Predicting Stock Market

Capstone Project, Machine Learning Engineer Nanodegree

Abhinav Singh

Definition

Project Overview

Investment firms, hedge funds and even individuals have been using financial models to better understand market behavior and make profitable investments and trades. Predicting the stock price trend by interpreting the seemly chaotic market data has always been an attractive topic to both investors and researchers. Among those popular methods that have been employed, Machine Learning techniques are very popular due to the capacity of identifying stock trend from massive amounts of data that capture the underlying stock price dynamics.

This project builds a stock price trend estimate tool for a given query date. The estimator leverages state-of-art machine learning techniques, and learns from large amounts of a wide variety of historic stock market data, which is not able to be achieved by human. It is going to provide a valuable prospective to predict stocks and assist stock investments. Further, it plays an essential role for constructing an automatic trading system.
Fortunately, stock market data can be easily accessed from a number of APIs or websites on the Internet. For instance, Google Finance, Bloomberg or Yahoo! Finance interface historic stock data containing multiple metrics, such as:

- Open: price of the stock at the opening of the trading.
- High: highest price of the stock during the trading day.
- Low: lowest price of the stock during the trading day.
- Volume: amount of stocks traded during the day.

World major stock indices are important factors in affecting the future trend of stocks. Globalization has deepened the interaction between financial markets around the world. Shock wave of US financial crisis (from Lehman Brothers crack) hit the economy of almost every country and debt crisis originated in Greece brought down all major stock indices. Nowadays, no financial market is isolated. Economic data, political perturbation and any other overseas affairs could cause dramatic fluctuation in domestic markets. Important market indices can be pulled down from Internet such as:

- NASDAQ Composite (^IXIC Yahoo Finance)
- Dow Jones Industrial Average (^DJI Yahoo Finance)
- Standard and Poor's 500 (^GSPC Yahoo Finance)
- Frankfurt DAX (^GDAXI Yahoo Finance)
- Paris CAC 40 (^FCHI Yahoo Finance)
- Tokyo Nikkei-225 (^N225 Yahoo Finance)
- Hong Kong Hang Seng (^HSI Yahoo Finance)
- Australia ASX-200 (^AXJO Yahoo Finance)

Problem Statement

Specifically, estimating the stock market in this project is defined as:

Input: daily trading data over a certain date range.
Query: the date and the company symbol of which the estimator is required to predict the stock trend.
Output: estimate of stock adjusted close price trend with confidence value (rise or fall).

Methodology: supervised learning.

The output is defined to be binary (rise or fall) instead of predicting the exact stock price. The reason is that stock price is a complex statistical problem and predicting price rise or fall simplifies the problem model from regression to binary-output classification. The rise or fall trend prediction can be more accurate than predicting prices and can provide more trustable and valuable prediction result and indeed provide a practical investment suggestion in reality. On the other hand, for regression, rise/fall prediction accuracy and price prediction accuracy are both important. The matric to measure regression accuracy only considers the squared error of prediction and actual truth and not consider the direction of the prediction (rise or fall). The metrics to measure rise/fall prediction is intuitive and simple. Therefore the output is defined to be binary (rise or fall) and the problem lies in binary-class classification.

To build an effective tool for stock market prediction, the following problems are needed to be solved:

Stock data acquisition and selection
There are numerous data on the Internet. It is significant to select important datasets that affect the prediction most and get cleaned and formatted data so that program is able to process.

Feature generation
Features dimension can easily reach the scale of thousands. Generating a limited number of features that successfully capture most information and stock dynamics to feed the machine-learning model is the key of this project.

Classification model training and comparison
Metrics should be selected to effectively compare performance of different algorithms. Appropriate parameters should be searched to get best training result and prevent over-fitting.

User interface design
It is important to land the design and make it handy for users and provide valuable investment suggestions for them.

Metrics

Since the stock trend prediction problem is addressed by classification approach, F1 score is better than accuracy because accuracy does not indicate balanced precision and recall rate. However F1 score is not necessarily the best metric. Here I will propose a new metric FA score. The reasons are explained as below:

For example the test data set has 286 days. The truth is that the stock has 85 days rises and 201 days falls.
A model predicts 10 days rise correctly, which means there are 75 days that are indeed rise are predicted to fall.
The model predicts 188 days fall correctly, which means there are 13 days that are indeed fall are predicted to rise.

