High Frequency Trading Prediction

Xinyue Chen
Department of Computer Science and Engineering
Shanghai Jiaotong University
Minhang, Shanghai, 200240
kiwisher@sjtu.edu.cn

Abstract

To deal with the task of stock prediction, we first conduct exploratory data analysis on both training and testing data. Based on pattern and characteristics of the data, we then try various ways to preprocessing the data. We also investigate feature engineering with respect to financial knowledge and basic trading strategies. The modeling is a natural highlight of this task, as the problem can be formulated differently as classification, regression or markov process. We consider in particular six methods- linear regression, tree-based regression, tree-based classification, neural network regression, deep markov model, and **stacking ensembles of heterogenous models**.

This paper will give experimental details of both feature engineering and the modeling itself, including several failed attempts.

We submit two models according to public scores. One is LSTM regression (public: 0.00148, private: **0.00143**), another is Xgboost regression (public:0.00147, private: 0.00145). Actually an ensemble of the above models scores slightly higher in private leaderboard (public:0.00148, private: **0.00142**). But we did not submit that version.

My team mate is Hao-Tian Tang and we cooperated throughout this task.

1 Introduction

In recent years, stock prediction has become a field of interest in the cross domain of computer science and financial trading. A set of factors come into play, including macro ones such as political news, demand and supply of commodities, economy of the nation, or micro ones specific to a few stocks or a particular time stride, such as weekdays, holidays, trading time, short-term market confidence, intraday demand-supply relation inside a section or a stock.

This project focuses particularly on **HFT** (high-frequency trading) of one given stock in a given period of time. Order book from June, 2018 to September, 2018 are taken as training data. Typical time lapse between records is **3 seconds**. And the task is to predict the stock′s average mid price of the following 20 timestamps given information in 10 timestamps, where mid price is the average of bid price 1 and ask price 1. Information provided for each record includes: date, time, last price, bid price 1, ask price 1, bid volume1, ask volume1 and volume.

We regard this project as a time-series prediction, due to the temporal nature of the data. A conventional way to deal with such data is **HMM**. But these years the thriving of deep learning gives us another choice, **RNN**. Traditional machine learning methods such as boosting trees or linear regression also stand a chance as temporal characteristics are not very profound.

When doing short-term trading, one thing that people constantly refer to is the **candlestick chart**. So it is reasonable to assume there is correlation between historical information and future price. Traders will also craft a few features out of the original data to distill information and more precisely indicate the trend of stock prices as well as the market′s response. That gives us motivation to investigate such metrics as **micro price**.

In this paper, we explore the provided data, propose ways for data preprocessing and building validation set. Then we introduce one of our base model, LSTM, followed by our ensemble of different tree-based models, with a technique called **stacking ensemble**. After that, we explain in detail our failed attempts, including combining historical features, introducing micro price, linear regression, **OHEM** (Online Hard Example Mining), predicting the whole sequence, and **Deep Markov Model**.

2 Related Work

2.1 Linear Regression

Linear regression is a linear approach to modelling the relationship between a scalar response and one or more explanatory variables.

Linear regression models are often fitted using the least squares approach, but they may also be fitted in other ways, such as by minimizing the ”lack of fit” in some other norm, as with least absolute deviations regression, or by minimizing a penalized version of the least squares cost function as in ridge regression(L2-norm penalty) and lasso (L1-norm penalty). Conversely, the least squares approach can be used to fit models that are not linear models. Thus, although the terms ”least squares” and ”linear model” are closely linked, they are not synonymous.

The basic principle in matrix notation is as follows:

$$\mathbf{y} = X\beta + \varepsilon \tag{1}$$

where X denotes a set of vectors known as explanatory variables or regressors, and y denotes a vector of observed values of the variable called regressand or response variable.

2.2 Boosting Method

Boosting is a ensemble meta-algorithm for primarily reducing bias, and also variance in supervised learning, and a family of machine learning algorithms that convert weak learners to strong ones. Boosting is based on the question :"Can a set of weak learners create a single strong learner?"

