CS7641 ML- Project 1 report

Chenxi Yu

Sunday, Feb 5, 2017

A. Vanka(sh000002) stock price movement direction prediction

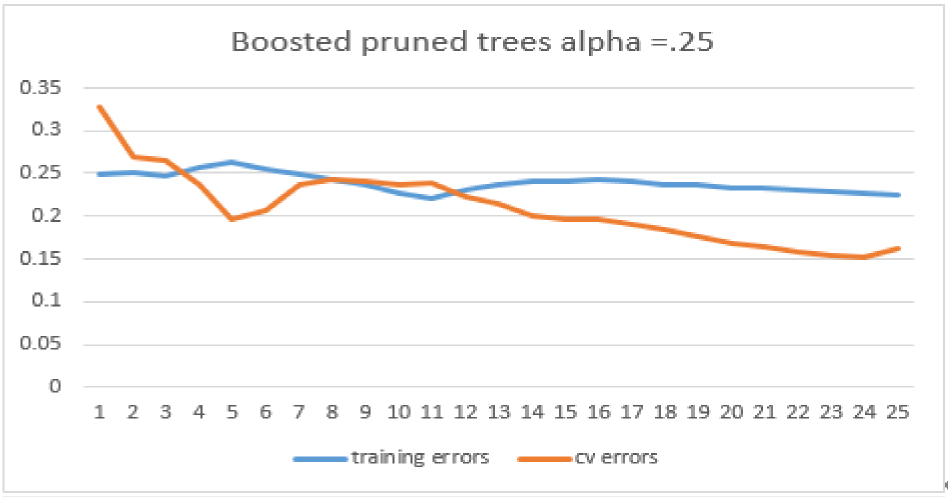
Recently，I was experimenting on applying ML methods for technical trading signals indicators selection. Thanks to Quantopians Talib library, I have used it to calculate all the technical indicators (x variables in our model)as follows: mfi(Money Flow Index), cci(Commodity Channel Index), cmo(Chande Momentum Oscillator), aroonosc(Aroon Oscillator), adx(Average Directional Movement Index), rsi(Relative Strength Index), bop(Balance Of Power), ad(Chaikin A/D Line), obv(On Balance Volume), atr(Average True Range), slowk(stoch's), slowd(stoch's), macd_result(result from macd calculations), based on the Vanka(sh000002) stock minutes frequency trading data in China. All of them are in continuous numeric values while the y-variable is the stock price moving prediction on the next period (which is in minute-base in trading time). The movement classes was defined by 3 possibilities: up (at least by .05% increase), even (small than .05% increase and decrease), down (at least by .05% decrease), all of which are based on the historical price data's standard deviation in 2013 and 2014. Meanwhile the data in this period is our training data set and the price data calculated in the same way as the above from Jan 2015 to Sep 2015 our testing data set. Additionally, the threshold for movement classes in the testing data excludes data from 2015 to avoid data snooping errors.