The confusion matrix is,

Count	Actual RISE	Actual FALL
Predicted RISE	TP = 10	FP = 13
Predicted FALL	FN = 75	TN = 188

$$Accuracy = \frac{Correctly\ Predicted}{All\ Cases} = \frac{TP + TN}{TP + TN + FP + FN} = 69\%$$

The accuracy reaches a nice number, 69%. However it is not balanced in the sense that the number, 69%, does not reflect the issue that, out of 10+188 correct predictions, there are few cases that predict the stock RISE correctly. Or in other words, it predicts many cases that are supposed to RISE incorrectly. We know it is very important for the estimator to successfully identify a potentially RISE stock. So we need to look at balanced prediction performance.

$$Precision = \frac{TP}{All\ Positive\ Predictions} = \frac{TP}{TP + FP} = 43\%$$

$$Recall = \frac{TP}{All\ Positive\ Truth} = \frac{TP}{TP + FN} = 12\%$$

$$F1 = 2\frac{Precision * Recall}{Precision + Recall} = 19\%$$

Precision can be thought of as a measure of classifiers exactness. A low precision can also indicate a large number of False Positives. Recall can be thought as a measure of classifier completeness. A low recall indicates many False Negatives. And F1 score is limited by the smaller value of the two quantities. In this case, F1 score = 19%, indicating it performs poorly in prediction.

However, F1 has a weakness that models using F1 score can easily converge to a behavior that it predicts all the labels to be positive, given that positive truth and negative truth has relatively equal counts. For example, positive truth = 50 and negative truth = 50, a model predicts all labels to be positive will get precision = 50%, and recall = 100%, the resulting F1 score will be 66.7%. Therefore it ignores the true negative rate (the rate that given a negative truth label the classifier produce label negative). Here I define FA score:

$$FA = \min(F1, Accuracy) = 19\%$$

FA score considers the worst case between accuracy and F1 score, and takes care of both true positive rate (TPR) and true negative rate (TNR). Therefore FA score is used as metrics to compare different classification models.

Analysis

Data Exploration

Take Google's stock data for an example, Google's historic data as well as market indices are collected from Yahoo! Finance. Data is collected from 2000/01/01 to 2016/07/06.

For each day, the data collected has the category of open, adjusted close, high, low, close and volume. The adjusted close price is the data quantity that of most interest. Also, the volume is very important: there is a common sense that the price is rising and the volume is high, the price is highly possible to rise in the future. And there are also other important volume-related rules. Therefore, for indices I collect only the adjusted close prices, for the Google stock itself, I collect adjusted close prices and volume.

The tail 5 rows of the collected data pandas data frame is like below:

Date	Closing_GOOG	Closing_^IXIC	Closing_^DJI	Closing_^GSPC	\
2004-08-19	50.119968	1819.890015	10040.820312	1091.229980	
2004-08-20	54.100990	1838.020020	10110.139648	1098.349976	
2004-08-23	54.645447	1838.699951	10073.049805	1095.680054	
2004-08-24	52.382705	1836.890015	10098.629883	1096.189941	
2004-08-25	52.947145	1860.719971	10181.740234	1104.959961	

Date	Closing_^GDAXI	Closing_^N225	Closing_^HSI	Closing_^AXJO
2004-08-19	3722.989990	10903.530273	12396.669922	3486.300049
2004-08-20	3712.610107	10889.139648	12376.900391	3497.699951
2004-08-23	3772.139893	10960.969727	12431.769531	3524.000000
2004-08-24	3771.000000	10985.330078	12646.490234	3526.300049
2004-08-25	3788.879883	11130.019531	12793.030273	3517.399902

The data shown is only part of the whole data frame. The date of the data frame can be traced back to 2000/01/01. The volume data of Google is listed in a separate data frame for special processing for feature generation.

Take an observation of the data collected, the weekend or the public holidays are excluded, as the transaction of that day is not performed. In this regard, we treat the next working day as second day in stock prediction model and ignore “empty” days without a transaction record.