A weak learner is defined to be a classifier that is only slightly correlated with the true classification (it can label examples better than random guessing). In contrast, a strong learner is a classifier that is arbitrarily well-correlated with the true classification.

While boosting is not algorithmically constrained, most boosting algorithms consist of iteratively learning weak classifiers with respect to a distribution and adding them to a final strong classifier. When they are added, they are typically weighted in some way that is usually related to the weak learners' accuracy. After a weak learner is added, the data weights are readjusted, known as "re-weighting". Misclassified input data gain a higher weight and examples that are classified correctly lose weight. Thus, future weak learners focus more on the examples that previous weak learners misclassified.

In our project, we investigate various boosting algorithms for regression, including XGBoost(**section 3.3**).

2.3 LSTM

A most natural way to tackle time-series is LSTMs. Here we briefly introduce the intuition behind LSTMs.

They are a special kind of RNN, capable of learning long-term dependencies. They were introduced by Hochreiter and Schmidhuber (1997), and were refined and popularized by many people in following work. They work tremendously well on a large variety of problems, and are now widely used.

The key to LSTMs is the cell state, the horizontal line running through c_{t-1} to c_t, which we have explained in detail in our sharing.

The LSTM has the ability to remove or add information to the cell state, carefully regulated by structures called gates. Gates are a way to optionally let information through. They are composed out of a sigmoid neural net layer and a pointwise multiplication operation. The sigmoid layer outputs numbers between zero and one, describing how much of each component should be let through. A value of zero means let nothing through, while a value of one means let everything through.

2.4 Reinforcement Learning

In the viewpoint of reinforcement learning, stock trading process is typically modeled as a Markov Decision Process (MDP).

The basic ingredients of reinforcement learning are *state*, *action*, *reward*, *policy* and *action-value function*. The specific definition of these ingredients should be in line with the task itself.

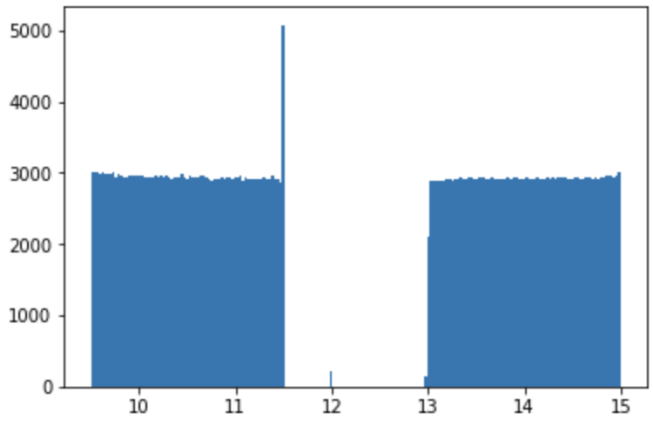

Figure 1: Timestamp distribution in training data in a day.

Recently, there is a paper (Zhuoran Xiong 2019) accepted by NIPS 2018, which applies reinforcement learning to stock trading.

They explore a deep reinforcement learning algorithm, namely Deep Deterministic Policy Gradient (DDPG), to find the best trading strategy in the complex and dynamic stock market. This algorithm consists of three key components: (i) actor-critic framework that models large state and action spaces; (ii) target network that stabilizes the training process ; (iii) experience replay that removes the correlation between samples and increases the usage of data. The efficiency of DDPG algorithm is demonstrated by achieving higher return than the traditional min-variance portfolio allocation method and the Dow Jones Industrial Average (DJIA).

Indeed, as our training data is quite limited not only in time span (3 months compared to their several years) but also stock information (orderbook for one stock only compared to that of 30 stocks), we think that reinforcement learning may not be an effective approach to our problem. In addition, the high-frequency nature of our data is not compatible with the reinforcement learning trading modeling where long-term profit is the ultimate goal. Thus, we regard our problem mainly as a time-series prediction problem instead of learning a trading strategy for accumulated profits.

