FRE 7773 – Final Project Report

Jingzhe Yu   jy2685

In this project, we applied the machine learning methods we learned in this course to deal with a real-world financial dataset, which contains the daily data for about 1500 stocks on the US stock exchange market, from Feb 2008 to July 2017.

A. Methodology

Basically, WE used two methods to handle this dataset and generate trading strategies. The "classical ML" approach we chose is Gradient Boosting Decision Tree (GBDT):

a) Gradient Boosting Decision Tree

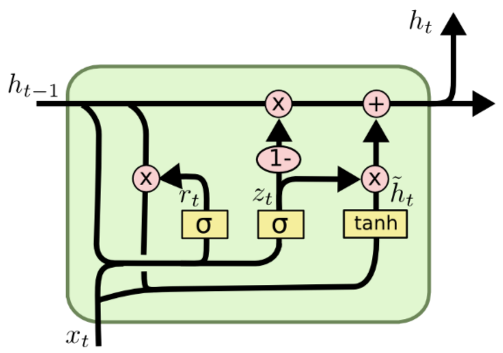
GBDT is an ensemble technique, which is constructed by a series of decision trees, and the next tree in the series is generated based on the residuals of the previous tree.

To be more specific, in the iteration of GBDT, assume the strong learner obtained by the last iteration is $f_{t-1}(\mathrm{x})$, and the loss function is $\mathrm{L}(\mathrm{y}, f_{t-1}(\mathrm{x}))$. Then the goal of this iteration is to get one weak learner $h_t(\mathrm{x})$ of the CART regression tree model makes the loss of this round $\mathrm{L}(\mathrm{y}, f_t(\mathrm{x})) = \mathrm{L}(\mathrm{y}, f_{t-1}(\mathrm{x})) + h_t(\mathrm{x})$ to be the smallest. In other words, the decision tree obtained by this iteration should make the loss of the sample as small as possible.

Nowadays, gradient boosting models are widely used in all kinds of data science competitions. They are proved to be relatively more efficient and accurate compared to other classical machine learning models. At the beginning of my project, we tried to use another gradient boosting model - LightGBM to handle my task. However, LightGBM seems to be too complex that its result overfits on the validation set a lot. Finally, we chose a vanilla GBDT model, *GradinetBoostingClassifier* from *sklearn* package to implement my classical ML approach.

b) Long-short term memory (LSTM)

LSTM is a derived model from basic Recurrent Neural Network (RNN). RNN is a kind of neural network that can capture the patterns of sequential data. Nonetheless, RNN is prone to be faced with the problem of gradient vanishing and gradient exploding. So, LSTM model is created to address this issue.



$$
\begin{aligned}
z_t &= \sigma\left(W_z \cdot [h_{t-1}, x_t]\right) \\
r_t &= \sigma\left(W_r \cdot [h_{t-1}, x_t]\right) \\
\tilde{h}_t &= \tanh\left(W \cdot [r_t * h_{t-1}, x_t]\right) \\
h_t &= (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t
\end{aligned}
$$

The structure of LSTM

Each cell of LSTM has three gates: *forget gate, input gate,* and *output gate.* These gates work together to help the LSTM model handle both long-term and short-term information. LSTM is one of the most popular deep learning techniques, especially in the field of the stock market.

In my project, to meet the requirement of LSTM input, we processed the raw data and made each sample become a 5-step sequence. The elements in this sequence are the 33 momentum factors for T-80, T-60, …T day. In other words, each sample in my new dataset has the feature dimension (5,33), and the label is original target label for day T. Why we designed features for LSTM like this is that, the 33 momentum features for adjacent days are very similar and overlapping, since each sample has its return on the previous 20 days as features. In order to make the information between each time step less overlapped, we selected one sample for LSTM input every 20-day.

Next, we built a neural network combined with an LSTM layer and a fully-connected layer. Then we used a sigmoid activation function to make the output a single value. Note that neural network models are very prone to overfit, and actually we only used momentum factors in this project, so the patterns behind data may not be so complicated. Thus, we limited the number of neurons no more than 20. The hyper-parameters would be discussed in detail later.

