Data Mining Project – Stock Market Trend Prediction

Jingguang Zhou
5140309495, wintersky@sjtu.edu.cn
Department of Computer Science, Shanghai Jiao Tong University
My partner: Haixuan Yu

Abstract

This project involves discovering how to understand the stock market trends in a data-driven approach. Project aims to use a variety of machine learning models to make predictions regarding the stock price movements based on the factors such as askprice, lastVolume, lastTurnover. etc. Those factors can reflect and affect investor reactions and represent the trend of the price. We process the data and add new features. We construct the predicting system combined with seven models (Lstm, Lstm-naive, Conv1D, Conv2D, MLP, TimeDistributed, GradientBoosting) to capture the feature and make prediction. We invent a lazy dataloader, so that our system has well copied with memory deficit and thus can be well applied to very large dataset. We ensemble our models to further improve the prediction. Extensive experiments are conducted to evaluate the capability of these models.

Introduction

As we all know, the chaotic system like the stock market is basically impossible accurately predicted. But still if we can reduce the risk of loss, then we can make significant progress. The appropriate machine learning algorithm with good models and feature selection can help improve the probability of success.

Our underlying assumption is that a strong correlation between the history of market price behavior and the future trend of price. We use ten features that TA gives us and then extracted two extra features from this time series data, indicating that it can provide meaningful, actionable slice of real market conditions and psychology.

Our system tends to explore the impact of financial report factors, such as LastPrice, Volume, Turnover, on short-term stock market trends during the after-market time period and the next trading date.

In sum, the input to our algorithm is divided into two parts. One is a seqlen*12-vector features which contains digital data from the reports for the market (LastPrice, Volume, LastVolume, Turnover, LastTurnover, AskPrice, BidPrice, AskVolumn, BidVolumn, OpenInterest, Newone, Newtwo). Here we set seqlen to be 100, so we have features with shape of 100*12. The other part is seqlen*1-vector labels. We look forward 20 ticks then compare that price with our current price. If it is bigger than the current price, we mark the label as 1, otherwise 0. So we have labels with shape of 100*1.

We processed input features with muiltiple models and ensemble of some models. Lstm, Lstm-naive, Conv1D, Conv2D, MLP, TimeDistributed, GBDT are used to to output the prediction of stock markets trends 20 ticks afterwards. Extensive experiments are taken to show the effects.

In summary, Our main contributions include:

- Process the data (clean, normalization, add new feature) and use prior 100(seqlen=100) ticks to predict the 20 ticks afterwards.
- Construct a Lazy Dataloader. It helps to decrease the memory use from (samples $\times$ seqlen $\times$ features) to (samples $\times$ features). If the seqlen is 100, we can reduce the memory use to its 1/100. Thus we can simultanously load all the data given by TA into our laptop.
- Try seven neural network approaches and one non-neural appoach(Gradient Boosting Tree). We have given our results and associated comparison in detail.
- Explore the influence of different seqlens of lstm model.
- Ensemble different models to give a further improvement.
- Consider different metrics besides the total accuracy to evaluate the performance.
- Discuss the ways against overfitting and the validation of our results.

Related work

Stock price prediction has long been a popular topic. In the past 10 years, many works have been done on application of machine learning techniques for stock price prediction. SVM has been proved a powerful algorithm by researches, including Subhabrata Choudhurys [1] and Jigar Patels work[2]. In addition to SVM, genetic algorithms [3] [4] , Decision Tree [5], [6], and some other regression models [7] [8] have been widely utilized in this topic. Moreover, in recent years, many researches have been done on applying Neural Networks to predict the stock price, including [9], [10]. Some of these studies tend to be quite successful, such as [10] used Maple as measurement, and achieved 0.98 accuracy. Also, the idea of using indirect information, such as text information, to predict stock price has also been widely

