Neutral Network in Dimension Reduction with Stock Return Prediction

Team members:

Yang Tao

Meihui Zhang

Yihe Liu

Chengyu Xin

Weijing Li

Sihang Liu

Instructor: Max Reppen

1. Introduction

This project will use machine learning tools and dimension reduction on real data to predict the stock returns. Machine learning contains several potential predictor variables, and its flexibility allows us to push certain frontiers of risk premium. Nowadays, machine learning has grown dramatically in the financial industry, and the data sets have become a higher dimension with more data collection techniques. Thus, we are interested in How the dimension reduction technique can help prediction, specifically, Principal Component Analysis and Autoencoder. PCA is commonly used in reducing dimension. The reason is that the complex relationships in the data set can be detected by deep learning at a certain level instead of the traditional method. The main idea of PCA is to construct new variables that do not exhibit multicollinearity. Autoencoder is another popular method to make a dimensional reduction in stock prediction. The main difference between them is PCA is only for linear regression, but Autoencoder is for both linear and non-linear data. We train and compare the mean square error of them in python to discover their different performance.

Firstly, we process and clean the dataset, such as nan, non-numerical data, fat tail effect, standardization, or normalization. We use different methods to preprocess the data to see the difference and how the dataset and models react. After eliminating the non-effective factors or small factors, we use Autoencoder to implement the dimensional reduction first, indicating that we only focus on non-linear relationships for this part. Then, we construct a PCA method to process the linear factor. Specifically, we want to get a covariance matrix and calculate the eigenvectors and eigenvalues. We can choose the crucial features according to the first largest eigenvalues; then, we use both PCR and

standard linear regression. After that, we cross-compare the performance under the PCA methods and Autoencoder. Later on, we implement the backtest to verify our prediction.

2. Data

Data sets:

We analyze the dataset, which contains three categories: market data, investor sentiment data, and company accounting data. Stock market data includes company identifier, monthly closing price, monthly volume, etc. (totaling 12 updated monthly characteristics). Sentiment data reflects the level of individual investors' confidence in the stock market in the short term, which involves bullish, neutral, bearish, etc. (8 characteristics of which are updated monthly). Accounting data comes from companies' income statements, and balance sheets include currency, assets, comprehensive income, etc. (68 characteristics of which are updated quarterly or yearly depending on accounting reports). Our sample begins in August 2016 and ends in December 2019, approximately four years, and the collection has 2847 firms. Observations are occasionally missing some characteristics. To improve the dataset's quality, we exclude those relevant columns that have incorrectly formatted or incomplete data. We then form about 71 influential characteristics collections.

Preprocess:

There are nan and non-numerical data in our data sets. We drop all non-numerical factors since it is difficult to transform them into numerical aspects. Then, we use four different methods to deal with the nan, which are ['drop nan,' 'fill nan with zero after normalization', 'fill nan with 0', 'fill nan with mean']. We also cut the tail 0.05 or 0.01 data to mitigate the fat tail effect. Besides, distributions of some characteristics in our dataset

are highly skewed and leptokurtic. Thus, we normalize total features into the interval [-1,1] to avoid outlying observations' undue influence. We also did standardization for comparison.