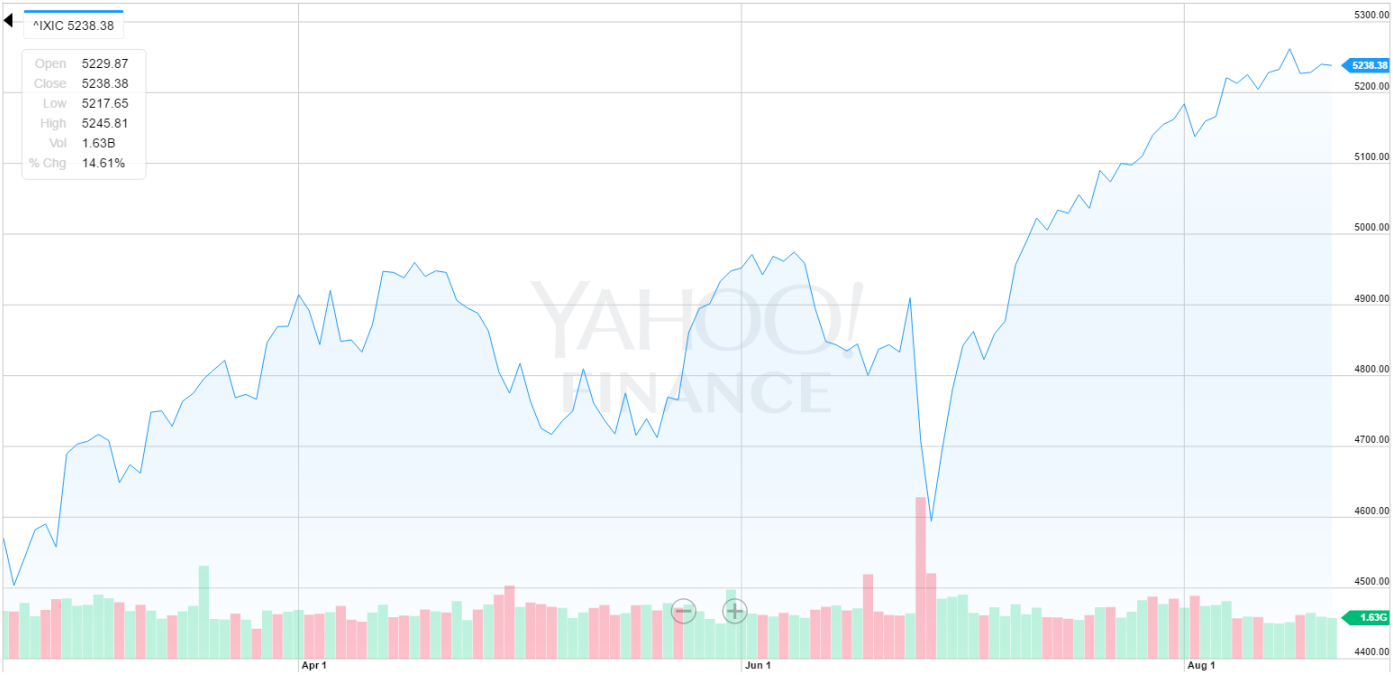
Data of each day only provides information of the day itself and does not reflect information of recent price dynamics, i.e., historic stock market running trend, whether it is experiencing a steady rising or falling slope or a perturbed intense jump. Therefore additional features are to be added to account for stock dynamics.

Data are of different scales. In the following implementation stage, they are processed by a natural logarithmic scaling in order to bring them to scale. Also, some quantities need to be normalized in order to have a valid differentiation, such as volume, and stock gain rate.

Visualization and Algorithms and Techniques

The plot of historic stock prices and indices are listed as below:

Google Daily Adjusted Close Price in USD, (source: Yahoo! Finance)



Indices NASDAQ Composite Daily Adjusted Close Price in USD, (source: Yahoo! Finance)

The algorithm is mainly two categories of tasks:

The first task category is feature generation. It processes the data and extracts valuable features that capture the recent dynamics of companies' stock and market indices. Firstly, for each day, I collect only the adjusted close prices of market indices and for the Google stock itself I collect adjusted close prices and volume. There are in total 7 indices that are considered in the model (NASDAQ Composite, Dow Jones Industrial Average, Standard and Poor's 500, Frankfurt DAX , Paris CAC 40, Tokyo Nikkei-225, Hong Kong Hang Seng, Australia ASX-200).