	LastPrice	Volume	LastVolume	Turnover	LastTurnover	AskPrice1	BidPrice1
count	239006.000000	2.390060e+05	239006.000000	2.390060e+05	2.390060e+05	239006.000000	239006.000000
mean	1776.275416	2.896786e+06	121.783340	5.146915e+10	2.159941e+06	1776.784231	1775.783955
std	10.594685	1.709080e+06	452.918221	3.032723e+10	8.060503e+06	10.587543	10.586537
min	1747.000000	3.234000e+03	1.000000	5.724180e+07	1.000000e+00	1748.000000	1747.000000
25%	1771.000000	1.476480e+06	1.000000	2.629872e+10	1.000000e+00	1771.000000	1770.000000
50%	1777.000000	2.688912e+06	4.000000	4.790352e+10	7.118000e+04	1777.000000	1776.000000
75%	1782.000000	4.181490e+06	32.000000	7.438716e+10	5.686200e+05	1782.000000	1781.000000
max	1844.000000	6.700936e+06	34306.000000	1.204410e+11	6.103974e+08	1846.000000	1845.000000

AskVolume1	BidVolume1	OpenInterest	New_one	New_two
239006.000000	239006.000000	2.390060e+05	239006.000000	2.390060e+05
3062.595186	2993.326339	3.106140e+06	44.787483	1.020437e+06
1521.248329	1583.119750	9.787864e+04	1278.696621	1.459557e+06
1.000000	1.000000	2.884978e+06	-10920.000000	5.258148e-03
2038.000000	1940.000000	3.085492e+06	-182.741071	5.492862e+00
2812.000000	2719.000000	3.136766e+06	0.956214	4.372788e+01
3790.000000	3770.750000	3.170454e+06	251.500000	3.092048e+06
20239.000000	14297.000000	3.321334e+06	15524.000000	3.321328e+06

Figure 1: The rough impression of the dataset

studied. Hagenau proposed a model based on financial news using context-capturing features in 2013 [11], and Shynkevich used different categories of articles to do predictions [12]. Other works include the AZFintext system proposed by Schumaker [13], and etc. In these works, the most widely studied issue is how to extract valuable information related to stock price prediction. However, bag of words [13] and noun phrases [11] are most commonly applied technique, but generally cannot produce satisfactory result. Thus in our work we utilize the direct information as features to predict the trends of the stock market. We propose multiple models to capture the dynamics of the data and construct a robust system that is scalable and robust to large data.

Data preprocess

The Dataset is given by TA and we use the data from DBExport. There are 47 excel files, each of which contains the trading information in a week. First, we clean the data and remove those features filling with 0 values, i.e. AskPrice5,AskPrice4,AskPrice3,AskPrice2, BidPrice2, BidPrice3, BidPrice4, BidPrice5, AskVolume5, AskVolume4, AskVolume3, AskVolume2, BidVolume2, BidVolume3, BidVolume4, BidVolume5, UpperLimitPrice, LowerLimitPrice. Then we drop InstrumentID, TradingDay, UpdateTime, UpdateMillisec because they will not be used in our training process. After dropping those feature entries, there remains 10 features(LastPrice, Volume, LastVolume, Turnover, LastTurnover, AskPrice, BidPrice, AskVolumn, BidVolumn, OpenInterest, Newone, Newtwo).

In addition to these features, we add two **extra features**.

- *(data['AskVolume1']-data['BidVolume1'])/data['LastVolume']*
- *(data['OpenInterest'] / data['LastTurnover']*

But if we do this directly, it will give us a bad result, that is, all the output in our trainin g process will go to NAN or INF. After a careful check, we find out some values of our origin data are actually 0 or 9999999. Thus, to avoid 0 to be divided, we smooth the data and add to 1. Then these two new features can be applied into our system. The data is illustrated in Figure 1.

Then we then use StandardScaler() in sklearn to normalize the feature to uniform Gaussian format. To learn the dynamics of the time-series data, we not only use the current tick to train, but also combine with the prior seqlen ticks to predict the trends. Here we set seqlen to be 100. We predict whether it will go up or down after 20 ticks. That means, if the price remains still, we just look forward standing on the position of 20 ticks, maybe 21 or 22 ticks or more. We make sure we can get whether it is up or down because it represents the trend. Thus, we have features with shape of seqlen*12 and labels with seqlen*1. Here seqlen=100.

By the way, all of our implementation is vector/matrix operations, which avoid for-loops and give a fast speed especially suited to the pre-process of large dataset.

Lazy DataLoader

We notice that all 47 files are totally several GB. Since we use the prior 100 ticks to predict the trend, the size of total training data should multiply 100 to be several HUNDRED GB. Absolutely it is not wise because of both cpu memory error and gpu memory deficit. How can we deal with the problem?