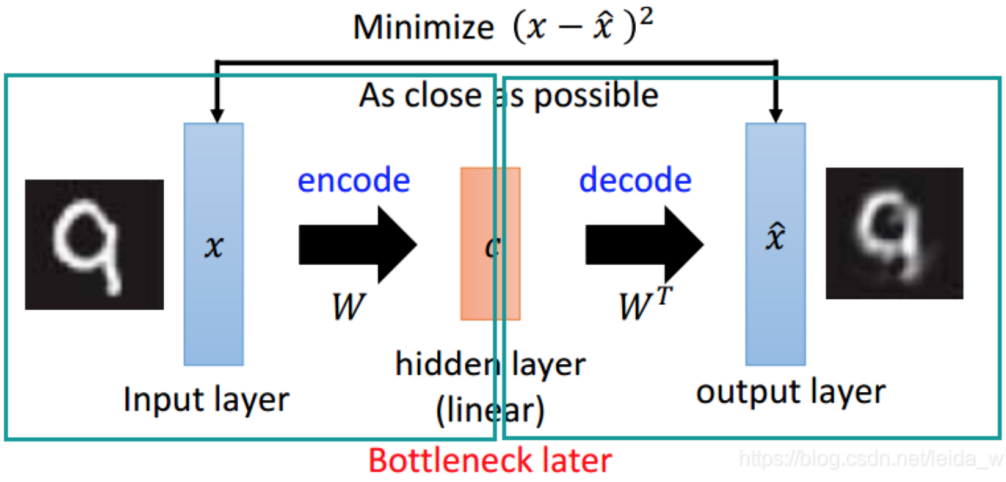
Moreover, before doing any technique analysis, we use an effective-test (T-test with 0.95 confidence interval) to eliminate ineffective factors or small factors. Finally, we separate our sample into two disjoint periods that still maintain the ordering of the data. The first subsample is "Calibrate," which is used during the learning process to fit the parameters in our model. These data control the complexity of the model and affect the performance of our machine learning results. The second subsample is "Validation," which is used for tuning the hyperparameters. The validation is to simulate an out-of-sample test for the model to search for a level of model complexity that could provide strong out-of-sample performance. We also sample batches from our datasets, both shuffle, and without-shuffle. We use shuffle to eliminate the time-series correlation.

3. Introduction of Tools

a. AutoEncoder

Autoencoder is an unsupervised learning method in which we leverage neural networks for the learning tasks. It is a widespread technique for dimensional reduction. There are two processes: First, we use the encoder to compress the input data into latent features; Second, we reconstruct the data from the latent features. In order to obtain a representative lower-dimension code, we have to minimize the error of reconstruction. From a single-layer autoencoder to a deep autoencoder, it achieves effective data dimensionality reduction. For example, if we have a batch of data (X1, X2...Xn), we can

use them to get the distribution of X: P(X). By decoding them, we then generate all of X, including (X1, X2...Xn) and those exceptions.



Autoencoder can find latent features of the data by combining the low-level features to form high-level and abstract attributes. At present, most machine learning algorithms' performance depends heavily on the characteristics of their selected features. Deep neural networks based on autoencoders can be effectively used for feature extraction. Autoencoder is an unsupervised, sparse, noise-reduction learning algorithm.

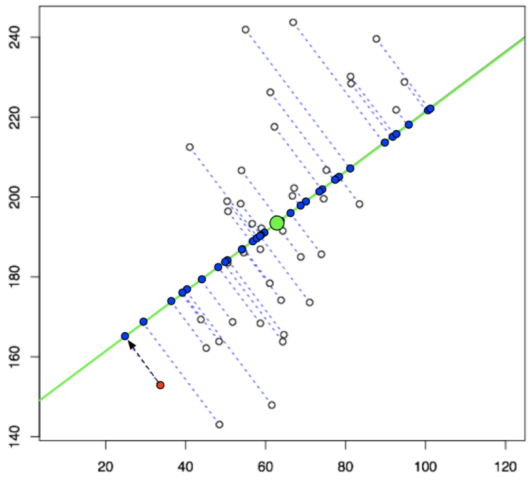
Moreover, the autoencoder's function is similar to PCA, but it is more flexible than PCA. This method provides a wide range of versatility, including linear and non-linear data. In the same level of dimension reduction, the autoencoder method gives more accurate results and predictions. However, it has some drawbacks because of the implementation, computational complexity, and overfitting.

b. PCA(Principal Component Analysis)

People use Principal Component Analysis commonly in data analysis. Through linear transformation, it can transform the data we initially have into a group of linearly

independent representations of every dimension. We could use it to extract the principal feature components from the dataset, and it is commonly used for dimensionality reduction of high-dimensional data.