3 Proposed Method

In this section, we introduce our two models. One is based on LSTM, and the other is an ensemble of regressions. Before that, we explain part of our data analysis, also our data preprocessing pipeline as well as our method of splitting training-validation data.

3.1 Data Analysis, Preprocessing and Validation

3.1.1 Data Distribution Analysis After visualizing both the training data and the test data. We have a few interesting observations.

- **Data frequency in training data is inconsistent.**

The typical time lapse is **3 seconds**. But there are plenty of aberrant time lapse from 1 second to over 7 seconds. Our strategy here is to filter out all the series that have time lapse other than 3 seconds.

- **Invalid timestamps.**

  Refer to **Figure 1** for an overall describe of timestamp distribution. In that figure, the bin is set to 200. The trading sessions for A shares is 9:30 to 11:30 and 13:00 to 15:00. But we can see timestamps existing at 12:00, and also **a few minutes before 13:00**. The bar at around **11:30** is also abnormal. So we just remove all these invalid timestamps based on their trading time.

- **Behavioral gap between time intervals.**

  It is natural to assume that the close state and the open state between days are not continuous with respect both market behaviors and stock price. In fact, the transition between close state in the morning and open state in the afternoon are not naturally close neighbors either.

  One reason is that there could be market news (shares splits, shares repurchase, merge and acquisition) posted out typically in the noon or after market news. So **information gain** naturally splits apart days, mornings and afternoons.

- **Discrepancy between training and testing data distributions.**

  Clearly the distribution of training data and testing data are quite different (**Table 1**). If we feed unprocessed prices into a model, chances are that it may behave reasonably bad because most of the testing data fall into ranges that it had never seen during training.

$Data$	$mean$	std	min	max
Train	3.487	0.152	3.258	3.870
Test	3.242	0.094	3.075	3.433

Table 1: mid price features in training and test datasets

Therefore, we propose

- Incremental prediction
- Normalization within batches

Details of these methods are given in **section 3.1.3**.

3.1.2 Handcrafted Features We are given 8 features for each time stamp: Time, MidPrice, LastPrice, AskVolume1, AskPrice1, BidPrice1, BidVolume1, Volume. Some of them are strongly correlated. MidPrice is just the average of AskPrice1 and BidPrice1, the information gain from LastPrice is also limited.

Given the low-dimensional features, we try to increase the dimensionality for each training sample by crafting new features.

The new features that we have explored includes:

- **Micro Price**

$$MicroPrice_t = \frac{BV1 \cdot AP1 + AV1 \cdot BP1}{BV1 + AV1} \quad (2)$$

  where $AP1$ stands for AskPrice1, $BP1$: BidPrice1, $AV1$: AskVolume1, $BP1$: BidVolume1.

  The intuition of micro price is to leverage demand and supply based on both prices and their volumes. For example, if BidVolume1 outweighs AskVolume1 a lot, then the new settle price will favor AskPrice1 more, because there are a lot of people who wants to buy.

  Actually we have experiments about mid price and micro price. It turned out that **using micro price alone for regression is better than using mid price alone**. The reason may be that micro price integrates information of both prices and volumes.

- **Exact Trading Time**

  Morning or afternoon is not the only time factor affecting market and prices. Instead of split data into morning and afternoon, we try to add time for each timestamp as a new feature. Specifically, we treated time as a float number based on hour, minute, and second of the timestamp.

- **Weekday**

  Markets behave differently in different weekdays. In particular, people are statistically more optimistic on Mondays. Based on this, we add a *weekday* feature and investigate its influence on mid price.

- **Volume Difference**

  Volume is accumulated throughout test data and training datasets. The value of it is also quite large compared to the other features. Therefore we apply row-wise difference to the Volume column.

