Feature Extraction Using Restricted Boltzmann Machine for Stock Price Prediction

Group 9: Jing Zhou, Lulu Deng, Linna Hu

● The project objectives

Restricted Boltzmann Machine (RBM) as an unsupervised Artificial Neural Network (ANN) model for machine learning algorithm that are applied to extract highly discriminative low-dimensional features and improve forecasting accuracy of the regression models.

The main contribution of this paper is that it presents a combined approach to predict stock price, which utilize the RBM and its deep form DBN as the feature extractors and the SVM as the regression model.

1. Background

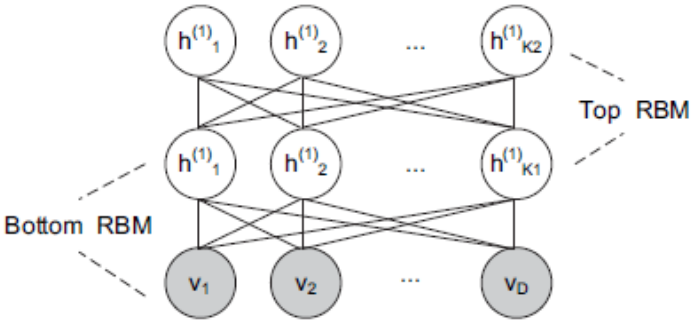
Using RBM as a Feature Extractor

E $[h_j \,|\mathrm{v}, \mathrm{W}] = \mathrm{p}\,(h_j = 1|\mathrm{v}, \mathrm{W})$ can be viewed as the extracted features of v (since hidden units model correlation of dataset). Extracted features can be input for training another RBM to capture higher order abstract features; repeatable as many times as needed to build deeper network. DBN Training:

1. Train the bottom RBM with input v.

2. hidden units of bottom RBM become deterministic, which output E $[h_j \,|\mathrm{v}, \mathrm{W}]$ with input v and serve as the visible units for the top RBM.

3. Repeat 1 and 2 for as many layers as desired (more = higher accuracy). Backpropogation (weights, biases) then used to improve extracting performance.



- **Techniques and Tools**

2. Experiment

2.1 Steps of the Experiment

Utilize the RBM and its deep form Deep Belief Network as the feature extractor, and the Support Vector Machine as the regression model.

We use the SVM for regression with the input data consisting of *f1~f20* of S&P 500. (SVM+1)

We use the SVM for regression with the input data consisting of f1~f4 of S&P 500 index and f5~f20 of the 20 stocks. (SVM+20)

We train the DBN using *f5~f20* of the 20 stocks, and then use the extracted features from DBN and *f1~f4* of S&P 500 as the input of SVM. (SVM+DBN)

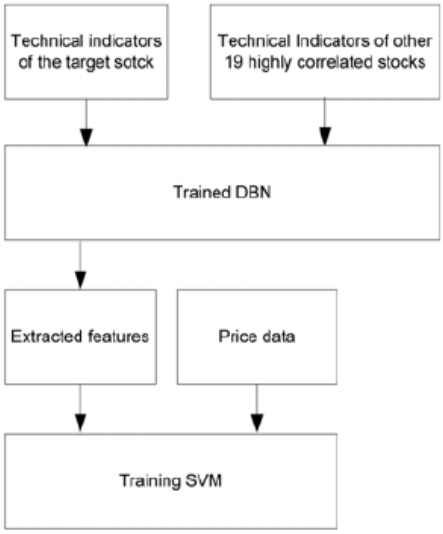


Fig. 3. The data flow of predicting stock price.

2.2 Initial Data Set

The set of data we used was historical price information downloaded from Yahoo Finance as Excel csv files. Besides the S&P information fundamental to our prediction of its own price, the nineteen other stocks we chose were mostly from the information technology sector. These were: ACN, ADP, AKAM, AMAT, CRM, CSCO, CTSH, CTXS, EBAY, FIS, GOOGL, HP, INTU, MA, NFLX, NTAP, T, VZ, WU for the time period ranging from 11/16/2006 to 11/14/2017.

This immediately covered features f1-f4.

2.3 Input Features: 20 Technical Indicators

After gathering the initial dataset, excel functions were utilized to produce formulas that calculated the indicators for every stock. Below is a brief summary.

Feature	Description
f_1	Close
f_2	Open
f_3	High
f_4	Low
f_5	Average Directional Movement Index(ADX)
f_6	Average True Range(ATR)
f_7	Chaikin's Oscillator
f_8	Commodity Channel Index(CCI)
f_9	The Exponential Moving Average (EMA)
f_{10}	Momentum
f_{11}	Moving Average Convergence/Divergence (MACD)
f_{12}	On Balance Volume (OBV)
f_{13}	Parabolic SAR
f_{14}	Price rate-of-change (ROC)
f_{15}	Relative Strength Index (RSI)
f_{16}	Simple Moving Average (SMA)
f_{17}	Stochastic %K
f_{18}	Stochastic %D
f_{19}	Stochastic Slow %D
f_{20}	William's %R

2.4 Using SVM for Regression

Support Vector Machines (SVMs) are supervised learning models with associated learning algorithms that analyze data used for classification and regression analysis:

SVC: find a hyperplane from which the distances of nearest sample points tend to be largest

SVR: find a hyperplane from which the distances of furthest sample points tend to be smallest

2.5 Framework of our Python Replication

We decided to use the Tensorflow library in Python, an extremely popular one used in Machine Learning.