As we know that the higher the dimensionality of the data, the more complex the machine learning algorithms might be. The level of complexity is even exponentially related to the dimensionality. On the one hand, if the dataset's dimension is too large, we might see machine learning could consume an unacceptable amount of resources. To solve this problem, we have to decrease the dimensionality of the data. On the other hand, if we reduce the dimensionality of the data, we must lose some information compared with the raw data. But depending on how much it is relevant to the original dataset, we can find ways to lose as little information as possible in the unmodified data while reducing the dimensionality. PCA is an excellent method to let us get a simpler regression result. (ProgrammerSought)

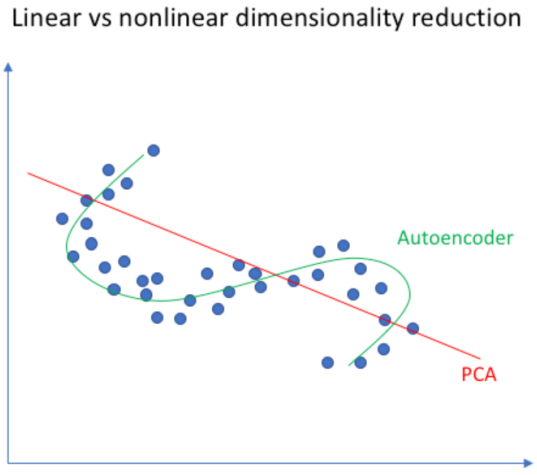


Assume we have a m n-dimensional dataset. The general process of PCA has the following steps. First, it would reform the unmodified dataset into an n rows and m

columns matrix, name X. Then it would zero-average each row of X to make each row standardize. In order to do that it would subtract each number by the mean of each row.

In the next step, it would find the covariance matrix as $C = 1/m * X * X'$. After that, it would compute the eigenvalues and corresponding eigenvectors of the above covariance matrix. Next, it would sort the eigenvectors in rows according to the size of eigenvalue from the largest to the smallest. Then, it uses the first k rows of the newly ranked matrix to form a matrix P which is the data corresponding to the top k eigenvalues. At last, Y=PX would be the data after the process. And we would then run the regression on the newly formatted dataset.(ProgrammerSought)

c. Difference between PCA and AutoEncoder



PCA can be considered as a linear version of the AutoEncoder. We can also say that both the encoding and decoding form and the reconstruction error form are completely the same for PCA and AutoEncoder. The only difference is PCA is linear. Linearity or not brings different properties to the optimization implementation: PCA can

directly obtain the optimal analytical solution, while AutoEncoder can only obtain the local optimal numerical solution through back propagation.

4. Algorithm

4.1 Autoencoder

The first part of our work is dimension reduction, helping us reduce the factors without losing features. There are two ways to process it-Autoencoder and PCA. Unlike PCA, which analyzes linear relationships between elements, Autoencoder focuses on capturing nonlinear relationships through an artificial neural network. By using both, we hope to capture both linear and nonlinear relations and obtain a better performance. According to the plan, we use Autoencoder to process data first, pick up the most influential factors or those that explain the most variance. In the meantime, we also want to pick up some less effective factors, explaining the errors.

For autoencoders, we use Keras and TensorFlow to build Autoencoder Model, using ‘adam’ as an optimizer and ‘mean squared error’ for loss function. There are six layers in our model; we use ‘relu’ as activations for the first five layers and ‘linear’ for the last layer. In order to evaluate our results, we choose the sum of absolute relative errors to represent the gap between our predictions and original X. Then, we obtain a series of indexes ranked from the smallest to largest errors. Then we can have a sense of which factors can explain most variance and most relative with others. The logic behind this is when we use the less dimension feature layer (most shared information) to reproduce X, and if we can reproduce the specific Xi pretty well, then the Xi is close relative to the most shared information or most essential features. In other words, Xi that

shares information with most of the others on the same market, can explain5 most of the variance.