- **Price Difference Ratio**

  In stock trading, we care more about price change percentage than the exactly difference between prices. So we also try replacing price differences with price difference ratios.

  But as we are now only dealing with one stock, the difference between price difference ratio and price difference is not profound.

The correlations between different features are illustrated in **Figure 2**. The first column demonstrates correlation between mid price and the other features. Also, the row of *micro price* is in line with the definition of micro price.

Note that we do not use all of these features in every model. It turns out that we can not observe obvious difference in public score with the addition of some features in some models, but whether they have a influence on private score is unknown. Public and private do not go hand in hand within a fluctuation of 0.0004.

3.1.3 Preprocessing

We tried

- **Incremental prediction.**

	MidPrice	LastPrice	Volume	BidPrice1	BidVolume1	AskPrice1	AskVolume1	Weekday	Daytime	MicroPrice	VolumeDiff	MidPriceDiff	M
MidPrice	1.000000	0.999989	-0.187397	0.999999	0.050993	0.999999	-0.033491	0.041532	0.002877	0.999996	0.000228	0.000644	
LastPrice	0.999989	1.000000	-0.187352	0.999988	0.051059	0.999988	-0.033640	0.041553	0.002913	0.999986	0.000225	-0.000205	
Volume	-0.187397	-0.187352	1.000000	-0.187347	0.066841	-0.187447	0.103486	0.066405	0.716541	-0.187396	0.022046	0.003319	
BidPrice1	0.999999	0.999988	-0.187347	1.000000	0.050991	0.999994	-0.033547	0.041533	0.002926	0.999995	0.000249	0.000610	
BidVolume1	0.050993	0.051059	0.066841	0.050991	1.000000	0.050996	0.057603	0.027867	0.012303	0.051687	0.005925	-0.022816	
AskPrice1	0.999999	0.999988	-0.187447	0.999994	0.050996	1.000000	-0.033435	0.041531	0.002828	0.999994	0.000208	0.000678	
AskVolume1	-0.033491	-0.033640	0.103486	-0.033547	0.057603	-0.033435	1.000000	0.009697	0.053087	-0.034782	0.002910	0.049704	
Weekday	0.041532	0.041553	0.066405	0.041533	0.027867	0.041531	0.009697	1.000000	0.000329	0.041540	-0.000266	0.000626	
Daytime	0.002877	0.002913	0.716541	0.002926	0.012303	0.002828	0.053087	0.000329	1.000000	0.002863	0.017592	0.002787	
MicroPrice	0.999996	0.999986	-0.187396	0.999995	0.051687	0.999994	-0.034782	0.041540	0.002863	1.000000	0.000227	0.000340	
VolumeDiff	0.000228	0.000225	0.022046	0.000249	0.005925	0.000208	0.002910	-0.000266	0.017592	0.000227	1.000000	0.085251	
MidPriceDiff	0.000644	-0.000205	0.003319	0.000610	-0.022816	0.000678	0.049704	0.000626	0.002787	0.000340	0.085251	1.000000	
MidPriceDiffRatio	0.000702	-0.000145	0.003288	0.000668	-0.022759	0.000736	0.049501	0.000652	0.002723	0.000399	0.082386	0.999248	

Figure 2: Correlation between features.

We tried both predicting delta between the 10_{th} mid price and the average of the next 20 timestamps, and predicting the ratio of difference between them.

The intuition of incremental prediction is that models generally work better when the output is supposed to be symmetric about zero. This greatly decreases the epoches needed for convergence.

This technique is really key to higher score, glad that after our sharing, our classmates' success validates this technique.

- **Batch normalization.**

  In our experiment, batch normalization is no superior to global normalization. They score almost the same in public board. On the other hand, We tried both min-max normalization and z-normalization, and unfortunately, there's no clear difference.

3.1.4 Validation

Splitting training and validation sets is non-trivial in time-series problems.