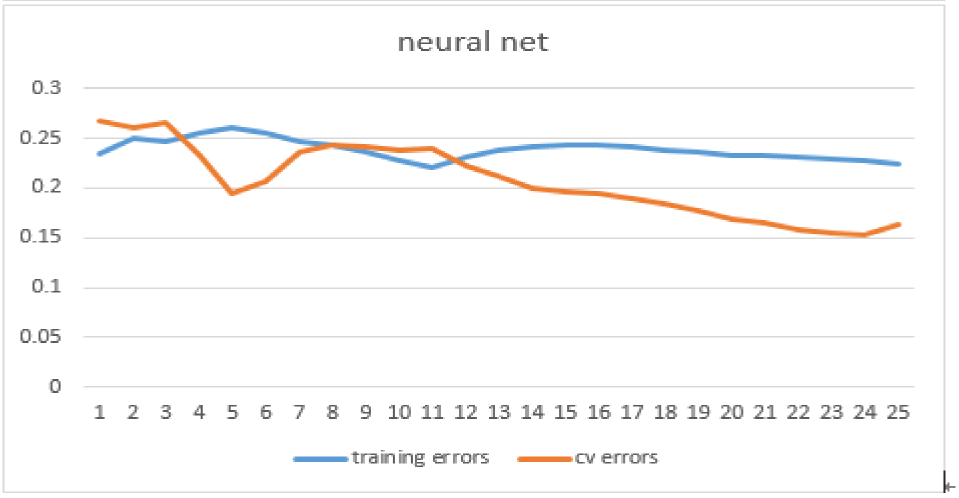
Why did I choose this data set? As a quantitative researcher in China, I have always been interested in applying ML methods to trading model constructions. Technical indicators have long been used in trading to identify profit opportunities. Proper combinations and quantifications on them will bring in successes in my team. However, as a classification in this project, it brings in challenges. First, as time series data sets, they need extra care while sampling and testing. Since the data is highly time sensitive, each movement in the time series order has information compared with non-time-series data. I should not do random sampling and testing. Second, though they are 30 m-bytes in size, it is hard to train and test them under complex models. As my practice in neural net tells, it may takes hours to run tests if iterations amount in each test is big. Third, there is not many resources in reference for this kind of classification problems online. However, it is interesting to explore this data set and horn my ML skills on top of it.

There are 5 main ML training methods in our classes so far: decision trees, neural network, boosting, KNN and SVM. There are 2 stages in exploring the performance of each algorithm: training and testing.

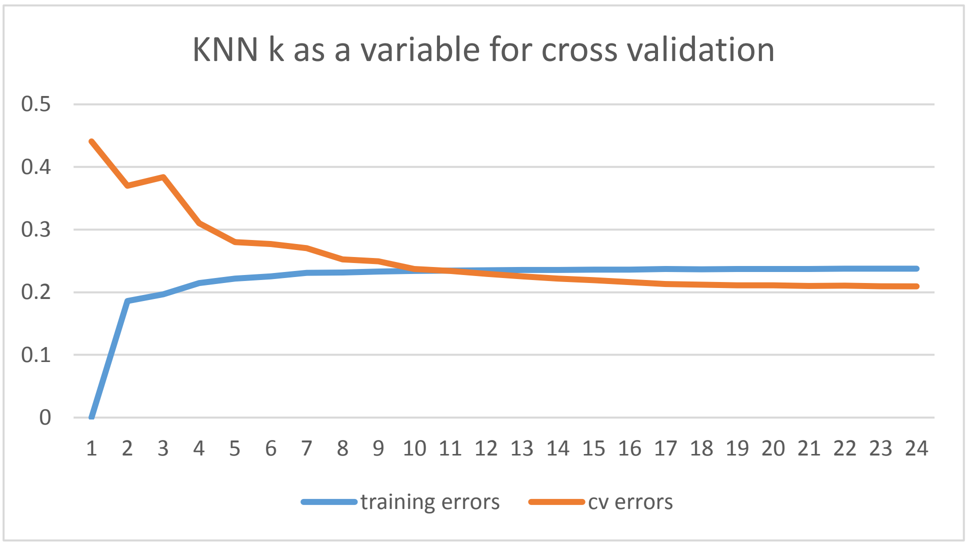
In training, I train each algorithm via cross validation based on a distribution of the data set into 25 folds, each fold is sized up by a new chunk size (2228 rows) based on the previous fold in both the training set and the validation set. However, the training set and the validation set are not mixed in each fold. This way would eliminate data snooping bias in assessing errors yet allow time varying bias due to market condition shift. As a result, most of them do not look identical to classically perfect-built models learning curves, as the sample are not randomly drawn from the population. Instead, it is drawn in the time varying natural order.

Exhibited in the document（data\result\,learning curves via data size iteration.docx）, we closely examine learning curves for each algorithm. The pure decision tree's shows the training errors remained roughly the same while the validation errors fluctuates over time yield remaining around .4 and .5, while in the boosted pruned trees with alpha = .25, we could see they both trend downward though validation errors become smaller than the training errors in the end. This indicates boosted pruned trees has improved decision trees predictive power, as a pure decision tree tend to overfit the model. Though higher training errors overall is the stake of the boosted and pruned trees model and its training errors have not decreased as data size increases, the validation errors was reduced along thanks to boosting and pruning.