There are two parts to our code.

A) The basic SVM prediction is in the **prediction.py** file.

We import train_test_split from sklearn.cross_validation to split the dataset into training set and test set, and then, from sklearn.svm we import SVR and do the the regression.

B) The Gaussian-Bernoulli Restricted Boltzmann Machine is the in **gbrbm.py** file.

We import the Tensorflow library and do the training. First we define the rbm function in **rbm.py** containing the fitting, reconstruction and initial conditions, based on which, we define the gbrbm function which contains the training process.

2.6 Performance Measurement

Normalized Mean Square Error - measure of the deviation between the actual and predicted values. The smaller the value of the NMSE, the closer are the predicted values to the actual values.

$$NMSE = \frac{\sum_{t=1}^{n} (y_i - \hat{y}_i)^2}{\sum_{t=1}^{n} (y_i - \bar{y}_i)^2}$$

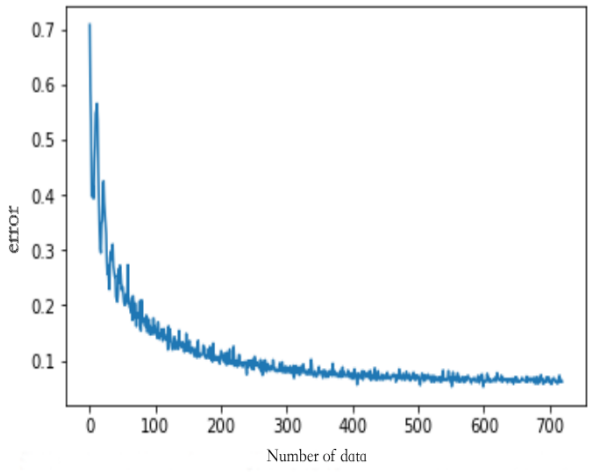
Directional Accuracy - provides an indication of the correctness of predicted direction of stock price.

$$DA = \frac{1}{T} \sum_{t=1}^{T} I\left[(y_t - y_{t-1})(\hat{y}_t - y_{t-1})\right]$$