	Communal	Index
0	0.000074	tfvceq
1	0.000162	epsf12
2	0.000311	ggroup
3	0.000948	gind
4	0.001413	epsfxq
…	…	…
53	0.419562	sic
54	0.424965	ltq
55	0.482115	atq
56	0.482119	lseq
57	0.561507	mkvaltq

4.2 Principal Component Analysis, PCA

After the Autoencoder results, we use PCA to capture the essential linear relationship, which is more stable in prediction and leads to more consistent performance. Below are basic steps to implement PCA.

Step 1: Take the matrix of data. Rows represent data items while columns stand for features.

Step 2: Standardize data. Subtract mean of the column from each data, so that our centered data as mean equals to 0 for each feature: .

Step 3: Get covariance matrix. Covariance of centered data X: XTX.

Step 4: Calculate Eigen Vectors and Eigen Values. We need to decompose covariance matrix $\mathbf{X^TX}$ into $\mathbf{PDP^{-1}}$, where **P** is the matrix of eigen vectors and **D** is diagonal matrix with eigen value, $\mathbf{l_1, l_2, \dots, l_m}$, on the diagonal**.**

Step 5: Sort Eigen Values. Sort $\mathbf{l_1, l_2, \dots, l_m}$ from largest to smallest, and sort eigen vectors based on the sorted eigen values. Denote sorted $\mathbf{P^*}$ and $\mathbf{D^*}$**.**

Step 6: Choose important features. Choose the eigen vectors according to the first n largest eigenvalues.

Pseudocode for Golub-Kahan method (EVD) algorithm

Algorithm Golub-Kahan method (EVD)
Parameter: Number of epoch S, Number of dimension K
Input: Data matrix $X \in R^{M*K}$
Output: Eigenvectors $W \in R^{M*K}$ $C = XX^T/M$ $U_1 T U_1^T$ = Householder (C) $U_2 \Lambda U_2^T$ = Diagonalization (T) $U = U_1 U_2$ $W = X^T U$