Incidentally, the learning curves from the neural net looks pretty much the same as the boosted pruned trees. It also exhibits better predictive power. However, training neural network consumes more time in training and hard to interpret especially for my data set. Interestingly, I also observe that the neural net requires a larger amount of data to quickly decrease its validation errors which is not the case for the later based on the first 4 folds of data training.



Meanwhile, the knn learning curves appears a bit different from the former 2 algorithms. As shown, knn k=10 is picked as a representative for its category (the reason will be explained in the later context). However, as the data size increases, its training errors is around .24 stably, while its validation error trend downward obviously. This curves pattern also indicates sound predictive power as the last 2 algorithms.

With regard to the SVM learning curves, there are 2 models based on kernels. The Radial Basis Function (RBF) kernel works the best for the model in the learning curves. Its training error is zero while its validation errors keep converging to somewhere close to 0. Basically, it is intrinsically consistent with our data set as it is hard to fit a linear pattern in the data set. The squared exponential kernel is a great tool with the amount of data we have. However, this mix performs much slower than the other algorithms. When it comes to the sigmoid kernel, its training error curve increases while the validation error curve shoots high in some data points and increases overall. Thus, this possibly indicates a under fitting model as data size gets larger.

Next, we move onto hyper-parameters tuning. Since the learning curve based on data size increment may bear data sampling bias (natural order drawing), now I proceed cross validation via deploying test iterations in a moving window pattern where allocates 75% of its data for training and 25% for validation. Within the window, both data chunks do not share common data sample.

First, I examine the effect of pruning for a single decision tree. In the codes, I control pruning via alpha's value. When it high ultralow, it stops pruning. Without pruning, the training error curve trend lower by 16th iteration. However, the validation errors are high. This curves pair indicates overfitting bias in this model. Noticeably, it seems normal to see some uptrend in the validation errors curve due to market condition shifts. When alpha is tuned 0, 0.1 and 0.25, we find the validation errors decrease overall and the overfitting bias is reduced. Similarly, I examine the effect of pruning for boosted trees. As alpha is tuned up, the validation errors run down and curves come closer as exhibited.

Second, how do I pick the best k parameter for the KNN algorithm? Since KNN is another overfitting prone approach, I examine each k via iterations as data enlarges in the previous discussed cross validation approach(reference to data/result/learning_curves_for_knn_tuning.docx) . The curves cross over at k= 10, after which the validation curve runs down slowly and the training curve is stably approaching .25, which is similar to the low bias and low variance classic case. This urges me to investigate the k parameter value by 10, 5, 4 and 3, all of which indicates k=10 is the best we get. As the validation errors are lower while both errors curve tends downward in general. The validation errors running below the training errors implies that overfitting is handled the best among them.

Finally, at the test stage, I pick the following models:

knn(n_neighbors=10)

svm.SVC(kernel='rbf', gamma=0.001, C=1)

adaboostclf_dt(dtclf_pruned(criterion='entropy',alpha=.25),n_estimators=10,learning_rate=1) mlp_clf = MLP_clf(hidden_layer_sizes=1000, activation='logistic', solver='sgd')

Why do I pick the above 4? I have explained why for knn(n_neighbors=10), adaboostclf_dt(dtclf_pruned(criterion='entropy',alpha=.25),n_estimators=10,learning_rate=1), one more thing added to why I pick entropy, this is because every split we make we want it help maximize the information gain. For SVM and MLP_clf cases, I have experimenting them earlier though I did not have a chance to record them down for this report since implementing one takes long time. Yet, the above parameters could be intuitively presumed to be the best it can get.

```
avg / total        precision  recall  f1-score  support
knn-10             0.41       0.48     0.41      43879
boost_clf          0.54       0.55     0.52      43879
mlp(neural net)    0.25       0.50     0.33      43879
svm                0.25       0.50     0.33      43879
```