We think of a way of lazy dataloading. The lazy means we do not prepare samples*seqlen*12 until we have to use the batch. To put it in other ways, we do not prepare samples*seqlen*12 input data, instead we only generate batchsize*seqlen*12 when we have to use the batch. If not used, we just lazily keep the indices of the data. When it is sampled, we just yield the batch associated with the index.

Implementation in detail is included in our submitted code. We cope with this in two different methods. I just give a rough impression. The one is using fit_generator and write a custom generator function to load the data. The other is using fit_generator but the difference is to write a custom class seqdataloader inherited from the default class torch.dataloader. Two approaches are included in our codes, but the advantage of the latter approach is that we can shuffle the data easily. Considering we use lazy dataloading, it will be stange to index and shuffle the data directly in a custom generator function. Thus we use a new class inheriting many good features of the default class to help deal with this.

The result is that we can simultanouslly train all data of these 46 files in our 4G-memory laptop.

Neural network methods

After preprocess and load the data, we have tried different models to fit the data. Actually only fitting the data is not enough, because they just fit the training set instead of generalizing on the testing set.Thus we hope our model can have a better understanding of the data, but of course they need to fit the training data first.

We propose seven models, and next we will illustrate the layers in Figure 4, 10, 6, 2.

MLP

At first, we have tried MLP model, i.e. using fully connected layer as Figure 2. We flatten all the features to one dimension and then connect them to next layers. However, we find that the mlp model have a poor performance–the loss does

not converge even after 1000 epochs. Thus we give up the simplest model.

Layer (type)	Output Shape	Param #
flatten_5 (Flatten)	(None, 1000)	0
dense_9 (Dense)	(None, 100)	100100
dense_10 (Dense)	(None, 1)	101

Figure 2: MLP model

Conv1D

The origin CNN architecture looks like Figure 3. Considering that CNN can help extract the features layer-by-layer automatically, we construct a CNN network. But here we first construct Conv1D layers, illustrated in Figure 4.

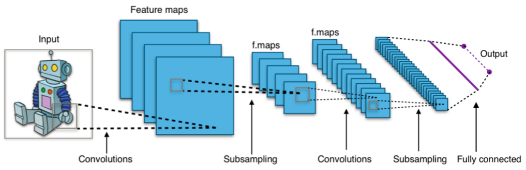

Figure 3: CNN architecture

Layer (type)	Output Shape	Param #
conv1d_1 (Conv1D)	(None, 98, 64)	1984
conv1d_2 (Conv1D)	(None, 96, 64)	12352
conv1d_3 (Conv1D)	(None, 94, 128)	24704
conv1d_4 (Conv1D)	(None, 92, 128)	49280
flatten_1 (Flatten)	(None, 11776)	0
dropout_1 (Dropout)	(None, 11776)	0
dense_2 (Dense)	(None, 1)	11777

Figure 4: Conv1d model

We focus on Conv1D layer because for each tick there are 12 dimensions. We hope 1D convolution will abstract the advanced features for this tick. However, we find that the Conv1D can neither give us a good result.

Conv2D

I come up with an idea that we can utilize the relations between ticks, especially ticks that are closed in time. Then we can extend CNN with Conv1D layers to CNN with Conv2D layers. The size of convolution filters becomes 3×3.

Time Distributed

Illustrated in Figure 6, Time Distributed model is actually another improvement on Conv1D model. We notice that there are no spatial-local relationships between the features of one tick, then cnn filter may be too small to capture the advance information. Thus we change another direction–for each tick, we try to use fully connected layers(Dense layer)

Layer (type)	Output Shape	Param #
reshape_1 (Reshape)	(None, 100, 10, 1)	0
conv2d_1 (Conv2D)	(None, 100, 10, 32)	320
conv2d_2 (Conv2D)	(None, 100, 10, 64)	18496
conv2d_3 (Conv2D)	(None, 100, 10, 32)	18464
flatten_1 (Flatten)	(None, 32000)	0
dense_1 (Dense)	(None, 256)	8192256
dropout_1 (Dropout)	(None, 256)	0
dense_2 (Dense)	(None, 1)	257

Figure 5: Conv2d model

to collect features and output new ones. The experiment shows that this method works a lot.