For date I and price type T (T can be Google adj. close price, or any market indices), one brute-force method of capturing the historic dynamics is to make all history prices values of T to be one of the features of date I. For example we have data for 2000 days before day i. We can directly treat all the 2000 days' Google historic stock prices to be date i's input feature. What's more, we also include all 2000 days' NASDAQ prices into the features, and include all the other indices prices the same way. Thus, with 1 stock price and 7 market indices, we have (1+7)*2000 features for one day; we pair this day the truth trend (RISE/FALL) to be the label. And use these training pairs to train the classification model. This way, we have only one pair of (features, label); the training data size is not enough to train the model.

Another way of choosing the features it to take the assumption that, as also can be observed in the price trending plot, only recent dynamics of affect the recent future trend the most and data that is far old in the history has trivial effect on the trend prediction. For example if we need to predict tomorrow's stock price, we cut our history evaluation window to only 15 days, which is day i, day i-1, day i-2… to day i-15. Thus for each day, to capture the recent market dynamics we need to include into features 15 days of history data of stock prices and those 7 market indices. In total, it includes 15 * (1 + 7) = 120 features in one day, which is an acceptable for some classifiers.

```
 Merged Data with past 13 day records as features:
['Closing_GOOG' 'Closing_^IXIC' 'Closing_^DJI' 'Closing_^GSPC'
 'Closing_^GDAXI' 'Closing_^N225' 'Closing_^HSI' 'Closing_^AXJO'
 'Closing_GOOG3' 'Closing_GOOG6' 'Closing_GOOG9' 'Closing_GOOG12'
 'Closing_GOOG15' 'Closing_GOOG18' 'Closing_GOOG21' 'Closing_GOOG24'
 'Closing_GOOG27' 'Closing_GOOG30' 'Closing_GOOG33' 'Closing_GOOG36'
 'Closing_GOOG39' 'Closing_^IXIC3' 'Closing_^IXIC6' 'Closing_^IXIC9'
 'Closing_^IXIC12' 'Closing_^IXIC15' 'Closing_^IXIC18' 'Closing_^IXIC21'
 'Closing_^IXIC24' 'Closing_^IXIC27' 'Closing_^IXIC30' 'Closing_^IXIC33'
 'Closing_^IXIC36' 'Closing_^IXIC39' 'Closing_^DJI3' 'Closing_^DJI6'
 'Closing_^DJI9' 'Closing_^DJI12' 'Closing_^DJI15' 'Closing_^DJI18'
 'Closing_^DJI21' 'Closing_^DJI24' 'Closing_^DJI27' 'Closing_^DJI30'
 'Closing_^DJI33' 'Closing_^DJI36' 'Closing_^DJI39' 'Closing_^GSPC3'
 'Closing_^GSPC6' 'Closing_^GSPC9' 'Closing_^GSPC12' 'Closing_^GSPC15'
 'Closing_^GSPC18' 'Closing_^GSPC21' 'Closing_^GSPC24' 'Closing_^GSPC27'
 'Closing_^GSPC30' 'Closing_^GSPC33' 'Closing_^GSPC36' 'Closing_^GSPC39'
 'Closing_^GDAXI3' 'Closing_^GDAXI6' 'Closing_^GDAXI9' 'Closing_^GDAXI12'
 'Closing_^GDAXI15' 'Closing_^GDAXI18' 'Closing_^GDAXI21'
 'Closing_^GDAXI24' 'Closing_^GDAXI27' 'Closing_^GDAXI30'
 'Closing_^GDAXI33' 'Closing_^GDAXI36' 'Closing_^GDAXI39' 'Closing_^N2253'
 'Closing_^N2256' 'Closing_^N2259' 'Closing_^N22512' 'Closing_^N22515'
 'Closing_^N22518' 'Closing_^N22521' 'Closing_^N22524' 'Closing_^N22527'
 'Closing_^N22530' 'Closing_^N22533' 'Closing_^N22536' 'Closing_^N22539'
 'Closing_^HSI3' 'Closing_^HSI6' 'Closing_^HSI9' 'Closing_^HSI12'
 'Closing_^HSI15' 'Closing_^HSI18' 'Closing_^HSI21' 'Closing_^HSI24'
 'Closing_^HSI27' 'Closing_^HSI30' 'Closing_^HSI33' 'Closing_^HSI36'
 'Closing_^HSI39' 'Closing_^AXJO3' 'Closing_^AXJO6' 'Closing_^AXJO9'
 'Closing_^AXJO12' 'Closing_^AXJO15' 'Closing_^AXJO18' 'Closing_^AXJO21'
 'Closing_^AXJO24' 'Closing_^AXJO27' 'Closing_^AXJO30' 'Closing_^AXJO33'
 'Closing_^AXJO36' 'Closing_^AXJO39']
```