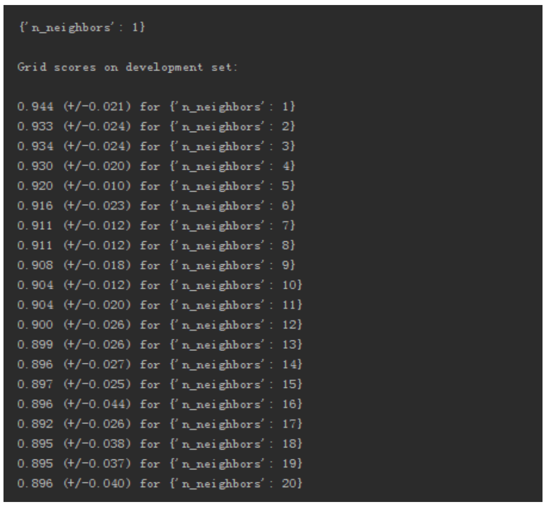
Based on the above classification report, svm and mlp get the worst result. However, all the algorithms except SVM perform worse than expected, compared with cross validation errors result. This could be explained by the market's dramatic change in 2015. For more information, please feel free to visit the graph presented for the classification report. All algorithms seem achieving a high f1 score in the '0' performance prediction, however both mlp and knn-10 has a high recall and low precision rate which means it return many results related to the training data's while most results are wrong in the confusion matrix.

The confusion matrix also indicate during the 2015 time period, for predicting a '0'label result, adaboost_dt, performs the best due to a high avg-f1 score, f1 score in the '0' label and how good both precision and recall values show as a mix.

B. Image Segmentation classification for every pixel

This data set is from the UCI Machine Learning repository. The instances were drawn randomly from a database of 7 outdoor images. The images were hand segmented to create a classification for every pixel. There are 210 instances for the training sample, and 2100 instances for the test set. Why do I like this data set? This data set is not too big, which could be easily manage as a simple example to explore classification algorithms. As a classic data set sample, there are many reference and studies based on it. So being familiar with it will help understand some open source tools. Whenever I learn new tools or new techniques, I could initiate my analyses on it, instead of exploring a new data set without enough knowledge on its feasibility and limitation. I will start off my analysis by exploring the best hyper-parameters for each model. Since we have 4 models: boosted pruned decision trees, neural network, KNN and SVM. We ought to tune hyper-parameters for all 4. Yet, due to time constraints, I have got my results except boosted pruned decision trees as follows.

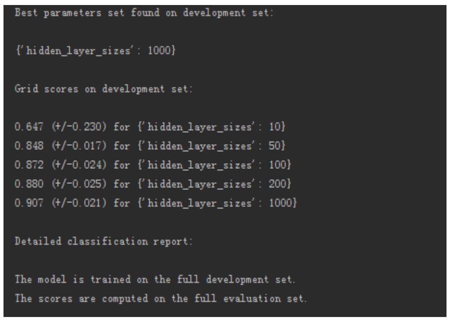
Below is the grid search result for n_neighbors hyper parameters tuning



```
{'n_neighbors': 1}

Grid scores on development set:

0.944 (+/-0.021) for {'n_neighbors': 1}
0.933 (+/-0.024) for {'n_neighbors': 2}
0.934 (+/-0.024) for {'n_neighbors': 3}
0.930 (+/-0.020) for {'n_neighbors': 4}
0.920 (+/-0.010) for {'n_neighbors': 5}
0.916 (+/-0.023) for {'n_neighbors': 6}
0.911 (+/-0.012) for {'n_neighbors': 7}
0.911 (+/-0.012) for {'n_neighbors': 8}
0.908 (+/-0.018) for {'n_neighbors': 9}
0.904 (+/-0.012) for {'n_neighbors': 10}
0.904 (+/-0.020) for {'n_neighbors': 11}
0.900 (+/-0.026) for {'n_neighbors': 12}
0.899 (+/-0.026) for {'n_neighbors': 13}
0.896 (+/-0.027) for {'n_neighbors': 14}
0.897 (+/-0.025) for {'n_neighbors': 15}
0.896 (+/-0.044) for {'n_neighbors': 16}
0.892 (+/-0.026) for {'n_neighbors': 17}
0.895 (+/-0.038) for {'n_neighbors': 18}
0.895 (+/-0.037) for {'n_neighbors': 19}
0.896 (+/-0.040) for {'n_neighbors': 20}
```