Layer (type)	Output Shape	Param #
time_distributed_3 (TimeDist	(None, 100, 10)	110
time_distributed_4 (TimeDist	(None, 100, 5)	55
flatten_4 (Flatten)	(None, 500)	0
dropout_1 (Dropout)	(None, 500)	0
dense_12 (Dense)	(None, 1)	501

Figure 6: TimeDistributed model

Lstm-naive

We know that RNN is good for predicting the time sequences. But vanilla RNNs will suffer from vanishing gradient or exploding gradient. Thus Long short-term memory network(LSTM) illustrated in Figure 10 was proposed to cope with the problem. LSTMs have a cell that stores the previous values and hold onto it unless a ”forget gate” tells the cell to forget those values. LSTMs also have a ”input gate” which adds new stuff to the cell and an ”output gate” which decides when to pass along the vectors from the cell to the next hidden state. LSTMs simply add a cell layer to make sure the transfer of hidden state information from one iteration to the next is reasonably high. Put another way, we want to remember stuff from previous iterations for as long as needed, and the cells in LSTMs allow this to happen.

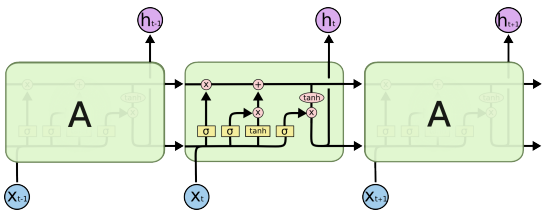

Figure 7: LSTM chain

In our implementation of Lstm-naive model in Figure 8, we just use one-layer lstm and only keep the last hidden unit as our output. Of course we can expand the outputs to be

multiple ticks afterwards, but we argue that will exacerbate overfitting to some extent. Thus we output only last one hidden unit. We find out that the lstm-naive model takes effect, outperforming the prior CNN model. We get a training accuracy of 0.8648 while the validation accuracy is 0.7305.

Layer (type)	Output Shape	Param #
lstm_1 (LSTM)	(None, 32)	5504
dense_3 (Dense)	(None, 1)	33

Figure 8: LSTM-naive model

Lstm-stack

Since one-layer LSTM may only capture some shallow features, we would like to introduce a new method call stack lstm. Figure 9 is a two-layer stacked Lstm. We believe by stacking lstm, we can capture high-level features from the time sequences, which is conducive to the trend prediction.

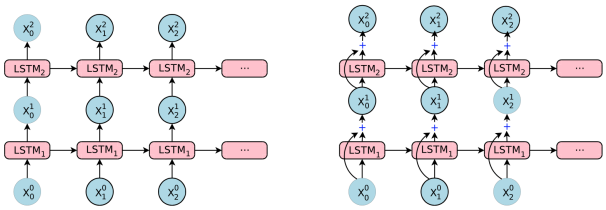

Figure 9: Lstm-stack model

Layer (type)	Output Shape	Param #
lstm_4 (LSTM)	(None, 100, 32)	5504
lstm_5 (LSTM)	(None, 32)	8320
dense_3 (Dense)	(None, 1)	33

Figure 10: Lstm model

Experimental evaluation

We conducted extensive experiments to show the validity of our models. Our dataset is the first csv file that TA gave us. Training set is the first 200000 samples and validation set is the last 40000 samples. The results are as follows.

The performance of different models

The input are of 10-dim origin features and new-added 2-dim features. We use the prior 100 ticks(seqlen=100) as our training samples to predict the trend of 20 ticks afterwards. We trained 200 epochs for each model and the result is illustrated as Figure 11, 12.

We find out that the CNN model, including Conv1D and Conv2D performs bad. We analyse the possible reasons as follows:

- No necessary spatial locality in our time sequence data.

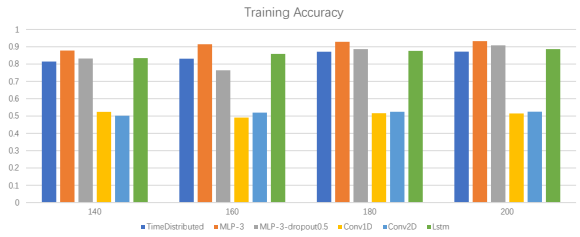

Figure 11: Comparison on Train Accuracy. The x-axis refers to training epochs for these models except LSTM. For LSTM we only sample the 4, 8, 12, 16 epochs because it converges faster than others.

