Final Project Machine Learning

Jiangguangyu Xue
jx1021

1. Introduction and Methodology

1.1. Random Forest

The reason I didn't choose SVM is that for training set like this large in our project, SVM is not suitable. Even if we choose linear core, it still takes a great amount of time to compute the cost function. However, Random Forest, compared with SVM, is much faster. We can set n_jbos = -1 to utilize all the processors to accelerate. But what has to be mentioned is that, it still takes me nearly half a day to do the Cross validation on two parameters with grid 3×3. Also, another advantage of Random Forest, compared with decision tree, is that it reduces overfitting problem and also reduces the variance.

I also think about the generalized Logistic Model (not just consider first order term but also take square terms into considerations, but for the reason that we are required to use the same 33 features to build up model, I gave up on Generalized Logistic model.

In Random Forest Model, I tuned three hyper parameters: n_estimators, max_depth and max_leaf_nodes. Having appropriate parameters can help us avoid over overfitting.

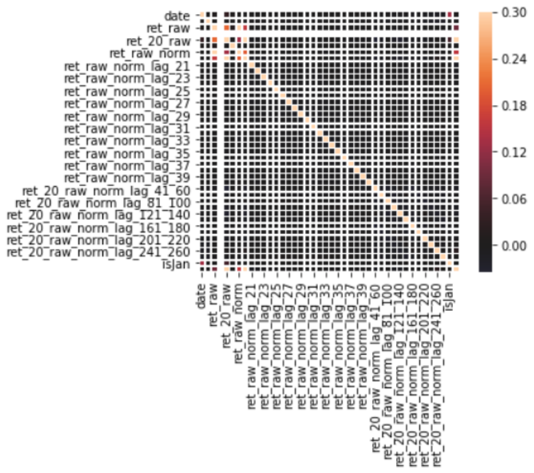
1.2 Deep Neural Network

I compare RNN and DNN here and finally go with RNN. The main reason I did not choose RNN is that our features have already reflected different historical time in the past. One of the biggest reasons that RNN is popular is that it has memory, But the way we deal with our features here actually has already taken care of the memory problem.

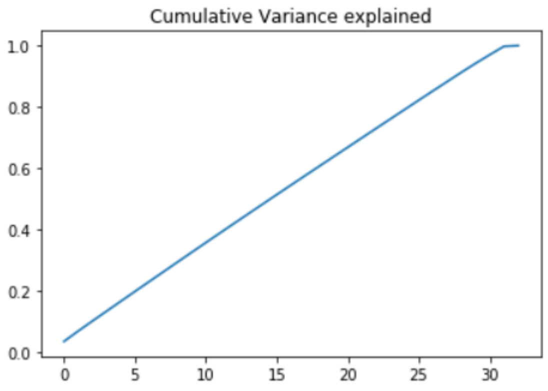
What's more, I have great flexibility here to design my own Neural Network. Dropout and choose of structure can help with the overfitting problem.

2. A brief look on data

We are given a 3547259×43 dataset. 43 columns include 33 features and 1 label. Let's first have a look at the correlations between features.



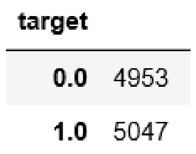
We can see from the graph above clearly that we don’t have correlations between features, which suits the assumption of linear model. I also calculate the cumulative variance explained by new features generated by PCA.



The line grows linearly, which indicates that the new features are no better than the old ones.

Therefore, I decided not to do PCA here.

Also, I want to find out if this data set is balanced dataset with about the same number of 1 and 0 in label. If this data set is unbalanced, we can consider assign different weights to the sample to rebalance it. I “Groupby” the first 10000 samples by its target. And it turns out to be balanced data set with 4953 of 0 and 5047 of 1.



target	
0.0	4953
1.0	5047

3. Training procedure

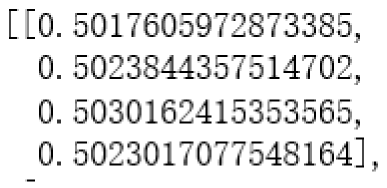
3.1. Random Forest

Before tuning the hyper parameter, we need to define the training set, validation set and test set. I used data from 2008-2-14 to 2012-6-30 as training set and data from 2012-7-1 to 2012-12-31 as validation set to test the result of my trained model. And then doing prediction of 2013, 2014, 2015, 2016 and 2017 separately with training set of previous 5 years. We do this on rolling basis to keep our model updated.