- **Data stride.**

  Data stride matters. Originally we take a sample every 5 records and split training-validation among these samples. This data stride works quite well. Then we try to decrease data stride and performance becomes worse in public score. The reason might be that decreasing data stride increases correlation between samples.

- **Strategy for splitting train-val sets.**

  We use 80% of training data as real training data and the rest 20% as validation data.

  We explore two strategies for data splitting. One is randomized index, the other is sequential. It turns out that randomized index is more robust than sequential ones. But in our stacking ensemble that will be introduced later, we use different sequential splits.

The aim of validation is to make the model more general and robust to different inputs. But sadly, validation cannot really help us identify the robustness of a model as **public score and private score are still not strictly correlated even given the same validation score**.

Figure 3: LSTM model structure

3.2 LSTM

The structure of this model is given in **Figure 3**, where we choose a batch size of 32 and hidden dimension of 128. The number of hidden dimension should be adjusted according to the number of features for each sample.

Next we introduce a few tricks from NLP that helps with convergence when using LSTMs.

- **Orthogonal Initialization.**

  Extensive researches have proved that orthogonal initialization of weights helps with gradient vanishing problem of RNNs.

- **Using three gates instead of four.**

  To put it simply,

  $$forget\_gate = 1 - input\_gate \tag{3}$$

  By doing this, we reduce the complexity of this model.

  Further, we set the bias of $input\_gate$ to zero, meaning that the default initial behavior of the model is to **retain all information from** c_{t-1} when calculating c_t.

Here we also talk about the choice of loss function. It is known that L1 loss will produce a sparse matrix and L2 tends to lead to a more well distributed matrix. **Table 2** illustrates public scores using different loss functions for LSTM model.

We apply three different loss functions the the model of LSTM.

$L1$	MSE	$rMSE$
0.00148	0.00150	0.00149

Table 2: public score with different loss functions in LSTM model

3.3 Meta-Ensemble

The second submitted model is actually a single XGBoost. But here we would like to introduce our trials on ensemble of tree-based models and LSTM.

In this ensemble we use four tree-based learner and one LSTM learner, Random Forest Regressor, two XGBoost Regressor(DART and GBDT), and LSTM Regressor.

The hyper parameters for each base learner is decided based on intuition, trials and GridSearchCV. For details of the hyper parameters part please refer to Hao-Tian Tang's report.

3.3.1 CART CART is almost a basis for every tree-based model we use in this section.

When CART is used for regression, we generally use samples' variance on deciding to split a node.

Typically,

$$\sigma = \sqrt{\sum_{i \in I} (x_i - \mu)^2} = \sqrt{\sum_{i \in I} x_i^2 - n\mu^2} \tag{4}$$

$$Gain = \sum_{i \in I} \sigma_i \tag{5}$$

And the algorithm chooses a node to split that maximize the $Gain$.

3.3.2 Random Forest Random forest is one of our base learners used for stacking ensemble(see **section 3.3.4**)

Random Forest is a bagging algorithm. It reduces variance. Bagging is when it creates different models by resampling the data to make the resulting model more robust.

So random Forest adds additional randomness to the model. Specifically, instead of searching for the most important feature while splitting a node, it searches for the best feature among a random subset of features. This results in a wide diversity that generally results in a better model.

Therefore, in Random Forest, only a random subset of the features is taken into consideration by the algorithm for splitting a node. We can even make trees more random, by additionally using random thresholds for each feature rather than searching for the best possible thresholds (like a normal decision tree does).

Deep decision trees might suffer from overfitting. Random Forest prevents overfitting most of the time, by creating random subsets of the features and building smaller trees using these subsets. Afterwards, it combines the subtrees.

3.3.3 XGBoost This is another base learner used for stacking ensemble.

Boosting reduces variance, and also reduces bias. It reduces variance because it is using multiple models (bagging). It reduces bias by training the subsequent model by telling it what errors the previous models made (the boosting part).

