 regularization. We still use mean square loss for auto-encoder part. However, we do not use gradient descent optimization here, because it is likely that this model would reach a local optimum rather than a global optimum. We use Adam optimization, which uses Kingma and Ba's Adam algorithm to control the learning rate [5]. It uses moving averages of the parameters (momentum) and it is more stable than gradient descent optimization method. Figure 10 shows our loss of training set, it converges much faster than pure SVM method. Loss is about twice of that from pure SVM method, because loss from SVM part and loss from auto-encoder part count about 50%-50% of total loss.

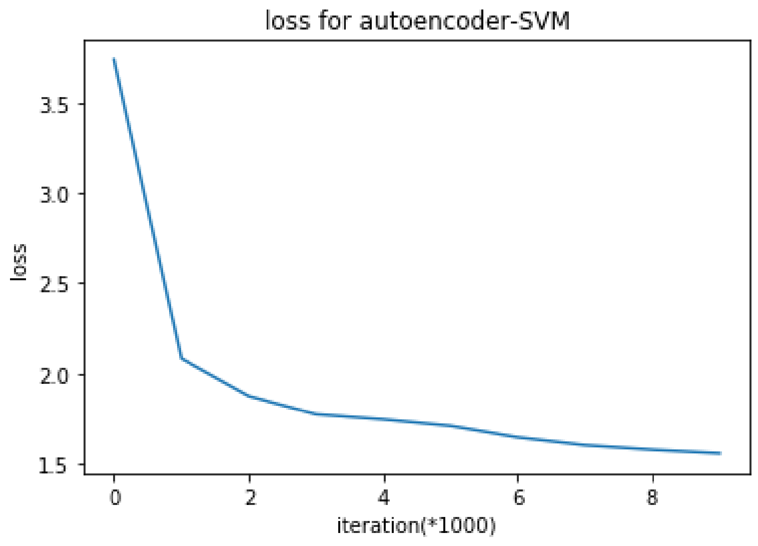


Figure 10: loss of training set for our auto-encoder model with SVM

Actually auto-encoder with SVM cannot give a very high prediction accuracy. SVM method and auto-encoder with SVM both would give a prediction accuracy around 53%, although training accuracy would be higher than 70%.

4.Trading Strategy

Now, we have every-day predicted trend signals and we can apply them to the validation/testing set. If predicted signal is +1, we long S&P 500 at that day and we go short otherwise. If the sign of the log return at that day is the same as the signal we produce, this strategy would make money; this strategy would loss money otherwise. Figure 11 shows our cumulative log return of S&P 500, strategy using SVM predictions and strategy using Auto-encoder + SVM predictions.

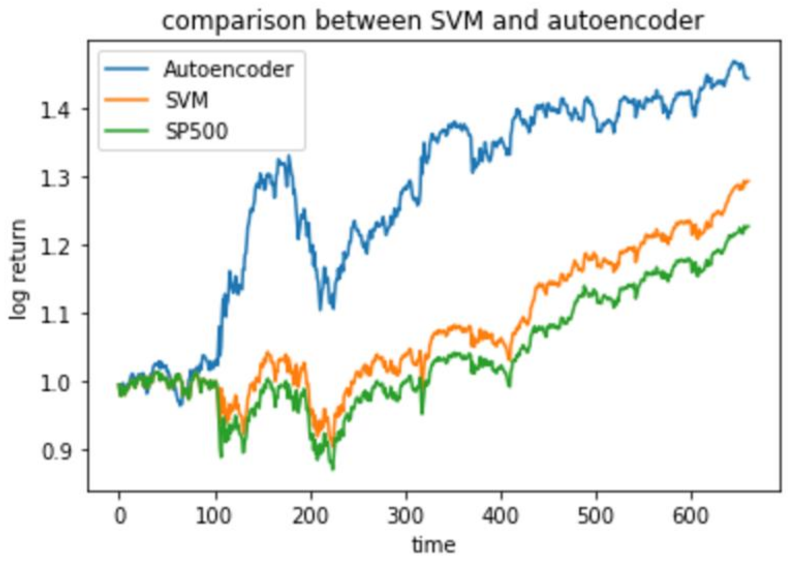


Figure 11: cumulative log-return for different strategies, our testing set contains data for about two years

Figure 11 shows, both pure SVM and auto-encoder combined with SVM would beat S&P 500 index. In other words, the strategy will outperform just holding S&P 500 index. The strategy generated by auto-encoder combined with SVM significantly outperforms pure SVM, indicating that extracted features can help us make predictions of the trend. It can sometimes capture the trend correctly, while it fails to make precise predictions very well. In the second half part, S&P 500 increases moderately, and the strategies both follow this increasing trend.

IV Conclusion

In summary, our neural network model would work to extract factors for non-i.i.d residuals, and is more stable than PCA method to deal with synthetic data. For real data, our auto-encoder model can extract important factors from original factors to make trend predictions. However, there are many points to improve. First, our neural network model is not very stable for real data, we do not have intuition for choosing hyper parameters such as number of hidden layers, number of nodes, activation function and others, so we just try some combinations of parameters, test them for the validation set, then apply them for the testing set. We may need to find the underlying mythology to choose parameters. Second, our classification prediction is based on SVM method, and we only make a two-class classification for long and short. We simply denote an uptrend and a down trend by whether log return is positive or negative. However, this is usually not the case in real life. Most of time people would hold rather than trade every day, so we may add a threshold, above which we may go long or short, or hold if prediction does not break this threshold. Third, we only use technical indicators, while we should also consider more macro factors; after decoding, the underlying meanings of extracted features are pretty unclear. These extracted features may do not make any sense, especially after nonlinear transformation. There are still a lot of further work to do in our work.

V Reference

[1] Random matrix approach to estimation of high-dimensional factor models, Joongyeub Yeo and George Papanicolaou
[2] https://en.wikipedia.org/wiki/Artificial_neural_network
[3] Deep Portfolio Theory, J. B. Heaton, N. G. Polson, J. H. Witte
[4] On Feature Reduction using Deep Learning for Trend Prediction in Finance, Luigi Troiano, Elena Mejuto, Pravesh Kriplani
[5] Practical Recommendations for Gradient-Based Training of Deep Architectures, Yoshua Bengio