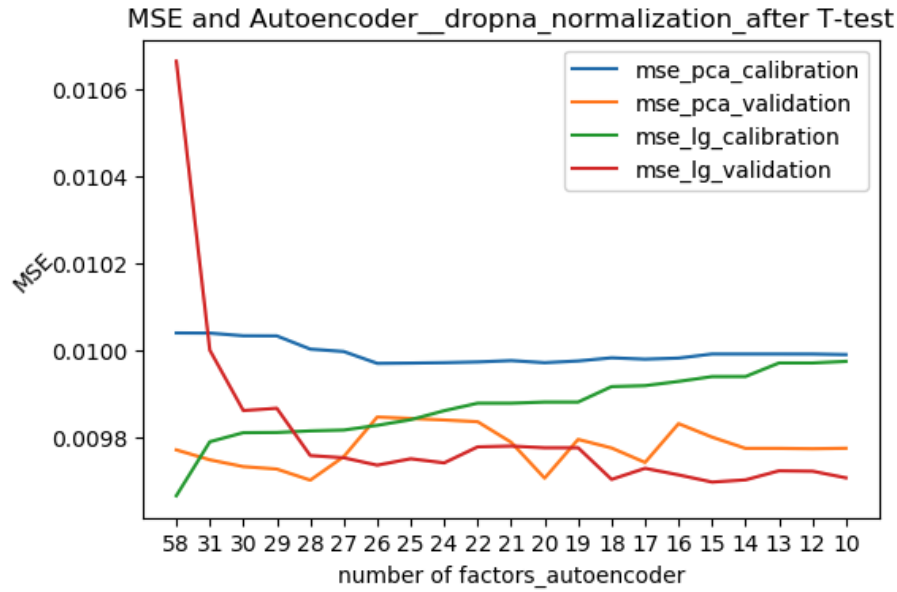
4.3 PCR

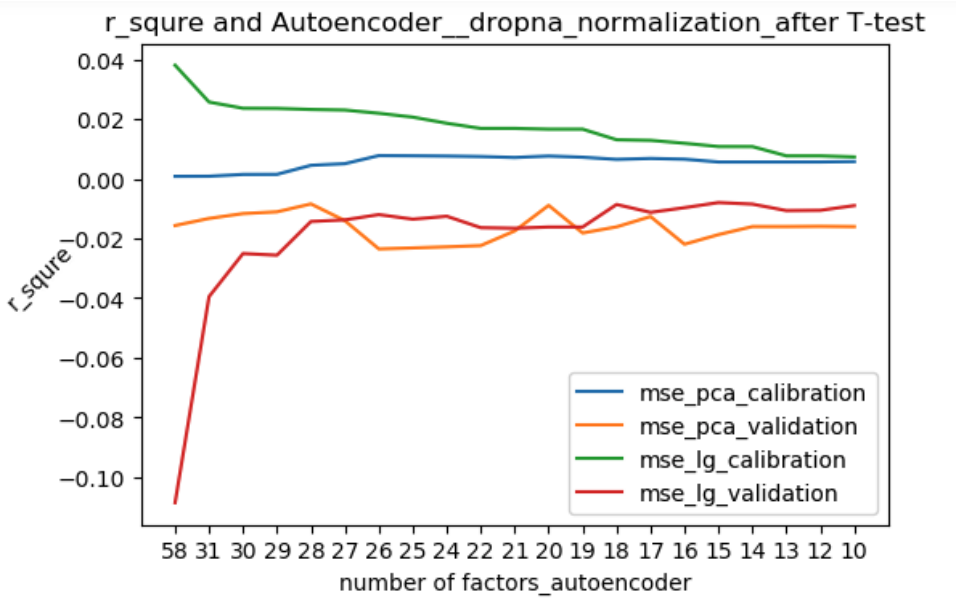
After projecting the feature space via PCA onto a smaller subspace, where the eigenvectors will form the axes,we can implement a linear regression based on new data sets whose dimensionality of feature space has been reduced.

$$X_i = e_{1,i} P_1 + \cdots + e_{n,i} P_n$$

5. Backtest

We test Autoencoder effects on both linear regression and PCR. We also compare the performance of linear regression and PCA. It is interesting to see that, standard linear regression has a large out-of-sample error without autoencoder performance than PCA since one of the PCA objects is to mitigate the performance variance. However, with the Autoencoder's present, the linear regression performance is improved and toward the PCA. On the other hand, PCA did not get much help from Autoencoder.



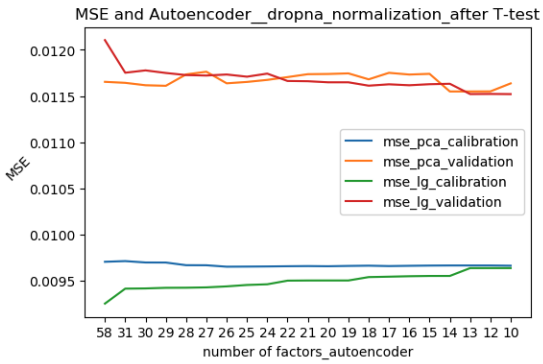


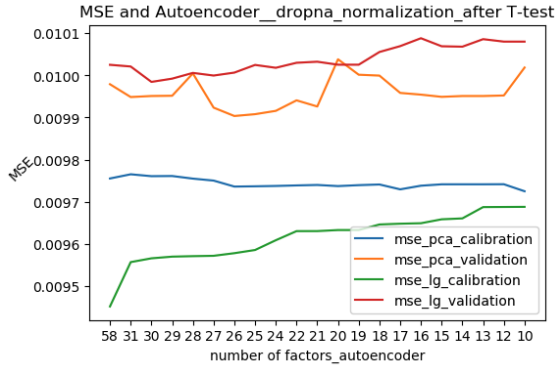
The x-axis is the number of factors from the autoencoder result. a large number indicates that we include more factors from the result and less effect of the autoencoder. The most left means not using the Autoencoder at all.