Features for one day's stock price trend prediction

However, for support vector machine (SVM) classifier, the feature space dimension is still too large for it to be trained within an acceptable time. Therefore principal component analysis is applied to downsize the features to 10 synthesized features that capture most of information. Also, return, the stock price gain of two different days, and average return are calculated to further enhance the capture of recent stock dynamics.

The second task category is classification implementation and selection. AdaBoost, RandomForest, Gaussian Naïve Bayes, Decision Tree, SVM and K-Nearest Neighbors are used to implement the classifier model and compared with their FA score on validation set. It is advantageous to implement multiple candidate classifiers due to the stock trend behavior is different from case to case, one particular classifier may not be a perfect fit for all the stock trend behavior predictions. Therefore, for a company stock, these classifiers will all try to fit and predict data, the one who achieves best F1 score finally get the qualification to predict the final result.

Some of the algorithms that I included in my project are:

AdaBoost (AB):

AB is a linear classifier with all its desirable properties. AB output converges to the logarithm of likelihood ratio. AB has good generalization properties. AB is a feature selector with a principled strategy (minimization of upper bound on empirical error). It is close to sequential decision making (it produces a sequence of gradually more complex classifiers). AdaBoost is an algorithm for constructing a "strong" classifier as linear combination.

Advantages:

- Very simple to implement
- Feature selection on very large sets of features
- Fairly good generalization

Disadvantages

- Suboptimal solution for α^-
- Can overfit in presence of noise

RandomForest:

A random forest model is a collection of decision tree models that are combined together to make predictions. When you make a random forest, you have to specify the number of decision trees you want to use to make the model. The random forest algorithm then takes random samples of observations from your training data and builds a decision tree model for each sample. The random samples are typically drawn with replacement, meaning the same observation can be drawn multiple times. The end result is a bunch of decision trees that are created with different groups of data records drawn from the original training data.

The decision trees in a random forest model are a little different than the standard decision trees we made last time. Instead of growing trees where every single explanatory variable can potentially be used to make a branch at any level in the tree, random forests limit the variables that can be used to make a split in the decision tree to some random subset of the explanatory variables. Limiting the splits in this fashion helps avoid the pitfall of always splitting on the same variables and helps random forests create a wider variety of trees to reduce overfitting.

Random forests are an example of an ensemble model: a model composed of some combination of several different underlying models. Ensemble models often yields better results than single models because different models may detect different patterns in the data and combining models tends to dull the tendency that complex single models have to overfit the data.

Gaussian NB:

Naive Bayes methods are a set of supervised learning algorithms based on applying Bayes' theorem with the "naive" assumption of independence between every pair of features. Given a class variable y and a dependent feature vector x_1 through x_n, Bayes' theorem states the following relationship:

$$P(y \mid x_1, \ldots, x_n) = \frac{P(y)P(x_1, \ldots x_n \mid y)}{P(x_1, \ldots, x_n)}$$

The different naive Bayes classifiers differ mainly by the assumptions they make regarding the distribution of

$P(x_i \mid y)$.

In spite of their apparently over-simplified assumptions, naive Bayes classifiers have worked quite well in many real-world situations, famously document classification and spam filtering. They require a small amount of training data to estimate the necessary parameters. (For theoretical reasons why naive Bayes works well, and on which types of data it does, see the references below.)

