CS230

Replacing financial charting with sequence models for trading stocks

Hiu Wai Terry Chan
Department of Management Science and Engineering
Stanford University
tchan1@stanford.edu

Songlin Qing
Department of Management Science and Engineering
Stanford University
qing0001@stanford.edu

Abstract

In this project our goal was to use deep learning to replace the technical analysis or "charting", a popular method used by Wall Street traders to predict the future direction of a stock just buy reading its historical price chart. We used LSTM based recurrent neural networks(RNNs) to predict the weekly price direction (up or down) of the returns of the S&P 500 constituent stocks using historical price changes, volume changes, and volatility time sequences as features. Despite the training AUC scores were slightly higher than a random guess of 0.5, our out-of-sample AUC scores were lower than 0.5. The result might suggests a lack of predictive power in the features and also RNN's tendency to overfit financial time series. However, our model provided valuable insight given it outputs a probability distribution of future price directions, which allows a trader to select only the most favorable situations to initiate a trade.

1 Introduction

Background

This project aims to use deep learning, in particular sequence models, to replace a commonly used technique in the financial industry called "technical analysis" or "charting". A "technical analyst" or "chartist" would try to make a prediction on the future direction of a stock by recognizing patterns from the historical charts of a stock. Although this process is highly subjective, a lot of successful traders religiously use charts to help make trading decisions. The academic community also started to study this approach given it is the exact opposite of the famous random walk theory, which believes that stock prices are random walks. Andrew Lo, an MIT professor and leading expert in technical analysis in academia, concluded that "several technical indicators do provide incremental information and may have some practical value"[1]. In fact, common technical indicators such as "moving average cross-over", "cup and handle", and "head and shoulders" are terms understood by almost all traders globally.

Why this topic

One of the authors was an equity trader and have studied and applied technical analysis since the beginning of his trading career. He strongly believes that technical analysis has merit and past prices do contain information about the future movements of a stock. Given modern deep learning technologies and the large amount of computational power, he believes that technical analysis can be done in a more objective, repeatable, and mathematically rigorous manner.

CS230: Deep Learning, Winter 2018, Stanford University, CA. (LateX template borrowed from NIPS 2017.)

2 Related work

Stock market prediction has attracted attention of deep learning researchers for its modeling difficulty due to its stochastic behavior. Ding et al. explored correlation between news and stock price by extracting structured events from news text and converting them into dense vectors, and building a prediction model based on convolutional neural network [2]. Ryo et al. worked on prediction based on numerical and textual information and leveraged recurrent neural network with LSTM to capture the long-lasting influence from words such as "financial crisis" [3]. Other prediction works were based on technical data: price, volume, and volatilities etc, with different area of focus. Singh, R., & Srivastava analyzed leveraged $(2D)_2$PCA and standard neural network to predict Google stock with 36 technical data features and achieved higher prediction accuracy compared with baseline method [4]. Nelson et al generated 175 features for each 15-mins stock data using the TA-Lib1 library and used LSTM to predict if stock price will go up or down [5]. Chen et al. divided the stock price movement direction into seven categories based on earning rate and used LSTM to predict the category of future price movement [6]. In the last two papers, accuracy was used as a primary metric but class distribution in dataset was not discussed. Recurrent neural network, together with LSTM, was the main choice of learning algorithm for previous works and thus adopted in this paper. Technical data were used as features considering the time limitation and data availability.

3 Dataset and Features

Raw Data

The dataset is the free Wiki-Prices dataset provided by a financial data startup Quandl. Quandl provides professional grade financial data at zero or very low costs. The Wiki-Prices dataset contains daily price and volume data on 3000+ stocks listed on the US exchanges going back to 1996. Each row contains “open”, “high”, “low”, “close” and “volume” for one stock on one day. These data have been adjusted for dividend and stock splits. Given the long training time required for all 3000+ stocks and the inaccuracy of price data for stocks that are infrequently traded, we chose to use only the current S&P 500 stocks as our universe. We also limited ourselves to the past 10 years of data given this increases the relevancy of using the current S&P 500 constituents as our stock universe.

Preprocessing

Preprocessing was a significant part of this project because the raw data contains data for all the stocks and dates in one csv file. Our features were: (i) price change, (ii) volume change, and (iii) volatility, for the past 52 weeks, whereas our output was a binary output of whether the current period price change was positive or negative (up or down). The motivation for using changes instead of values is to allow apples-to-apples comparison across different time periods and stocks. This would hopefully allow our model to learn a generalized model for predicting all stocks using their historical data, which is the spirit of technical analysis.