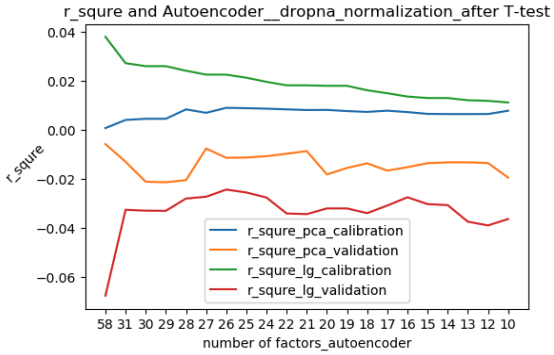
The r-squared told the same story. Another interesting finding is that the model has a more consistent performance at batches without shuffle than the shuffled sets. This

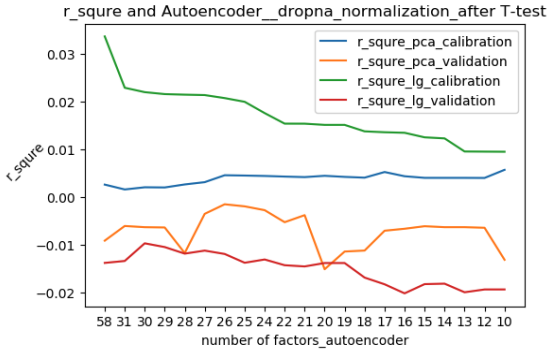
finding leads us to the hypothesis that there is an autocorrelation along the time in the factors. We can prove this hypothesis in future research.