Naive Bayes learners and classifiers can be extremely fast compared to more sophisticated methods. The decoupling of the class conditional feature distributions means that each distribution can be independently estimated as a one dimensional distribution. This in turn helps to alleviate problems stemming from the curse of dimensionality.

On the flip side, although naive Bayes is known as a decent classifier, it is known to be a bad estimator, so the probability outputs from predict_proba() are not to be taken too seriously.

Decision Tree:

It uses a decision tree as a predictive model which maps observations about an item to conclusions about the item's target value. It is one of the predictive modelling approaches used in statistics, data mining and machine learning. Tree models where the target variable can take a finite set of values are called classification trees. In these tree structures, leaves represent class labels and branches represent conjunctions of features that lead to those class labels. Decision trees where the target variable can take continuous values (typically real numbers) are called regression trees. In decision analysis, a decision tree can be used to visually and explicitly represent decisions and decision making. In data mining, a decision tree describes data but not decisions; rather the resulting classification tree can be an input for decision making. This page deals with decision trees in data mining.

Amongst other data mining methods, decision trees have various advantages:

- Simple to understand and interpret. People are able to understand decision tree models after a brief explanation.
- Requires little data preparation. Other techniques often require data normalization, dummy variables need to be created and blank values to be removed.
- Able to handle both numerical and categorical data. Other techniques are usually specialized in analyzing datasets that have only one type of variable

Limitations:

- The problem of learning an optimal decision tree is known to be NP-complete under several aspects of optimality and even for simple concepts. Consequently, practical decision-tree learning algorithms are based on heuristics such as the greedy algorithm where locally-optimal decisions are made at each node. Such algorithms cannot guarantee to return the globally-optimal decision tree. To reduce the greedy effect of local-optimality some methods such as the dual information distance (DID) tree were proposed.
- Decision-tree learners can create over-complex trees that do not generalize well from the training data. This is known as overfitting. Mechanisms such as pruning are necessary to avoid this problem.
- There are concepts that are hard to learn because decision trees do not express them easily, such as XOR, parity or multiplexer problems. In such cases, the decision tree becomes prohibitively large.

Support Vector Machines:

SVMs, also support vector networks are supervised learning models with associated learning algorithms that analyze data used for classification and regression analysis. Given a set of training examples, each marked for belonging to one of two categories, an SVM training algorithm builds a model that assigns new examples into one category or the other, making it a non-probabilistic binary linear classifier. An SVM model is a representation of the examples as points in space, mapped so that the examples of the separate categories are divided by a clear gap that is as wide as possible. New examples are then mapped into that same space and predicted to belong to a category based on which side of the gap they fall on. In addition to performing linear classification, SVMs can efficiently perform a non-linear classification using what is called the kernel trick, implicitly mapping their inputs into high-dimensional feature spaces.

Applications:

- SVMs can be used to solve various real world problems of Uncertainty in Knowledge-Based Systems
- SVMs are helpful in text and hypertext categorization as their application can significantly reduce the need for labeled training instances in both the standard inductive and transductive settings.
- Classification of images can also be performed using SVMs. Experimental results show that SVMs achieve significantly higher search accuracy than traditional query refinement schemes after just three to four rounds of relevance feedback.
- Hand-written characters can be recognized using SVM.

Issues:

- Requires full labeling of input data
- Uncalibrated class membership probabilities
- The SVM is only directly applicable for two-class tasks. Therefore, algorithms that reduce the multi-class task to several binary problems have to be applied; see the multi-class SVM section.
- Parameters of a solved model are difficult to interpret.

K-Nearest Neighbors:

K-Nearest Neighbors algorithm is a non-parametric method used for classification and regression. In both cases, the input consists of the k closest training examples in the feature space. The output depends on whether k-NN is used for classification or regression. In k-NN classification, the output is a class membership. An object is classified by a majority vote of its neighbors, with the object being assigned to the class most common among its k nearest neighbors (k is a positive integer, typically small). If k = 1, then the object is simply assigned to the class of that single nearest neighbor.