Gradient boosting is an ensemble by itself. In this ensemble, the base learner must be weak. If it overfits the data, there won't be any residuals or errors for the subsequent models to build upon.

XGBoost is an effective implementation of gradient boosting algorithm. It surpasses traditional GBDT in a few aspects.

- **Add regularization to objective functions, pre-prune when optimizing objectin functions, prevent overfitting.**

$$\Omega(f_t) = \gamma T + \frac{1}{2}\lambda \sum_{j=1}^{T} w_j^2 \tag{6}$$

$$\Omega(f_t) = \gamma T + \frac{1}{2}\lambda \sum_{j=1}^{T} w_j^2 \tag{7}$$

$$Gain = \frac{G_L^2}{H_L + \lambda} + \frac{G_R^2}{H_R + \lambda} - \frac{(G_L + G_R)^2}{H_L + H_R + \lambda} - \gamma \tag{8}$$

  where l is the training loss, T is the number of leaves, the second item of Ω is the L2 norm of leaf scores.

  Introducing γ in $Gain$ pre-prunes the model. Introducing λ in $Gain$ smoothens the leaf score.

- **Column subsampling.** This originates in the algorithm of random forest. It not only prevents over-fitting but also reduces complexity.

3.3.4 Stacking Ensemble (Meta-Ensemble) Stacking (also called meta ensembling) is a model ensembling technique used to combine information from multiple predictive models to generate a new model.

Often times the stacked model will outperform each of the individual models due its smoothing nature and ability to highlight each base model where it performs best and discredit each base model where it performs poorly.

In stacking, **the training data** is split into n folds. We treat every fold as stacking-phase testing data and the rest folds as stacking-phase training data to train every model once. So, suppose we have n folds and m learners, the total number of training is $n \cdot m$.

In the perspective of the i_{th} fold as testing data, each of the m learners has their own prediction of it. So, a **shared** linear regression component is used to unify the decision of various learners into a final prediction for the i_{th} fold. In other words, linear regression is the final step for the training of stacking-phase.

Then, trained learner j will predict the whole test data and generate $Prediction_{ij}$.

Refer to **Figure 4** for an illustration.

The prediction from base learner j on the true test data is given by:

$$Prediction_j = \frac{\sum_{i=1..n} Prediction_{ij}}{n} \tag{9}$$

The last step is to apply our trained linear regression model to these predictions.

$$Prediction = LR(Prediction_1, ..Prediction_m) \tag{10}$$

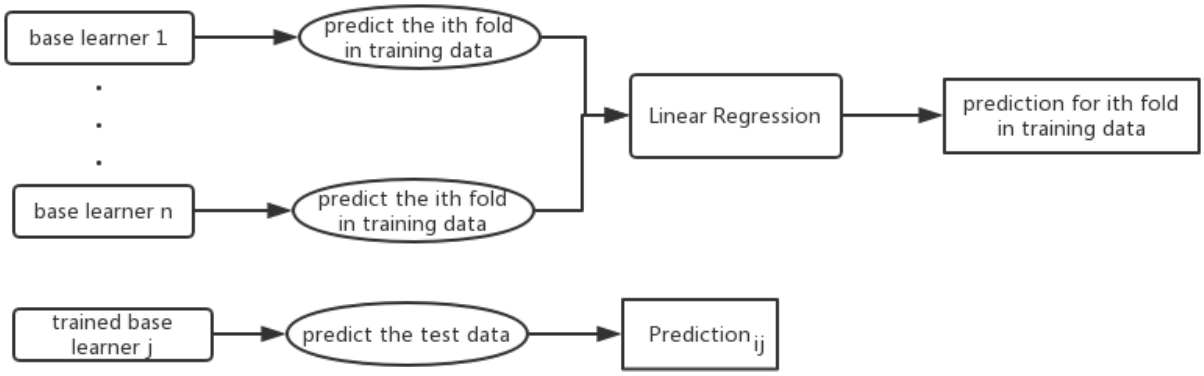

Figure 4: Stacking ensemble in the perspective of the i_{th} fold

4 Failed Attempts

4.1 Linear Regression

Linear regression has a severe problem of over-fitting. Even though after careful tuning, the model could be generalized to validation dataset, the public score does not benefit from it. Worse still, it happens that a model generalizes to the public score, but behaves poorly in private score.