batch with shuffle

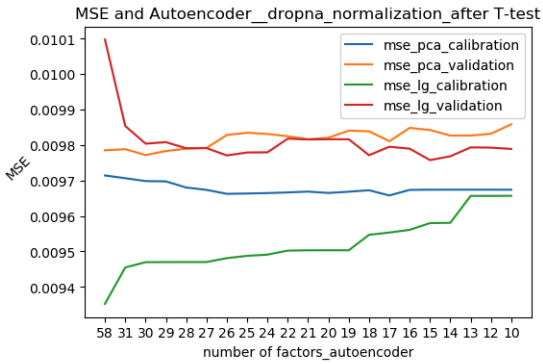


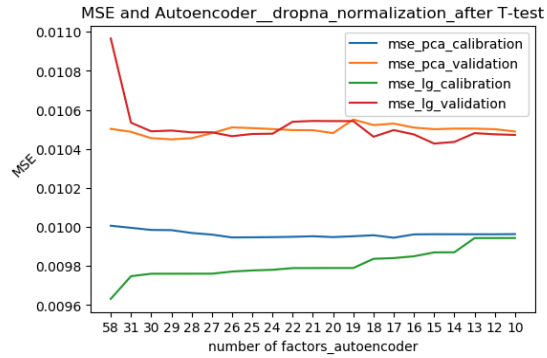




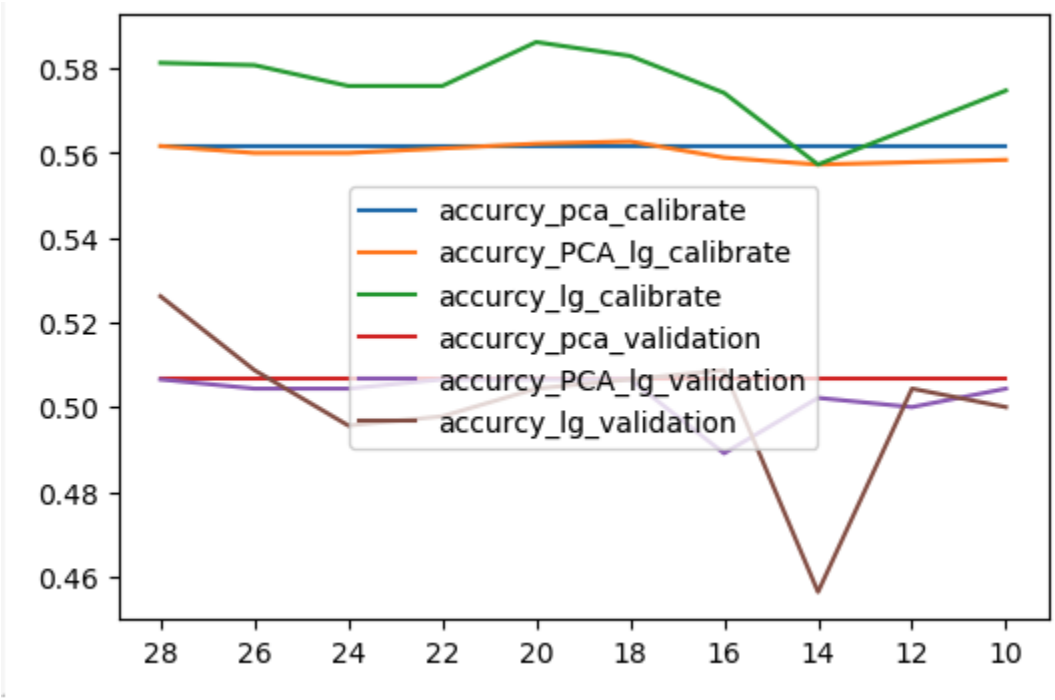


batch without shuffle





we also notice that in the direction prediction, the linear regression has best performance in the sample but large variance out of sample.



6. Conclusion

Originally, Dimension Reduction is designed to reduce computation time and feature space. The sense behind this is because the sample density decreases with the increase of the dimensionality. When we keep adding features, the dimensionality of the feature space grows dramatically. It becomes much easier to find an “overfitting” solution for the machine learning model with a high sparsity, which is the main problem for most machine-learning models. Moreover, according to the manifold hypothesis, the residual is assumed to be noise. Thus, if people manage dimensionality reduction correctly, they should improve the model's performance

by modeling the signal rather than the noise. It's not just a problem with space and complexity. This hypothesis coincides with our experiment results. Both PCA and Autoencoder import the performance of linear regression, especially the out of sample.

Besides, PCA has some drawbacks, including PCA tends to find linear relationships between features, which is not the whole story, and PCA relies on mean and covariance, which sometimes are not enough to define datasets. Nevertheless, we do not see much evidence from our experiments. You can see when fewer factors from the autoencoder result(autoencoder is acting), the performance of PCA and standard linear regression are close to each other. We have a few hypotheses about this result. First, we use linear regression after dimension reduction instead of using the neural network itself. Second, the data sets have a strong linear relationship, and linear correlation explained most variance. After all, further research needs to be conducted.

At this point, we can state that we successfully implement the dimension reduction to improve the prediction's performance. However, we do not prove that Autoencoder has a better performance than PCA due to various factors. In the end, we believe the theory behind the dimension reduction is valid, providing results from our experiment. In addition, we think that the dimension reduction technique would be essential for the prediction concerning higher dimension data sets with more available date sources in modern society.

7. Reference

Chabanenko, V., and L. Nazarov. "Calibration of Models for Option Pricing Using Neural Networks." Journal of Mathematical Sciences (2020): 1-7.

Gu, Shihao, Bryan Kelly, and Dacheng Xiu. "Autoencoder asset pricing models." Journal of Econometrics (2020).

Soleymani, Farzan, and Eric Paquet. "Financial Portfolio Optimization with Online Deep Reinforcement Learning and Restricted Stacked Autoencoder-DeepBreath." Expert Systems with Applications (2020): 113456.

"Principal Component Analysis PCA Principle and Code Implementation." *ProgrammerSought*, www.programmersought.com/article/17513838922/.