In k-NN regression, the output is the property value for the object. This value is the average of the values of its k nearest neighbors. k-NN is a type of instance-based learning, or lazy learning, where the function is only approximated locally and all computation is deferred until classification. The k-NN algorithm is among the simplest of all machine learning algorithms.

k-NN is a special case of a variable-bandwidth, kernel density "balloon" estimator with a uniform kernel. The naive version of the algorithm is easy to implement by computing the distances from the test example to all stored examples, but it is computationally intensive for large training sets. Using an appropriate nearest neighbor search algorithm makes k-NN computationally tractable even for large data sets. Many nearest neighbor search algorithms have been proposed over the years; these generally seek to reduce the number of distance evaluations actually performed. k-NN has some strong consistency results. As the amount of data approaches infinity, the two-class k-NN algorithm is guaranteed to yield an error rate no worse than twice the Bayes error rate (the minimum achievable error rate given the distribution of the data).[10] Various improvements to the k-NN speed are possible by using proximity graphs.

Benchmark

One way of benchmark is from the domain knowledge. It is well known that stock prediction is highly random and is affected by numerous factors. People find it very challenging to find a prediction solution that achieves a steady probability win versus random guess, i.e., an prediction accuracy more than 50%. Because once you achieve prediction that constantly has accuracy of more than 50%, you can take advantage of it and make profit. Therefore, an essential benchmark is to evaluate the classifier on the test set and evaluate how much more prediction accuracy it gets with respect to 50% (random guess).

Different solutions are compared by F1 and accuracy score on its validation set. Since the validation dataset is generated by the real case and the performance of classifier of validation set can be projected to be the performance in the future.

Methodology

Data Preprocessing

The data is retrieved by pandas.io.data interfaces. It directly gets data from Yahoo Finance:

```
def getStockData(symbol, start, end, getVolume=False):
    """Downloading Stock data from Yahoo Finance dataset from pandas method
       and calculate daily returns based on Adj Close column"""
    df = get_data_yahoo(symbol, start, end)

    df.rename(columns = {'Day Close' : 'Closing' + '_' + symbol}, inplace=True)

    ed = -2 if getVolume else -1
    return df.drop(df.columns[:ed], axis = 1)
```

The function getStockData() also processes the collected data in which it has open, close, high, low and volume information. It collects only adj. close price for market indices and only adj. close price and volume for the company in interest.