4.1.1 Using historical features We tried adding the information of timestamps prior to the given 10 points, selecting several individual points or adding the mean of previous periods into the feature. Neither of them worked.

Indeed, historical features drags LR model to a public score of 0.00168 and private score of 0.00176.

4.1.2 Using only prices for regression We think the failure of linear regression may result from strongly correlated features, so we tried reducing the features to only prices.

It does helps a little, but the performance is not satisfying as well.

4.2 XGBoost Classification

Instead of directly doing regression using XGBoost, we tried first train an XGBoost classifier.

Then if the classifier predict the price to go up, we add the **standard deviation** of the 10 timestamps' midprices to the midprice of the 10_{th} timestamp. Conversely, if the classifier say it is going down, we substract the standard deviation from the midprice of the 10_{th} timstamp.

The classification accuracy we achieved is around 60%, the model doesn't really outperform the LSTM one. So we abandoned this thought.

4.3 OHEM (Online Hard Example Mining)

Online Hard Example Mining (OHEM) is a way to pick hard examples with reduced computation cost to improve your network performance on borderline cases which generalize to the general performance. It is mostly used for object detection.

OHEM performs hard example selection batch-wise. Given a batch sized K, it performs regular forward propagation and computes per instance losses. Then, it finds $M < K$ hard examples in the batch with high loss values and it only back-propagates the loss computed over the selected instances.

In our task, we define the samples where the target average mid price is **more than** 0.005 **away** from mid price of the 10_{th} timestamp as hard samples. Following this definition, there are altogether $7,000$ hard samples out of $60,000$ in the training dataset.

Using OHEM, we reduce the $easy - hard\ sample\ ratio$ from $8:1$ to $1:1$.

We use this strategy to train our LSTM model and achieved the public score of 0.00155, meaning that this attempt is not successful.

4.4 Predicting the Whole Sequence

This is an attempt almost destined to doom because there is no ground truth guiding successive predictions.

And we gave it a try, the revised model is similar to an encoder-decoder model, where the decoder part is inputless. The structure of this model is trivial so we skip the illustrations.

No miracle happens in this trial. Reasonable.

4.5 Deep Markov Model

Inspired by Liu's knowledge sharing, we began to investigate DMM.

The original algorithm given by (Rahul G. Krishnan 2017) illustrated in **Figure 5**.

In DMM, it is allowed for the transition probabilities governing the dynamics of the latent variables as well as the the emission probabilities that govern how the observations

Algorithm 1 Learning a DMM with stochastic gradient descent: We use a single sample from the recognition network during learning to evaluate expectations in the bound. We aggregate gradients across mini-batches.

Inputs: Dataset $\mathcal{D}$
Inference Model: $q_\phi(\vec{z}|\vec{x})$
Generative Model: $p_\theta(\vec{x}|\vec{z}), p_\theta(\vec{z})$
while $notConverged()$ **do**
1. Sample datapoint: $\vec{x} \sim \mathcal{D}$
2. Estimate posterior parameters (Evaluate μ_ϕ, Σ_ϕ)
3. Sample $\hat{\vec{z}} \sim q_\phi(\vec{z}|\vec{x})$
4. Estimate conditional likelihood: $p_\theta(\vec{x}|\hat{\vec{z}})$ & KL
5. Evaluate $\mathcal{L}(\vec{x}; (\theta, \phi))$
6. Estimate MC approx. to $\nabla_\theta \mathcal{L}$
7. Estimate MC approx. to $\nabla_\phi \mathcal{L}$
(Use stochastic backpropagation to move gradients with respect to q_ϕ inside expectation)
8. Update θ, ϕ using ADAM (Kingma and Ba 2015)
end while

Figure 5: Deep Markov model.

are generated by the latent dynamics to be parameterized by (non-linear) neural networks.