Via observation, we find k= 5 is the best parameters for the exploration in the learning curves as for its scores and standard deviation.

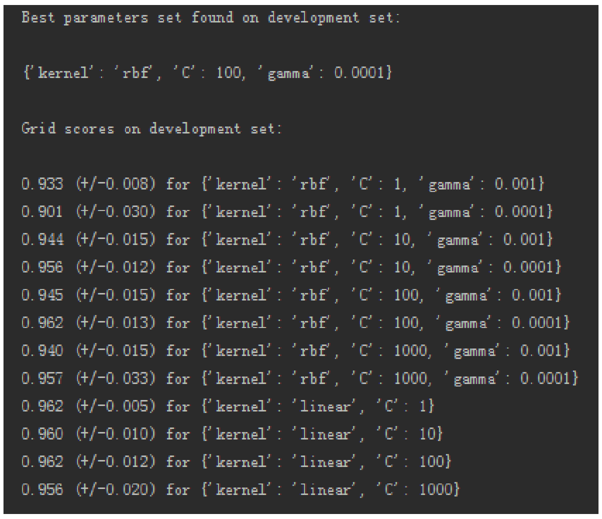
Below is the grid search result for neural net hyper parameters tuning



```
Best parameters set found on development set:

{'hidden_layer_sizes': 1000}

Grid scores on development set:

0.647 (+/-0.230) for {'hidden_layer_sizes': 10}
0.848 (+/-0.017) for {'hidden_layer_sizes': 50}
0.872 (+/-0.024) for {'hidden_layer_sizes': 100}
0.880 (+/-0.025) for {'hidden_layer_sizes': 200}
0.907 (+/-0.021) for {'hidden_layer_sizes': 1000}

Detailed classification report:

The model is trained on the full development set.
The scores are computed on the full evaluation set.
```

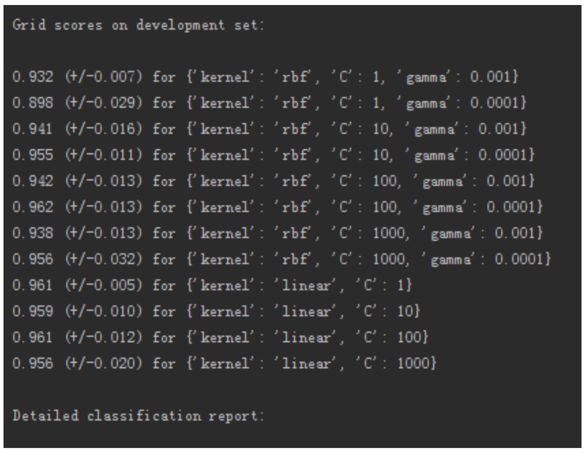
Via observation, the one with hidden layer size by 1000 is the best as it has the best score available with relatively low standard deviation.