B. Training Procedures

a) Gradient Boosting Decision Tree

First, WE used the data from 2008 to 2012 to tune hyper-parameters. Since our goal is to use 5-year data for training and subsequent one-year data for testing, it's essential to create our validation set based on time. That's because the aim of our validation set is to evaluate the possible performance of our model on the test set. If we pick a random split, we would use future information to make predictions on the validation set, which is a big mistake. Hence, we used the data in the last half of the year 2012 as the validation set, and else as the training set.

After determining the training and validation set, we use the *hyperopt* package to tune parameters (*Hyperopt* is an efficient tuning-parameter library that allows for simple applications of Bayesian optimization). Our parameter space for tuning is

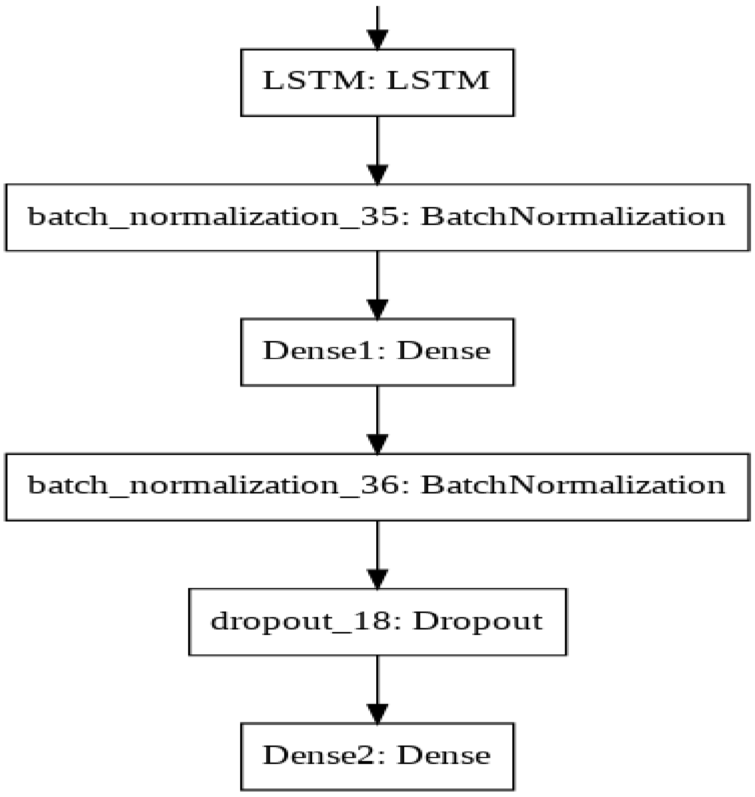
Max_depth	N_estimators	Learning_rate
[1, 2, 3]	[20, 40, 60, 80, 100]	[0.1, 0.2, 0.5, 1]

And we chose the very set of parameters that maximized the ROC-AUC on the validation set. After the tuning process, we found the set {max_depth:2, n_estimators: 100, learning_rate: 1} produced the highest ROC-AUC, which is 0.503. Then we used this set to build our GBDT model, then trained on the previous 5-year data and made predictions on each year from 2013 to 2017. The testing result would be discussed in the next part of this report.

b) LSTM

Similarly, we split the training set based on time, i.e., 2008-2011 and the first half of 2012 data for training, the second half of 2012 for validation.

In this part, we chose *Keras* as our deep learning API. *Keras* is a user-friendly deep learning framework which can be constructed easily. In order to avoid overfitting, we added batch normalization and dropout techniques for both the LSTM and Dense layer:


LSTM: LSTM
batch_normalization_35: BatchNormalization
Dense1: Dense
batch_normalization_36: BatchNormalization
dropout_18: Dropout
Dense2: Dense

We then used *hyperas* package to tune parameters for our LSTM model. There are quite a few parameters for us to tune and selected, the space listed below:

Table 1 hyper-params space for LSTM

LSTM layer		Dense layer		Other parameters	
neurons	Range (4,21,4)	neurons	Range (2,9,2)	Optimizer	[adam, rmsprop]
Activation	[relu, elu, tanh]	Activation	[relu, elu, tanh]	Batch size	Range (1000,5000,1000)
Dropout rate	Uniform (0,1)	Dropout rate	Uniform (0,1)		

And we used an early stopping technique to avoid overfitting: if the log loss on validation set doesn't improve on 5 epochs, the training process will be terminated.