Specifically, the transition and emission is regarded as a gaussian distribution, where the mean and variance are given by neural-network regressions.

$$z_t \sim \mathcal{N}\left(G_\alpha\left(z_{t-1}, \Delta_t\right), S_\beta\left(z_{t-1}, \Delta_t\right)\right) \quad (Transition) \tag{11}$$

$$x_t \sim \Pi\left(F_\kappa\left(z_t\right)\right) \quad (Emission) \tag{12}$$

On the other hand, due to the intractability of $p_\theta(z|x)$, the algorithm posit an approximate posterior distribution $q_\phi(z|x)$ to obtain the following lower bound on the marginal likelihood:

$$\log p_\theta(x) \geq q_\phi(z|x) E\left[\log p_\theta(x|z)\right] - \mathrm{KL}\left(q_\phi(z|x) \| p_\theta(z)\right) \tag{13}$$

In our task, as we're facing an regression problem, an idea is to maximize $P(x_t(T)|x_1, X_2, ...x_t - 1)$, and obtain μ and σ of $P(x_t)$. Or we can simply do a regression using the hidden state series Z and observed state series X. But this method doesn't seem to work well.

4.6 Respectively build models for morning and afternoon data

The intuition is that market behaves differently in the two periods.

But this attempt does not help the model to perform better as well (public score: 0.00149). Chances are that the discrepancy is reasonably small across the two periods. Also, using two models means we have less training data for each model.

4.7 LSTM Distillation

Inspired by TA's reference to Proxy Task, we explore the effect of distillation, which is proposed by (Geoffrey Hinton 2014).

We train a LSTM classifier with can see all the timestamps in the training data, and then train another LSTM model without data penetration. The second one has two heads, one for classification and another for regression. We train the classifier head to minimize KL divergence of between its prediction and the prediction from the first LSTM, which can be seen as a proxy task. And we train its regression head to minimize the L1 loss.

5 Conclusion

To prepare for this project, we first get to learn the basics of stock trading in financial perspective.

Personally I have consulted my friends who have experiences in equity research or investment banking. According to them, there's intuitively hardly any effective solution (better than novice traders) given only an orderbook with such a time span. But indeed, during this phase I got to know a lot about the mechanism of stock market and high-frequency trading.

Our first submission (simple LSTM) achieved 0.00148 in public score and 0.00143 in private score, which is only 0.00001 worse than our best model (ensemble of LSTM and XGBoost). Our first submission is also our final submission for the competition.

Motivated by other people's $0.0012x$ in the public leaderboard. We began to try out different ways in both data preprocessing and modeling. In the past two weeks, we have learned and implemented various algorithms and strategies as explained previously. Also, we have fostered an ability to analyze data, and an instinct for data and their compatible models during trial and error, which I believe is the most importance takeaway in this project.

Acknowledgements

We thank our TAs for presenting this real-world task along with the dataset for us. Also, we would like to express our gratitude for the classmates that kindly offered their perspective and inspired us in our modeling.

References

Geoffrey Hinton, Oriol Vinyals, J. D. 2014. Distilling the Knowledge in a Neural Network.

Hochreiter, S., and Schmidhuber, J. 1997. Long short-term memory. *Neural Computation* 9(8):1735–1780.

Rahul G. Krishnan, Uri Shalit, D. S. 2017. Structured Inference Networks for Nonlinear State Space Models.

Zhuoran Xiong, Xiao-Yang Liu, S. Z. H. Y. A. W. 2019. Practical Deep Reinforcement Learning Approach for Stock Trading.