For parameters, I didn't do grid search on all the hyper parameters since it will take forever to finish and I know we should search for the optimal hyper parameter every time before we do the prediction on each test set. (find optimal parameters every time before do the prediction on 2012, 2013, 2014) However, it will consume a lot of time.

1) N_estmators

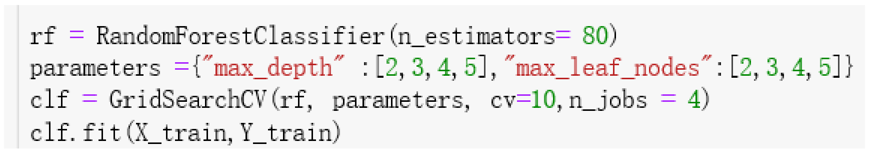
So, I tune the n_estimators first, try different value with other parameters fixed. I used 10-fold cross validation on n_estimators = [20,50,80,100]. The mean of these cross validation are shown in the table below. And as a result, I chose 80 for n_estimators.



```
[[0.5017605972873385,
  0.5023844357514702,
  0.5030162415353565,
  0.5023017077548164],
```

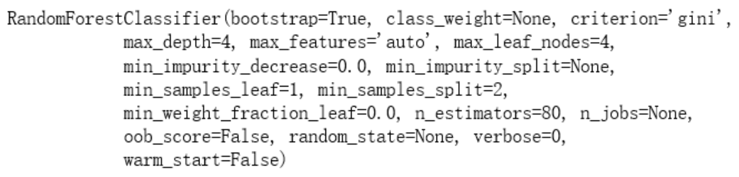
2) Max_Depth and Max Leaf Nodes:

Then, I fix n_estimators as 80 and do the grid search on max_depth and max_leaf_nodes.



```
rf = RandomForestClassifier(n_estimators= 80)
parameters ={"max_depth" :[2,3,4,5],"max_leaf_nodes":[2,3,4,5]}
clf = GridSearchCV(rf, parameters, cv=10,n_jobs = 4)
clf.fit(X_train,Y_train)
```

Output the best estimator we can find that max_depth = 4 and max_leaf_nodes = 4.



```
RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
            max_depth=4, max_features='auto', max_leaf_nodes=4,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, n_estimators=80, n_jobs=None,
            oob_score=False, random_state=None, verbose=0,
            warm_start=False)
```

These three parameters can help us avoid the overfitting problem. It can help our model not focus too much on the noise on the training set. As we all know financial data is full of noise.

And that is all about the parameter's selection in Random Forest model.

3.2 Deep Neural Network

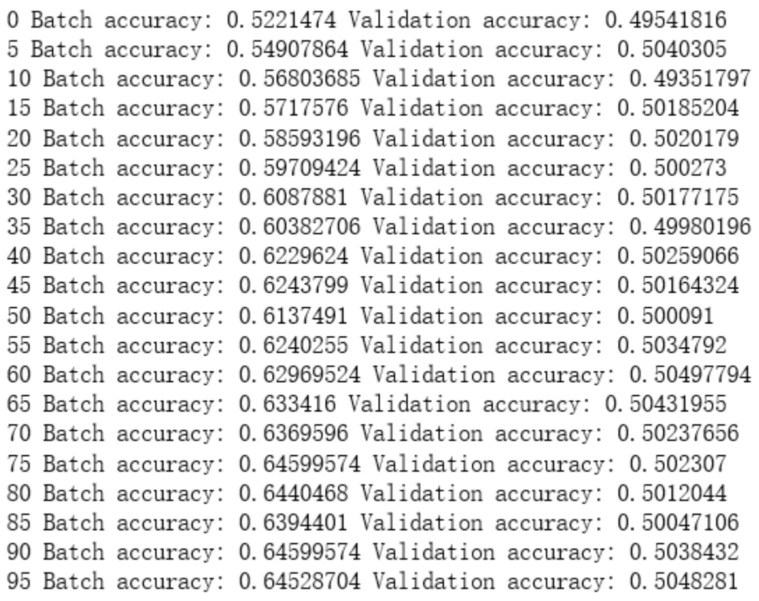
In this part, I try to determine the optimal structure as well as the hyper parameters for my DNN. I didn't use Cross validation all the time because 10-fold cross validation will cost me really a long time and my computer crushed several times when doing cross validation.