The first step of our data preprocessing involves aggregating the daily data into weekly data. This is to reduce the noise in the data and to smooth out any random fluctuations of the stocks. We used the closing price from the last trading day of the week as the weekly closing price and sum up the volume for all days of the week as the weekly volume. We then grouped the data by ticker and sorted them by date in order to calculate the sequential weekly changes of price and volume for each stock. Next we added volatility as an extra feature because it is a popular feature within the financial community. We used the annualized 12-week rolling weekly volatility as our feature given that we believe 12-week window provides enough data for a meaningful volatility calculation (which is based on the standard deviation of returns) but at the same time is frequent enough for capturing volatility changes week-over-week.

$$\textit{Annualized weekly volatility} = \sigma(\textit{weekly return}) * \sqrt{52}$$

Next we log-transformed the changes in price and volume to make values less extreme and more symmetrical given the price/volume changes can often take on extreme values. In addition, financial research have shown that asset prices tend to follow a log-normal distribution, hence taking the log before using standard scaling makes sense.

$$\textit{log-weekly return} = \log(1 + \textit{weekly return})$$

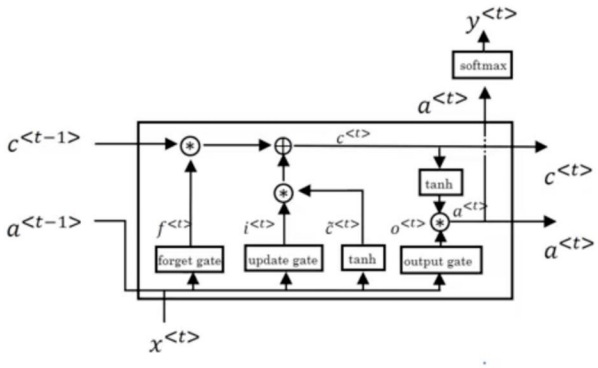
The next step was to construct the feature time series (for each of price change, volume change, volatility) for each output time step. We used the past 52 weeks for each of our three features which gave us a total of 156 features. Given we are using a long window of 52 weeks, we unfortunately would lose data at the beginning time periods for each stock and we would need to drop any time steps that do not have the full 52 weeks of features. Finally we centered and standard scaled all our features and converted our output as binary (up or down).

Comparison of data before and after preprocessing

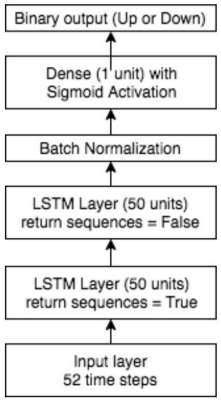
Our data started with ≈11 years worth of daily data on the price and volume of the S&P 500 constituents, which is equivalent to ≈1.8m rows of data with 2 features. After preprocessing, we ended with ≈10 years of weekly data(≈200k rows of data) with 156 features (3 features * 52 weeks).

4 Methods

Considering the time sequence nature of the stock price data, recurrent neural network (RNN) was chosen for the prediction task. Long short-term memory (LSTM) is one of the building blocks of RNN, which effectively bridges minimal time lags in excess of 1000 discrete time steps by enforcing constant error flow through "constant error carousels" within special units [7]. With a memory cell to remember value over arbitrary time intervals and an input gate, an output gate and a forget gate to regulate the values flowing through the LSTM cells, LSTM is particularly effective in extracting information from time-series data.



(a) Figure 1a: Structure of a LSTM unit



(b) Figure 1b: Network Structure

As shown in Figure 1a, $c^{<t-1>}$ is the value stored in the memory cell. $\tilde{c}^{<t>}$ is the candidate value for memory cell and calculated based on input $x^{<t>}$ and previous activation value $a^{<t-1>}$ via a tanh activation function. The update gate, forget gate, and output gate are derived based on previous activation value $a^{<t-1>}$ and input $x^{<t>}$ through a sigmoid activation function. Values of update gate and forget gate decide if $\tilde{c}^{<t>}$ replaces $c^{<t-1>}$ as the new memory cell value. New activation value $a^{<t>}$ is calculated from the memory cell value and output gate.

Our network structure is shown in Figure 1b, where the input features were fed sequentially to the 2 stacked layers of LSTM units, a batch normalization layer, a dense layer, and a single output layer with a sigmoid activation function to predict the output probability. The two layers of LSTM with 50 hidden units allow the network to capture the price data structure dynamically over time. Batch Normalization layers centers the output data from LSTM layer with unit variance to prepare the processing of a fully connected dense layer. The sigmoid function receives the output from dense layer and produces a probability value between [0,1].