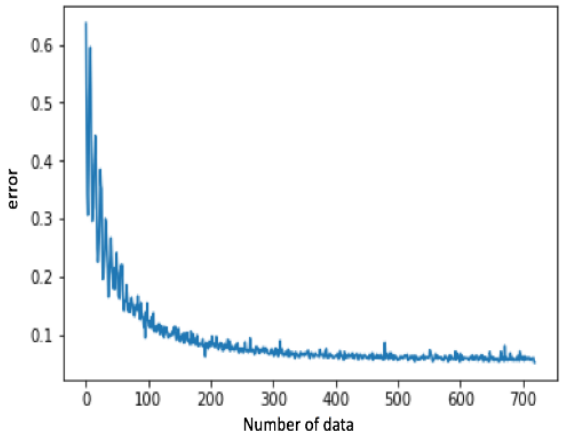
- **Results and Conclusions**

3.1 Training Output

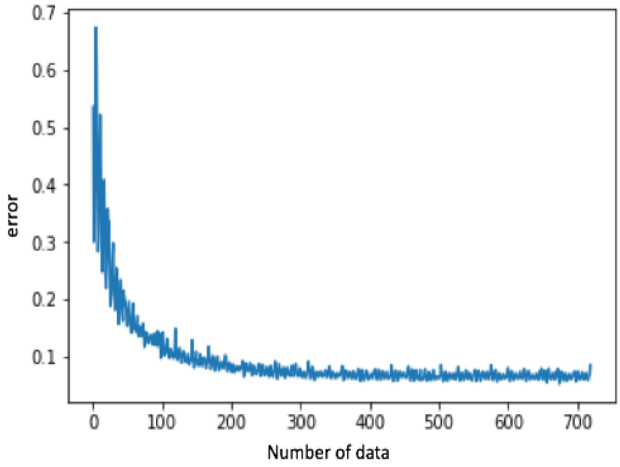
Set 320 visible nodes and 2000, 4000 and 8000 hidden nodes respectively to the machine and calculate the err of each step within 30 epochs and the batch size of 90 with the following formula: $\text{err} = (x_{old} - x_{new})^2 / N$




```
Epoch: 0                Epoch: 10               Epoch: 20
Train error: 0.4702     Train error: 0.0905     Train error: 0.0688

Epoch: 1                Epoch: 11               Epoch: 21
Train error: 0.2698     Train error: 0.0860     Train error: 0.0689

Epoch: 2                Epoch: 12               Epoch: 22
Train error: 0.2099     Train error: 0.0852     Train error: 0.0666

Epoch: 3                Epoch: 13               Epoch: 23
Train error: 0.1708     Train error: 0.0811     Train error: 0.0661

Epoch: 4                Epoch: 14               Epoch: 24
Train error: 0.1485     Train error: 0.0782     Train error: 0.0654

Epoch: 5                Epoch: 15               Epoch: 25
Train error: 0.1319     Train error: 0.0754     Train error: 0.0658

Epoch: 6                Epoch: 16               Epoch: 26
Train error: 0.1203     Train error: 0.0736     Train error: 0.0645

Epoch: 7                Epoch: 17               Epoch: 27
Train error: 0.1099     Train error: 0.0725     Train error: 0.0644

Epoch: 8                Epoch: 18               Epoch: 28
Train error: 0.1033     Train error: 0.0715     Train error: 0.0637

Epoch: 9                Epoch: 19               Epoch: 29
Train error: 0.0963     Train error: 0.0697     Train error: 0.0638
```




```
Epoch: 0                Epoch: 10               Epoch: 20
Train error: 0.3863     Train error: 0.0757     Train error: 0.0608

Epoch: 1                Epoch: 11               Epoch: 21
Train error: 0.2338     Train error: 0.0732     Train error: 0.0609

Epoch: 2                Epoch: 12               Epoch: 22
Train error: 0.1777     Train error: 0.0704     Train error: 0.0608

Epoch: 3                Epoch: 13               Epoch: 23
Train error: 0.1397     Train error: 0.0672     Train error: 0.0603

Epoch: 4                Epoch: 14               Epoch: 24
Train error: 0.1180     Train error: 0.0659     Train error: 0.0598

Epoch: 5                Epoch: 15               Epoch: 25
Train error: 0.1046     Train error: 0.0655     Train error: 0.0598

Epoch: 6                Epoch: 16               Epoch: 26
Train error: 0.0943     Train error: 0.0639     Train error: 0.0599

Epoch: 7                Epoch: 17               Epoch: 27
Train error: 0.0867     Train error: 0.0631     Train error: 0.0598

Epoch: 8                Epoch: 18               Epoch: 28
Train error: 0.0832     Train error: 0.0622     Train error: 0.0592

Epoch: 9                Epoch: 19               Epoch: 29
Train error: 0.0792     Train error: 0.0626     Train error: 0.0585
```




```
Epoch: 0          Epoch: 10         Epoch: 20
Train error: 0.3804 Train error: 0.0759 Train error: 0.0666

Epoch: 1          Epoch: 11         Epoch: 21
Train error: 0.2107 Train error: 0.0741 Train error: 0.0668

Epoch: 2          Epoch: 12         Epoch: 22
Train error: 0.1577 Train error: 0.0730 Train error: 0.0669

Epoch: 3          Epoch: 13         Epoch: 23
Train error: 0.1310 Train error: 0.0704 Train error: 0.0686

Epoch: 4          Epoch: 14         Epoch: 24
Train error: 0.1148 Train error: 0.0700 Train error: 0.0668

Epoch: 5          Epoch: 15         Epoch: 25
Train error: 0.1006 Train error: 0.0687 Train error: 0.0670

Epoch: 6          Epoch: 16         Epoch: 26
Train error: 0.0929 Train error: 0.0677 Train error: 0.0666

Epoch: 7          Epoch: 17         Epoch: 27
Train error: 0.0888 Train error: 0.0686 Train error: 0.0671

Epoch: 8          Epoch: 18         Epoch: 28
Train error: 0.0815 Train error: 0.0683 Train error: 0.0660

Epoch: 9          Epoch: 19         Epoch: 29
Train error: 0.0784 Train error: 0.0670 Train error: 0.0665
```

According to the graphs above, the errors converges to a certain value, which means out training is plausible.

3.2 Results

Methods	NMSE	DA
SVM+1	0.0094718	98.40810%
SVM+20	0.0158766	97.25036%
SVM+DBN(2000)	0.0117525	97.39508%
SVM+DBN(4000)	0.0119996	97.25036%
SVM+DBN(8000)	0.0120328	97.53979%

3.3 Conclusion

Comparing SVM+1 and SVM+20, the generalization performance of SVM is deteriorated when dimension of input is increasing even though more input variables contain more information

SVM+DBN converge to a small NMSE with a higher DA on the test set than SVM+1 and SVM+20, which means using DBN as feature extractor would greatly facilitate regression.

3.4 Individuals contributions

All of us were very engaged in this project. The code, presentation slides and the report were the results of our hard working and cooperating. Everyone have done their own job well.

Particularly, Jing came up with the idea for the topic and did SVM regression and conclusion part. Lulu did the background introduction and importing some function packages and calculating part. Linna did the data processing and calculating part. We all did presentation and wrote code for our own parts. Finally, we worked together for this report.

Reference

Paper:
"Feature Extraction Using Restricted Boltzmann Machine for Stock Price Prediction"
By Xianggao Cai, Su Hu, and Xiaola Lin

Indicators:
http://www.stockcharts.com

Code:
https://github.com/meownoid/tensorfow-rbm