So, I try several models without cross validation at first. After some adjustments of the model, then I do the cross validation to confirm if I pick the right model with right hyper parameters.

I used data from 2008-2-14 to 2012-6-30 as training set and data from 2012-7-1 to 2012-12-31 as validation set to test the result of my trained model.

1) Model structure:

At first, I start with an MLP model: build a deep ANN with five hidden layers of 100 neurons each, He initialization, the ELU activation function, batch normalization (momentum=0.9), mini-batch gradient descent (batch_size=128), and use the ADAM optimizer. This is just like what we did in homework. However, this doesn't work well.

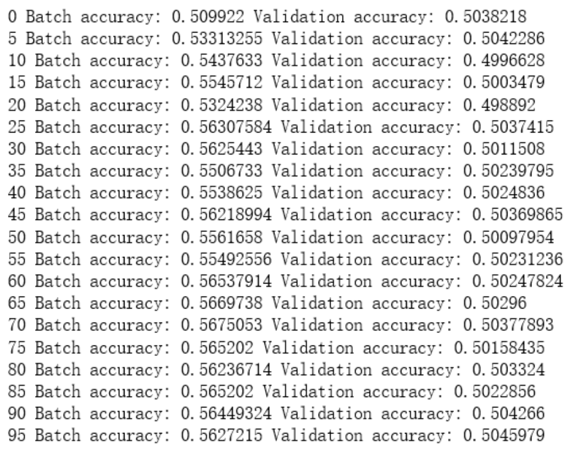


```
0 Batch accuracy: 0.5221474 Validation accuracy: 0.49541816
5 Batch accuracy: 0.54907864 Validation accuracy: 0.5040305
10 Batch accuracy: 0.56803685 Validation accuracy: 0.49351797
15 Batch accuracy: 0.5717576 Validation accuracy: 0.50185204
20 Batch accuracy: 0.58593196 Validation accuracy: 0.5020179
25 Batch accuracy: 0.59709424 Validation accuracy: 0.500273
30 Batch accuracy: 0.6087881 Validation accuracy: 0.50177175
35 Batch accuracy: 0.60382706 Validation accuracy: 0.49980196
40 Batch accuracy: 0.6229624 Validation accuracy: 0.50259066
45 Batch accuracy: 0.6243799 Validation accuracy: 0.50164324
50 Batch accuracy: 0.6137491 Validation accuracy: 0.500091
55 Batch accuracy: 0.6240255 Validation accuracy: 0.5034792
60 Batch accuracy: 0.62969524 Validation accuracy: 0.50497794
65 Batch accuracy: 0.633416 Validation accuracy: 0.50431955
70 Batch accuracy: 0.6369596 Validation accuracy: 0.50237656
75 Batch accuracy: 0.64599574 Validation accuracy: 0.502307
80 Batch accuracy: 0.6440468 Validation accuracy: 0.5012044
85 Batch accuracy: 0.6394401 Validation accuracy: 0.50047106
90 Batch accuracy: 0.64599574 Validation accuracy: 0.5038432
95 Batch accuracy: 0.64528704 Validation accuracy: 0.5048281
```

The accuracy on the training set goes up significantly, but the validation accuracy doesn't change much. I think there might be some overfitting problems in training set. Maybe just like what happens on Random Forest, this model also captures noise of stock data and focus on it.