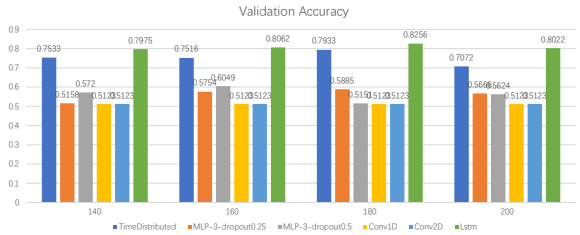

Figure 12: Comparison on Validation Accuracy. The x-axis refers to training epochs for these models except LSTM. For LSTM we only sample the 4, 8, 12, 16 epochs because it converges faster than others.

- The size of the filter may be not big enough. But if we set the size to be the dim of the data, say 12, the CNN model will be degraded to a MLP model.
- No special tricks on CNN model. Facebook uses a lot of tricks to improve CNN network on machine translation tasks. But we did not do this. We just use the vanilla CNN with Dropout and BatchNormalization layers, which seems not fit the data.

The MLP model cannot converge either. Time-Distributed model performs so well that it almost catches up the LSTM. The LSTM performs best because it captures the time-variant features.

Against overfitting

We admit there exists that the training accuracy is higher than validation accuracy because of too much noise in the financial data. However, multiple methods are used in our training process to relieve the impact of overfitting.

- Dropout

- Earlystopping
- Using Lazy DataLoader to allow large-scale data to be loaded into our memory.

The influence of seqlen over the performance

Since we know that LSTM gives a best performance, we would like to explore the impact of seqlen over the performance. Originally the seqlen is set to be 100. The result is shown as Figure 13, 14.

Figure 13: Comparison on Train Accuracy with different seqlen of lstm. The x-axis refers to training epochs.

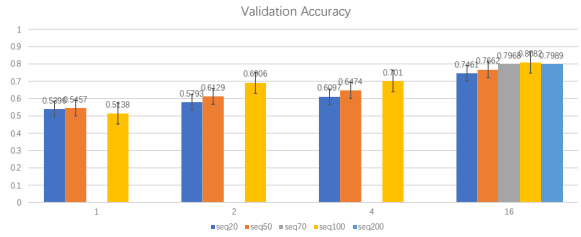

Figure 14: Comparison on Test Accuracy with different seqlen of lstm. The x-axis refers to training epochs.

We find out as the seqlen increases, the model will give us better predicion results. However, the edge will vanish as the seqlen becomes larger and larger. When it increases to 200, the model conversely performs a lower accuracy than the counterpart model with seqlen being 100. Perhaps because too large seqlen will bring more noises and uncertainties to our model.

Gradient Boosting Tree

Unlike neural network methods, Gradient Boosting Tree is an ensembling method in statistical machine learning. Gradient Tree Boosting or Gradient Boosted Regression Trees (GBRT) is a generalization of boosting to arbitrary differentiable loss functions. GBRT is an accurate and effective off-the-shelf procedure that can be used for both regression and classification problems. Gradient Boosting Tree is merited for its robustness to outliers in output space via robust loss functions.

We also tried this method, finding that the training accuracy is 0.8070 and the validation accuracy is 0.5627.

The ensemble of several models

We ensemble different models which are saved from the last epoch as Figure 15. We use lstm with different seqlens to ensemble, which is called Multi-scale learning(from 80.8 to 84.9). Then we add TimeDistributed model, achieving another improvement (from 84.9 to 86.6).

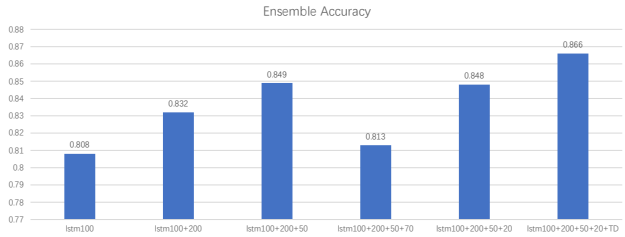

Figure 15: The ensemble of different models. Lstm200 means that seqlen=200 for lstm model. TD means TimeDistributed model. We find that combining multi-scale of Lstm and TD will give a best result.