Similarly, we chose the set of parameters that minimized the log loss on validation set, which is 0.6928. The selected set is listed below:

Table 2 Selected parameters for LSTM model

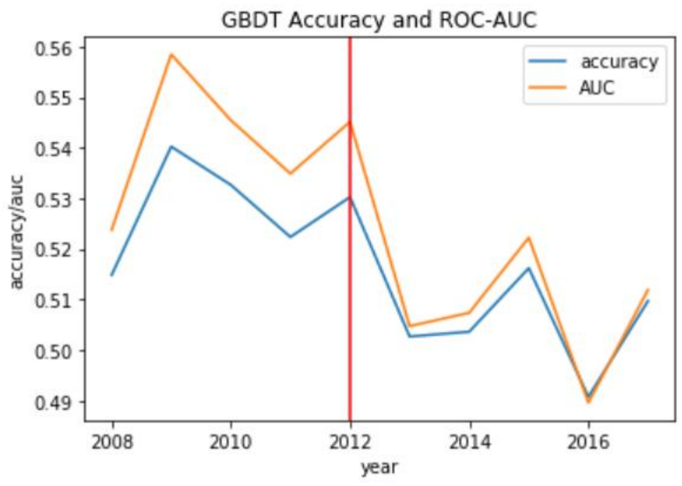
LSTM layer		Dense layer		Other parameters	
neurons	20	neurons	2	Optimizer	adam
Activation	tanh	Activation	elu	Batch size	3000
Dropout rate	0.517	Dropout rate	0.326	Output activation	sigmoid
BN	Momentum =0.9	BN	Momentum =0.9		
Kernel initializer	Random uniform	Kernel initializer	Random uniform		

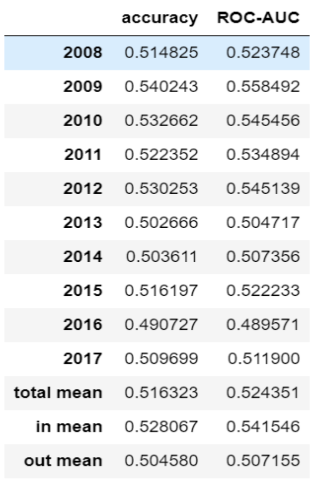
Deep learning can overfit on the validation set very quickly, and we found that the validation loss of our best model started to grow from the 4th epoch. Hence, while making predictions on the test set, we fix the epoch as 3.

C. Results

a) GBDT

The following graph and table depict in-sample and out-of-sample performance of our GBDT model:





	accuracy	ROC-AUC
2008	0.514825	0.523748
2009	0.540243	0.558492
2010	0.532662	0.545456
2011	0.522352	0.534894
2012	0.530253	0.545139
2013	0.502666	0.504717
2014	0.503611	0.507356
2015	0.516197	0.522233
2016	0.490727	0.489571
2017	0.509699	0.511900
total mean	0.516323	0.524351
in mean	0.528067	0.541546
out mean	0.504580	0.507155

We can figure out that our model performed well in 2015, but relatively worse in 2016.

Then we put our result into the back-test framework:

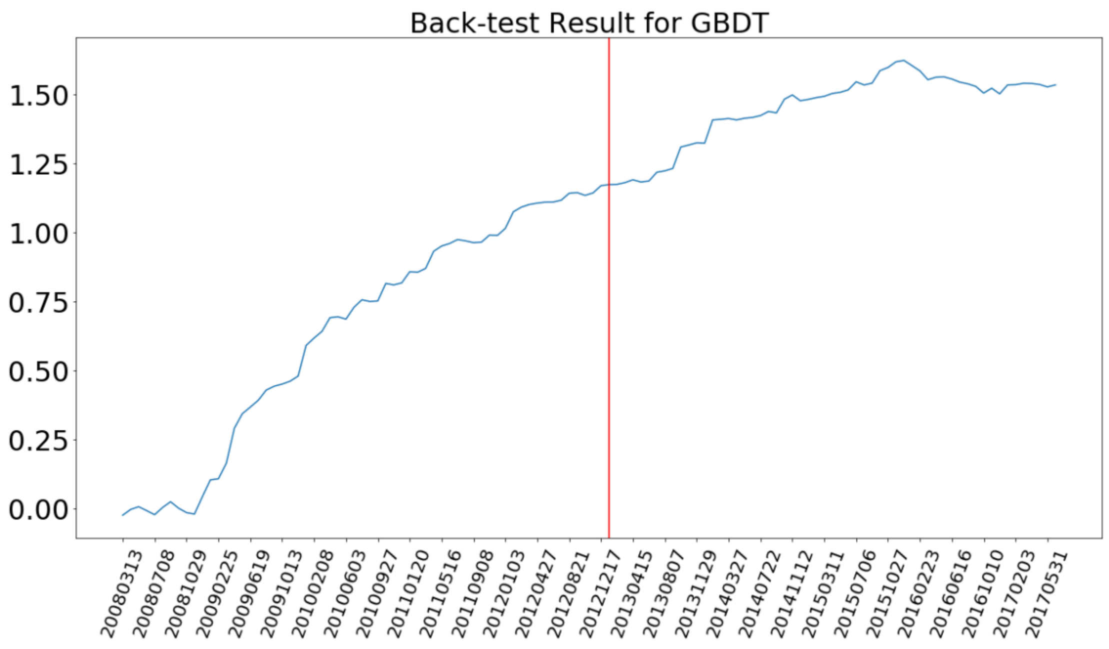


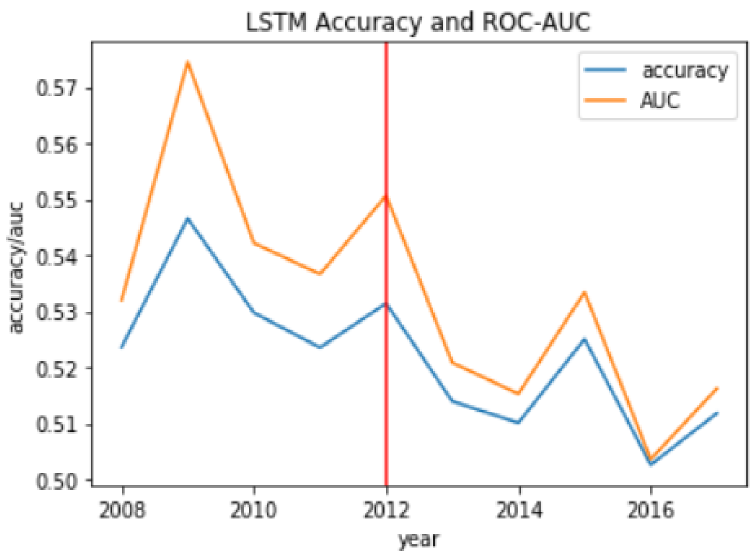
Table 3 Return and Sharp Ratio for GBDT

Year	2008	2009	2010	2011	2012	2013	2014	2015	2016	2017	total mean	in mean	out mean
Return	0.042	0.437	0.338	0.172	0.181	0.154	0.154	0.146	-0.121	0.032	0.153	0.234	0.073
Sharpe	0.641	4.847	3.469	3.220	3.613	2.498	2.093	3.715	-3.255	1.591	2.243	3.158	1.329

We can see that our model performs generally better than the baseline model, whereas it has a larger drawdown in the year 2016. Presumably, outliers have higher negative effects on more complicated models.

b) LSTM

Similarly, we present the in-sample and out-of-sample performance of our LSTM model:

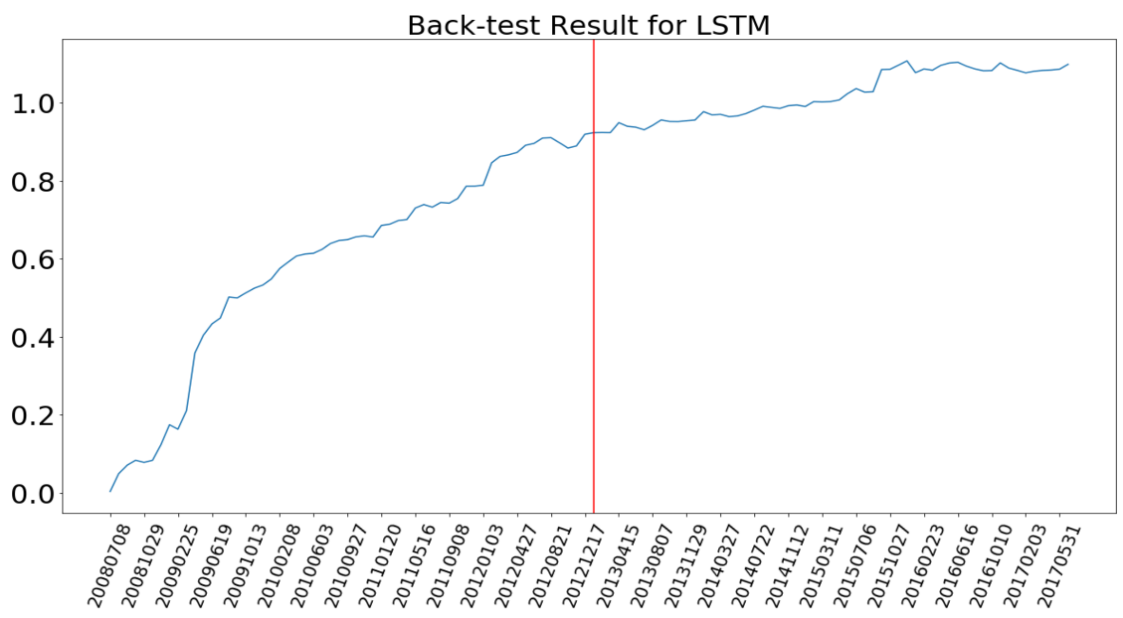


	accuracy	ROC-AUC
2008	0.523600	0.532030
2009	0.546550	0.574474
2010	0.529743	0.542174
2011	0.523516	0.536623
2012	0.531418	0.550638
2013	0.513937	0.520839
2014	0.510101	0.515266
2015	0.525038	0.533398
2016	0.502625	0.503617
2017	0.511804	0.516201
total mean	0.521833	0.532526
in mean	0.530966	0.547188
out mean	0.512701	0.517864

It seems that our LSTM model has both better accuracy and AUC than GBDT model. Anyway, the trends in performance are very similar for both models.

Furthermore, we take a look at our back-test result: (the next page)

Unfortunately, the back-test result for our LSTM model didn't perform so well out of sample. It seems that LSTM didn't have a decent capability to capture the pattern of the stocks with extreme returns, i.e., the top 10% and bottom 10% stocks.



Year	2008	2009	2010	2011	2012	2013	2014	2015	2016	2017	total mean	in mean	out mean
Return	0.123	0.409	0.123	0.131	0.133	0.037	0.039	0.113	-0.019	0.010	0.110	0.184	0.036
Sharpe	4.191	3.723	5.274	3.849	2.537	1.380	1.812	2.450	-0.535	0.996	2.568	3.915	1.221

Pros and Cons for “classical ML” vs “deep learning” models

GBDT:

Pros: It performs quite well on the test set so it may have a desirable ability to detect a single kind of pattern behind data, like the momentum pattern in this project

Cons: In this project, the speed of training GBDT models is a little slow. Its efficiency is not so good on large datasets, like computation vision or language materials.

LSTM:

Pros: It can handle very large datasets, as well as capture a complex relationship between features and target.

Cons: It’s very prone to overfit on test sets, especially on stock markets which is full of all kinds of noise.

D. Conclusion and Further Work

With this task and given this dataset, the GBDT model is proved to be better than LSTM and baseline models on the test set. GBDT has a higher average return and sharp ratio. Deep learning models are not so proper for stock data with only one kind of factor.

In order to improve our models, it’s necessary to enhance the diversity of our features, like adding value factors and growth factors into our dataset.