2) Reduce Complexity of model (to prevent overfitting):

My idea here is to simplify the layers. Let's first lower down the complexity of the layers structure. I still have 5 layers but decrease the number of neurons to 50,40,30,20,10 for each layer. And the result comes out like the table below.


```
0 Batch accuracy: 0.509922 Validation accuracy: 0.5038218
5 Batch accuracy: 0.53313255 Validation accuracy: 0.5042286
10 Batch accuracy: 0.5437633 Validation accuracy: 0.4996628
15 Batch accuracy: 0.5545712 Validation accuracy: 0.5003479
20 Batch accuracy: 0.5324238 Validation accuracy: 0.498892
25 Batch accuracy: 0.56307584 Validation accuracy: 0.5037415
30 Batch accuracy: 0.5625443 Validation accuracy: 0.5011508
35 Batch accuracy: 0.5506733 Validation accuracy: 0.50239795
40 Batch accuracy: 0.5538625 Validation accuracy: 0.5024836
45 Batch accuracy: 0.56218994 Validation accuracy: 0.50369865
50 Batch accuracy: 0.5561658 Validation accuracy: 0.50097954
55 Batch accuracy: 0.55492556 Validation accuracy: 0.50231236
60 Batch accuracy: 0.56537914 Validation accuracy: 0.50247824
65 Batch accuracy: 0.5669738 Validation accuracy: 0.50296
70 Batch accuracy: 0.5675053 Validation accuracy: 0.50377893
75 Batch accuracy: 0.565202 Validation accuracy: 0.50158435
80 Batch accuracy: 0.56236714 Validation accuracy: 0.503324
85 Batch accuracy: 0.565202 Validation accuracy: 0.5022856
90 Batch accuracy: 0.56449324 Validation accuracy: 0.504266
95 Batch accuracy: 0.5627215 Validation accuracy: 0.5045979
```

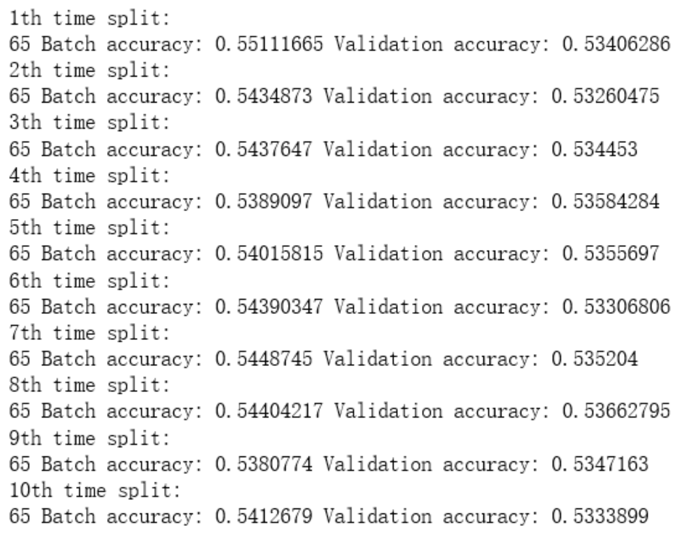
3) Reduce overfitting with dropout

I still can't get satisfied with this result, since there is still overfitting problem. Therefore, I reduce the number of hidden layers from 5 to 3. Also, I want to prevent overfitting by using Dropout. I add a dropout layer for each layer including input and all the hidden layers.

4) For epochs

From table from last model, we can see, it converges at 60 epochs. However, I am not going to set epoch as 60 but a larger number because dropout is added and the epoch needed to converges might increase. So, I use epoch = 70 here.

Then, I used cross validation here, to find out whether the accuracy on validation set is good and stable for different training set and validation set. In other words, I just repeat last model 10 times with the help of train-test split function.