Binary cross-entropy was used as the loss function for the logistic regression problem, which measures dissimilarity between true probability and predicted probability.

$$L(\hat{y}, y) = -(y log\hat{y} + (1-y) log(1-\hat{y}))$$

5 Experiments/Results/Discussion

Experiment design

Adam optimizer, an algorithm for first-order gradient-based optimization of stochastic objective functions, was adopted in the model. Adam is straightforward to implement, computationally efficient, and well suited for problems that are large in terms of data and/or parameters [8]. Default parameters values were used in the Adam optimizer: $\beta_1 = .9$, $\beta_1 = .999$, $\epsilon = 10^{-8}$.

Mini-batch size was initially chosen as 32 to leverage the noisy gradients to escape local minima. This strategy was not particularly effective as the loss quickly reduces over the first two epochs and further reduction happened at a much slower rate. Therefore, the mini-batch size was increased to 64, 128, 256, and 512 respectively while the loss reduction progress was closely monitored. 512 was the final value for mini-batch size. A higher mini-batch size provided stable estimation of the gradient of the full dataset, and leveraged GPU parallelization on the AWS system for faster training and iterations.

A grid searching of learning rate [0.00001, 0.0001, 0.001, 0.01] was performed based on observation of smoothness of training loss reduction graph over epochs and convergence speed. 0.0001 was chosen in final models after (hyper)parameter tuning. Dropout rate of 0.5 was initially applied to the model to prevent over-fitting to training data. It is later removed as we did not observe much difference between the training metrics and dev metrics.

Receiver Operating Characteristic (ROC) Area Under the Curve (AUC) was chosen as the primary metric of the test. There was slight imbalance of up or down outputs in the stock price dataset: percentage of records with stocks going up was around 9% higher than percentage of records with stocks going down. Therefore, accuracy was not an appropriate metric. AUC score is insensitive to class imbalance and a better choice for model performance.

In contrast to traditional 8/1/1 split in machine learning, more data was allocated to training dataset to improve the model performance and less data allocated to dev and test dataset. In particular, stock price change between Apr 2008 and Aug 2017 was allocated as training data, Sep-Oct 2017 was allocated as dev data, and Nov-Dec 2017 was allocated as test data. The overall percentage of train/dev/test was approximately 96/2/2.

Model construction

As described in section 3, there were 3 sets of time-sequence features: price change, volume change, and volatility. In addition to the model shown in Figure 1b (Network B), there was a simpler version with only 1 layer of 50 LSTM units (Network A). By combining the feature set and network structure, 6 models were built and evaluated against the training set and dev set. The best performer in dev set was evaluated against test set to get an estimation of the model performance in real-life scenario.

Feature Set 1: Price Change Only (1 channel)
Feature Set 2: Price and Volume Change (2 channels)
Feature Set 3: Price Change, Volume Change, and Volatility (3 channels)
Network A: 1 layer of 50 LSTM units
Network B: 2 layers of 50 LSTM units (Figure 1b)

Table 1: Model performances

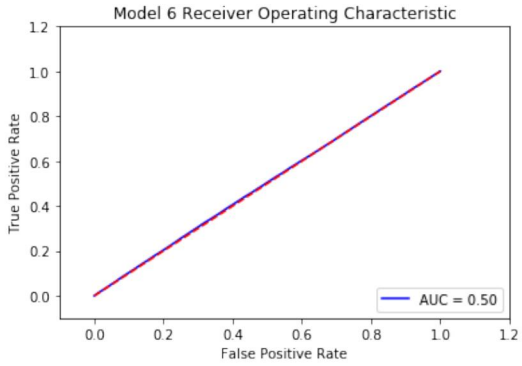
Model	Description	Training AUC	Dev AUC	Test AUC
1	Feature Set 1 + Network A	0.5011	0.4921	
2	Feature Set 2 + Network A	0.5002	0.5119	
3	Feature Set 3 + Network A	0.5011	0.4915	
4	Feature Set 1 + Network B	0.5036	0.4984	
5	Feature Set 2 + Network B	0.5014	0.5080	
6	Feature Set 3 + Network B	0.5031	0.5154	0.4857

As shown above, all 6 models have training AUC score very close to 0.5, which indicates the performance is similar to a random classifier. AUC score of 3 models dropped below 0.5 on dev set, with model 6 outperformed other models slightly. As the best performer in dev set, model 6 resulted in unsatisfactory AUC score of less than 0.5 in test set. The ROC curve for model 6 on test data is shown below in Figure 2a. Despite the low AUC score of the model in general, one interesting use of the model is to trade only predictions with high probabilities based on the distribution below (see Figure 2b). This would allow the trader to pick only the best opportunities (for example, trade only when probability is higher than 0.6 and forgo the opportunities that hover around 0.5 probability)