Different metrics

We not only consider the overall accuracy, but also consider the respective accuracy when the groundtruth trend is increasing or decreasing. We argue that these two metrics will provide us more information about our models.

In our dataset, there are 38887 groundtruth samples, among which 19921 samples are going up and the other is going down. We use the best ensemble model lstm100+200+50+20+TD.

For those groundtruth samples with increasing trend, our model has a prediction accuracy of 0.875. For those groundtruth samples with decreasing trend, our model has a prediction accuracy of 0.831.

Discussion on why the accuracy is so high

- First we are confident to say that all our results are easily reproducible. Because when we train our model, we did not choose the best one with least validation loss. We always choose the last model even its loss is slightly higher than the best one. If you choose the best one, the results will be better. The results can be shown in the ipynb file.

- In fact, the performance of a model will depend on the specific dataset. We use the first csv file containing a-week's data for our training and validation. We would like to say our model give strong prediction on this dataset. Maybe if we travel through time and buy the stock according to our model, we will make a fortune.
- Certainly, we have a good design of our model so that they should be adapted to different situations. We also have added new features according to their effects. They have their underlying tendency to give us a better prediction. If we use MLP, CNN, gradient boosting, that will give a relatively low results as Figure 12. But our lstm, TimeDistributed model will give a good prediction.

References

[1] Subhabrata Choudhury, Subhajyoti Ghosh, Arnab Bhattacharya, Ki- ran Jude Fernandes, and Manoj Kumar Tiwari. A real time clustering and svm based price-volatility prediction for optimal trading strategy. Neurocomputing, 131:419426, 2014.

[2] Jigar Patel, Sahil Shah, Priyank Thakkar, and K Kotecha. Predicting stock and stock price index movement using trend deterministic data preparation and machine learning techniques. Expert Systems with Applications, 42(1):259268, 2015.

[3] Esmaeil Hadavandi, Hassan Shavandi, and Arash Ghanbari. Integration of genetic fuzzy systems and artificial neural networks for stock price forecasting. Knowledge-Based Systems, 23(8):800808, 2010.

[4] Kyoung-jae Kim and Ingoo Han. Genetic algorithms approach to feature discretization in artificial neural networks for the prediction of stock price index. Expert systems with Applications, 19(2):125132, 2000.

[5] Jar-Long Wang and Shu-Hui Chan. Stock market trading rule discovery using two-layer bias decision tree. Expert Systems with Applications, 30(4):605611, 2006.

[6] Robert K Lai, Chin-Yuan Fan, Wei-Hsiu Huang, and Pei-Chann Chang. Evolving and clustering fuzzy decision tree for financial time series data forecasting. Expert Systems with Applications, 36(2):37613773, 2009.

[7] Sheng-Hsun Hsu, JJ Po-An Hsieh, Ting-Chih Chih, and Kuei-Chu Hsu. A two-stage architecture for stock price forecasting by integrating self- organizing map and support vector regression. Expert Systems with Applications, 36(4):79477951, 2009.

[8] Theodore B Trafalis and Huseyin Ince. Support vector machine for regression and applications to financial forecasting. In IJCNN (6), pages 348353, 2000.

[9] Reza Hafezi, Jamal Shahrabi, and Esmaeil Hadavandi. A bat-neural network multi-agent system (bnnmas) for stock price prediction: Case study of dax stock price. Applied Soft Computing, 29:196210, 2015.

[10] Jonathan L Ticknor. A bayesian regularized artificial neural network for stock market forecasting. Expert Systems with Applications, 40(14):55015506, 2013.

[11] Michael Hagenau, Michael Liebmann, and Dirk Neumann. Automated news reading: Stock price prediction based on financial news using context-capturing features. Decision Support Systems, 55(3):685697, 2013.

[12] Yauheniya Shynkevich, TM McGinnity, Sonya Coleman, and Ammar Belatreche. Stock price prediction based on stock-specific and sub- industry-specific news articles. In 2015 International Joint Conference on Neural Networks (IJCNN), pages 18. IEEE, 2015.

[13] Robert P Schumaker and Hsinchun Chen. Textual analysis of stock market prediction using breaking financial news: The azfin text system. ACM Transactions on Information Systems (TOIS), 27(2):12, 2009.