Below is the grid search result for SVM hyper parameters tuning



```
Best parameters set found on development set:

{'kernel': 'rbf', 'C': 100, 'gamma': 0.0001}

Grid scores on development set:

0.933 (+/-0.008) for {'kernel': 'rbf', 'C': 1, 'gamma': 0.001}
0.901 (+/-0.030) for {'kernel': 'rbf', 'C': 1, 'gamma': 0.0001}
0.944 (+/-0.015) for {'kernel': 'rbf', 'C': 10, 'gamma': 0.001}
0.956 (+/-0.012) for {'kernel': 'rbf', 'C': 10, 'gamma': 0.0001}
0.945 (+/-0.015) for {'kernel': 'rbf', 'C': 100, 'gamma': 0.001}
0.962 (+/-0.013) for {'kernel': 'rbf', 'C': 100, 'gamma': 0.0001}
0.940 (+/-0.015) for {'kernel': 'rbf', 'C': 1000, 'gamma': 0.001}
0.957 (+/-0.033) for {'kernel': 'rbf', 'C': 1000, 'gamma': 0.0001}
0.962 (+/-0.005) for {'kernel': 'linear', 'C': 1}
0.960 (+/-0.010) for {'kernel': 'linear', 'C': 10}
0.962 (+/-0.012) for {'kernel': 'linear', 'C': 100}
0.956 (+/-0.020) for {'kernel': 'linear', 'C': 1000}
```

Via observation, the one with svm(kernel = 'rbf' ,C = 100 and gamma =.0001) and svm(kernel = 'linear' ,C =1)are the best as it has the best score available with relatively low standard deviation.



```
Grid scores on development set:

0.932 (+/-0.007) for {'kernel': 'rbf', 'C': 1, 'gamma': 0.001}
0.898 (+/-0.029) for {'kernel': 'rbf', 'C': 1, 'gamma': 0.0001}
0.941 (+/-0.016) for {'kernel': 'rbf', 'C': 10, 'gamma': 0.001}
0.955 (+/-0.011) for {'kernel': 'rbf', 'C': 10, 'gamma': 0.0001}
0.942 (+/-0.013) for {'kernel': 'rbf', 'C': 100, 'gamma': 0.001}
0.962 (+/-0.013) for {'kernel': 'rbf', 'C': 100, 'gamma': 0.0001}
0.938 (+/-0.013) for {'kernel': 'rbf', 'C': 1000, 'gamma': 0.001}
0.956 (+/-0.032) for {'kernel': 'rbf', 'C': 1000, 'gamma': 0.0001}
0.961 (+/-0.005) for {'kernel': 'linear', 'C': 1}
0.959 (+/-0.010) for {'kernel': 'linear', 'C': 10}
0.961 (+/-0.012) for {'kernel': 'linear', 'C': 100}
0.956 (+/-0.020) for {'kernel': 'linear', 'C': 1000}

Detailed classification report:
```

The one with svm(kernel = 'rbf' ,C = 100 and gamma =.0001) and svm(kernel = 'linear' ,C =1) are the best among all.

Overall, for the svms, I will stick with svm(kernel = ‘rbf’ ,C = 100 and gamma =.0001) as the best.

Due to a technical issue, I default hyper parameters for the boosted pruned trees model by criterion = ‘entropy’ and alpha = .25.

Next, I will investigate into the learning curves for the above (See images in the result folder by *_learning_curves format). Again, both boosted pruned decision trees and neural network have the same pattern in graph except the variation’s difference like what happened before in the stock price movement classification problem. It is likely that they have similar biases and complexity pattern with similar fitness extent and predictive power. Yet, it is noticeable that they are under-fitted as the training examples get larger in the second half of the graph since they both have decreasing accuracy scores. Meanwhile, the KNN model exhibits a paralleled increasing trend in the learning curves. This looks too promising and tends to over-fit the data.

Finally, I further dive into the test stage (the result is in the result folder). Due to time constraint, I just analyze the mlp and adaboostclf_dt cases. In mlp, the predictive accuracy seem pretty high by looking at the confusion matrix, the worst result lays in the Class_windows category yet not bad at all in value. The precision recall curve (area=.99) and ROC curve(area =1) curve perform pretty nice. The confusion matrix in the adaboostclf_dt case perform less ideal than the mlp model since Class Cement, Class Foliage and Class window have great prediction errors. The precision recall curve (area=.84) and ROC(area = .97) curve perform worse, too.

Overall, the whole classification project indicates mlp (neural net) as the best model among the models. Though it may be difficult to interpret, it has yielded the best result in identifying the classes under label training.