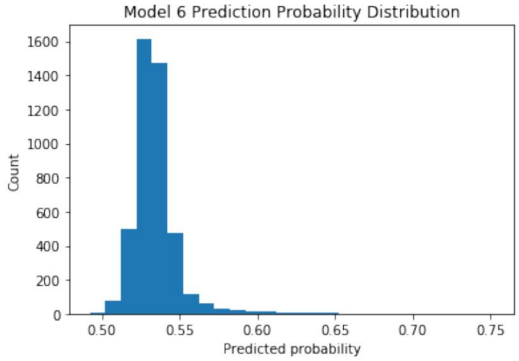
Investigation of model performance

In order to investigate if the sub-optimal performance was due to incorrect model setup or weak signal in feature sets, the following experiments were conducted.

Overfit the model: Feed the model with only 20 tickers with 6 months' data and train the model for extended number of epochs. The model eventually over-fitted to training data and produced an AUC score of 0.56. This test showed that the model was able to extract the information from training set and overfit to it. The performance of this model on dev set was poor as expected: 0.41.
Train longer: Train the models with more epochs but no improvement was observed.
Deeper network: 1 additional layer of LSTM was added but no improvement was observed.
(Hyper)parameter tuning: Learning rate and mini-batch size were adjusted but no improvement was observed .
Network setting adjustment: SGD and RMSProp optimzers were used to replace Adam but no improvement was observed .



(a) Figure 2a: Model 6 performance on test data



(b) Figure 2b: Model 6 Prediction Probability

6 Conclusion/Future Work

From the low training AUC score and unsatisfactory out-of-sample AUC scores. We can make two observations: i) there may not be enough information in the feature set for predicting future returns and ii) a RNN model is prone to overfitting financial time series and it is difficult to generalize the model to other time periods probably due to non-stationarity of financial data. From this project, we can conclude that longer-term share price prediction might need more than price/volume/volatility data in order to be accurate. Our result might also suggest that technical analysis used by Wall Street practitioners maybe more of an art than science (not rigorous statistically but may incorporate other subjective "features" such as market experiences and judgment of market sentiments).

Potential ways to improve the low AUC score in this project might include adding non-technical features such as fundamental company data (earnings, revenue, P/E ratio, etc), shortening the time frame to daily or even intraday data similar to what's practiced at high frequency trading firms, or using alternative data such as real-time news and social media feeds. These options also represent the opportunities for future extensions to this project.

7 Contributions

Both team members contributed fairly equal amount of work to this project. Terry collected and preprocessed the data, decided on the general direction of the project and the choice of features, and coded the early RNN models. Songlin was responsible for coding the models on AWS and structuring a strategy for tuning the models and presenting the results. Both team members worked together on the written parts of this project and the presentation poster.

References

Paper references

[1] Lo, A. W., Mamaysky, H., & Wang, J. (2000). Foundations of technical analysis: Computational algorithms, statistical inference, and empirical implementation. The journal of finance, 55(4), 1705-1765.

[2] Ding, X., Zhang, Y., Liu, T., & Duan, J. (2015, July). Deep learning for event-driven stock prediction. In Ijcai (pp. 2327-2333).

[3] Akita, R., Yoshihara, A., Matsubara, T., & Uehara, K. (2016, June). Deep learning for stock prediction using numerical and textual information. In Computer and Information Science (ICIS), 2016 IEEE/ACIS 15th International Conference on (pp. 1-6). IEEE.

[4] Singh, R., & Srivastava, S. (2017). Stock prediction using deep learning. Multimedia Tools and Applications, 76(18), 18569-18584.

[5] Nelson, D. M., Pereira, A. C., & de Oliveira, R. A. (2017, May). Stock market's price movement prediction with LSTM neural networks. In Neural Networks (IJCNN), 2017 International Joint Conference on (pp. 1419-1426). IEEE.

[6] Chen, K., Zhou, Y., & Dai, F. (2015, October). A LSTM-based method for stock returns prediction: A case study of China stock market. In Big Data (Big Data), 2015 IEEE International Conference on (pp. 2823-2824). IEEE.

[7] Hochreiter, S., & Schmidhuber, J. (1997). Long short-term memory. Neural computation, 9(8), 1735-1780.

[8] Kingma, D. P., & Ba, J. (2014). Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980.

Python libraries used

Numpy, Pandas, Matplotlib, Keras, Sklearn, Jupyter Notebook

Source Code

Github: https://github.com/terryhwchan/stanford_CS230
Google Drive: https://drive.google.com/open?id=130J61pJUWm9kWAMS6b1Oc8qzwtCaQlQn