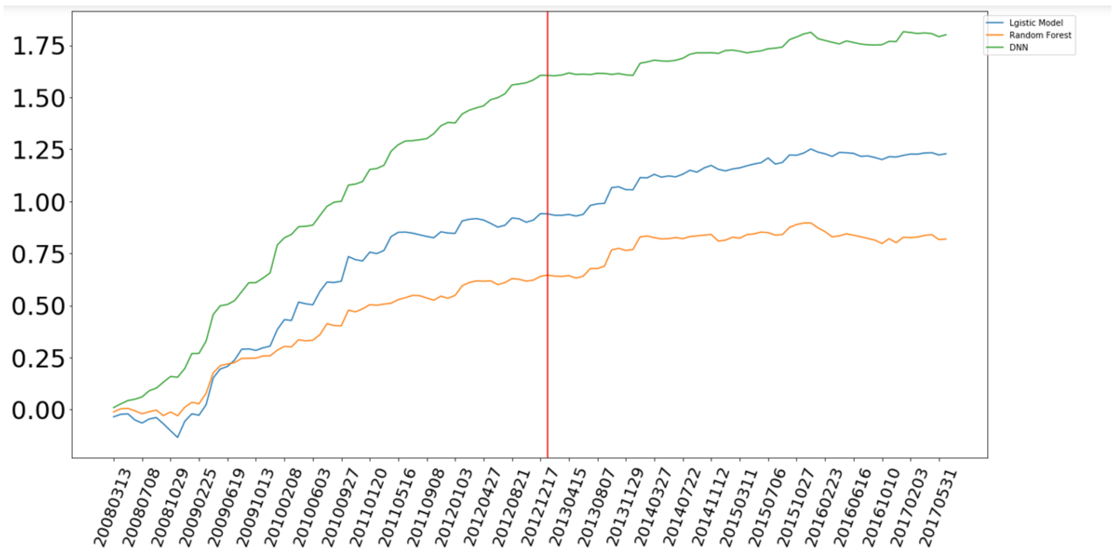

```
1th time split:
65 Batch accuracy: 0.55111665 Validation accuracy: 0.53406286
2th time split:
65 Batch accuracy: 0.5434873 Validation accuracy: 0.53260475
3th time split:
65 Batch accuracy: 0.5437647 Validation accuracy: 0.534453
4th time split:
65 Batch accuracy: 0.5389097 Validation accuracy: 0.53584284
5th time split:
65 Batch accuracy: 0.54015815 Validation accuracy: 0.5355697
6th time split:
65 Batch accuracy: 0.54390347 Validation accuracy: 0.53306806
7th time split:
65 Batch accuracy: 0.5448745 Validation accuracy: 0.535204
8th time split:
65 Batch accuracy: 0.54404217 Validation accuracy: 0.53662795
9th time split:
65 Batch accuracy: 0.5380774 Validation accuracy: 0.5347163
10th time split:
65 Batch accuracy: 0.5412679 Validation accuracy: 0.5333899
```

This table confuses me actually. I did not expect the accuracy on the validation set to be this high actually. I don't have enough time to go further on this. This can be added to further work needed to be done in the future.

5. Results and Conclusions

To predict negative or positive stock prices changes for 5 years, from 2013 to 2017. I separate these 5 years into 5 1-year test set and use previous 5 years data as training set.

Finally, we have results for three models shown in the graph below.



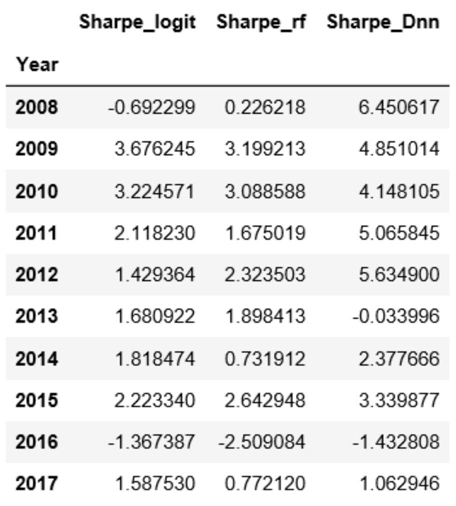
The red line divides 2012 and 2013. We can see that on training set (what is on the left-hand side of the red line).

1) Analysis of performance on training set:

DNN outperforms the other two strategies, but maybe there is a bit overfitting problem (I didn't use forward-biased data and just follow the back-test framework). Random Forest kind of underfits and worse than the two other strategies.

For the performance on the training set, Random Forest is more stable and even, while Logistic Model suffers loss on the first year and has more volatile Sharpe Ratio. Dnn is overfitting and it will be crazy, if we have Sharpe ratio over 4 or 5 in our real life.

2) Analysis of performance on test set:

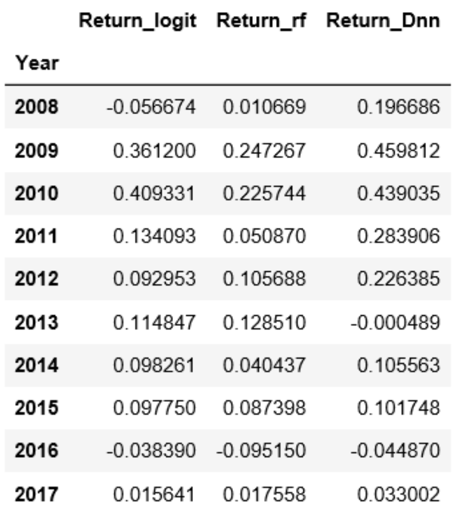


	Sharpe_logit	Sharpe_rf	Sharpe_Dnn
Year			
2008	-0.692299	0.226218	6.450617
2009	3.676245	3.199213	4.851014
2010	3.224571	3.088588	4.148105
2011	2.118230	1.675019	5.065845
2012	1.429364	2.323503	5.634900
2013	1.680922	1.898413	-0.033996
2014	1.818474	0.731912	2.377666
2015	2.223340	2.642948	3.339877
2016	-1.367387	-2.509084	-1.432808
2017	1.587530	0.772120	1.062946

Looking on the Sharpe ratio on the table above, we can find that Sharpe ratio of Random Forest on training set and test set are similar to each other. For DNN and Logistic Model, the Sharpe Ratio on the training set is larger than that on the test set. To some degree, these two models are overfitting and DNN has a more serious problem in overfitting.

However, although DNN overfits on the training set, but in my opinion, beats the other two strategies on training set. But the advantage of DNN on testing set is not that large. The other two strategies are also not bad.

Table for return is similar to Sharpe ratio.



	Return_logit	Return_rf	Return_Dnn
Year			
2008	-0.056674	0.010669	0.196686
2009	0.361200	0.247267	0.459812
2010	0.409331	0.225744	0.439035
2011	0.134093	0.050870	0.283906
2012	0.092953	0.105688	0.226385
2013	0.114847	0.128510	-0.000489
2014	0.098261	0.040437	0.105563
2015	0.097750	0.087398	0.101748
2016	-0.038390	-0.095150	-0.044870
2017	0.015641	0.017558	0.033002

3) Further Discussion

We can do more about this project. Maybe aggregate these three models to acquire more stable PnL and Sharpe ratio. We can take a weighted sum of these three models or just do the hard voting.

There is still more work on cross validation needed to be done. It can take a great amount of time to do the cross validation. But it is necessary and beneficial for our models. The hyper parameter in my model might not be optimal because I did not test enough number of parameters or sometimes did not do grid search on different parameter pair or build up 3- or 4-dimensions grid search.

6. Reference

1. Takeuchi, L., & Lee, Y. Y. A. (2013). Applying deep learning to enhance momentumtrading strategies in stocks. InTechnical Report. Stanford University.
2. Jegadeesh, N., & Titman, S. (1993). Returns to buying winners and selling losers:Implications for stock market efficiency.The Journal of finance,48(1), 65-91.