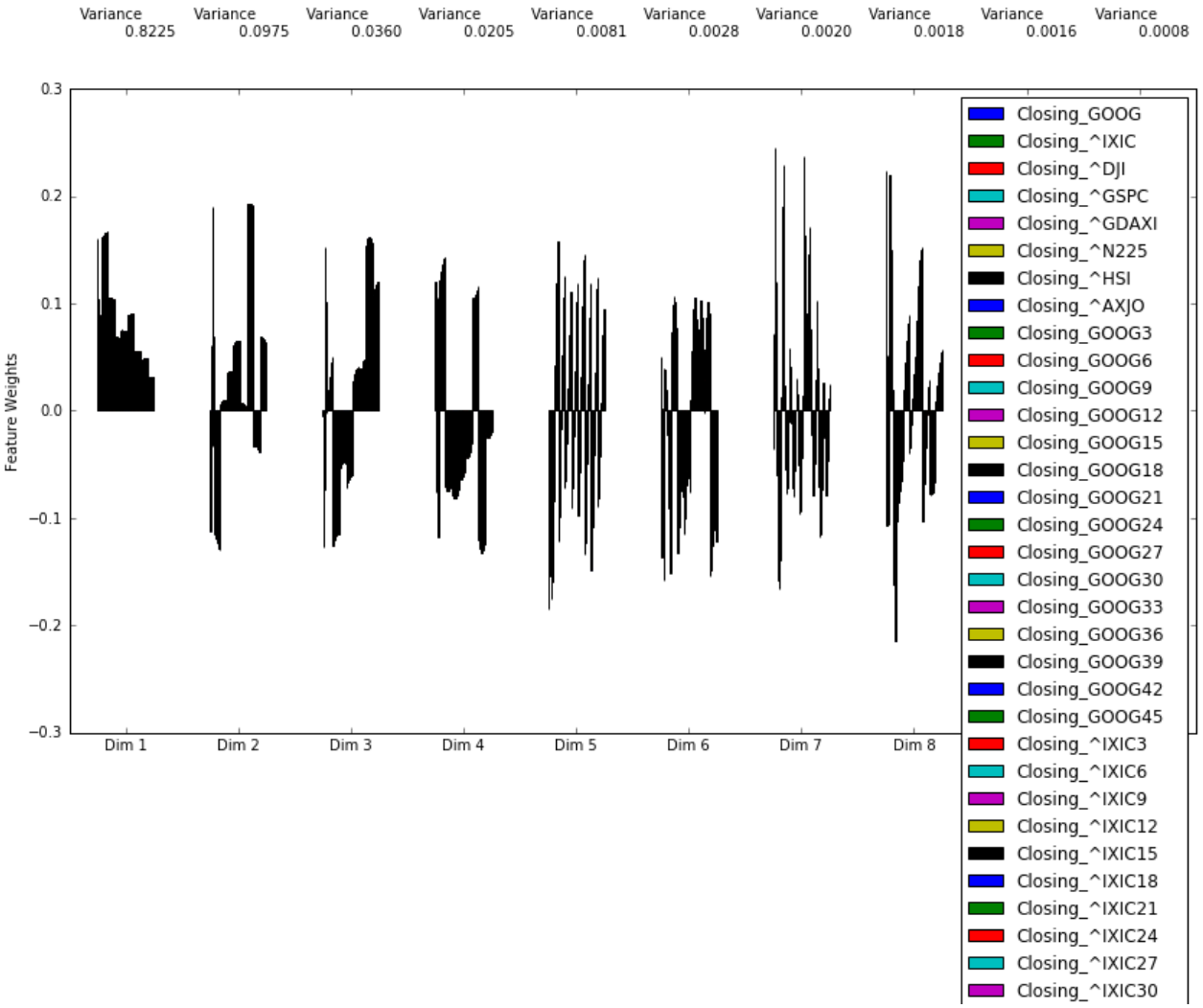
After data collection from yahoo, data is concatenated to one data frame, meg_data. And add history data to every day's row. Here I defined two parameters:

Forecast: the number of days you want to make trend prediction after data end time, if data end time = today and forecast = 3, it means the model is to predict stock trend the day after tomorrow.

N_history: the number of days you observe in the history to capture data recent dynamics. For example when forecast = 1 and N_history = 15 and suppose the observed day is day i, it means day i-1, i-2,…,i-15 trading record (adj. close price for company, price of indices) are added to the meg_data. If forecast = 2 and N history = 15, it means day i-2, i-4,…,i-30 trading record (adj. close price for company, price of indices) are added to the meg_data.
After the shift and addition, each row of meg_data contains 15 * (1+7) columns. It added N_history=15 days' historic stock prices and indices in to each day. Here note the heading rows does not have N_history days historic data, therefore they are dropped, as they don't have sufficient defined features.

Then principal component analysis (PCA) is performed to downsize the feature space.

To perform PCA, it is very important to put every category of numbers to similar scale. Therefore natural logarithm is performed for all the data in meg_data, resulting in log_data. Define PCA_n to be the feature dimension we want to downsize to. Here PCA_n = 10. After fitting the data using PCA.fit(), the log_data is transformed to reduced_data through PCA.transform().



PCA feature weights for the transformed dimensions

PCA analysis reports data variance mainly lies in the first 4 dimensions. Here I include PCA_n = 10 dimensions in order to preserve the information. Note the 10^{th} dimension has the explained variance to be 0.008. The trivial number indicates first 10 dimensions capture most of information. As a result of PCA, reduced_data shrinks meg_data 126 features down to 10 features, relaxing the training time for the entire model.

However, past N_history days absolute value data may not be sufficient to provide information of recent stock and market dynamics. Therefore for day i, I define:

$$return_i = \frac{adj.close_i - adj.close_{i-1}}{adj.close_{i-1}}$$

Return of day i is defined as price gain from its yesterday to day i.

$$return_{i,\boldsymbol{\Delta}} = \frac{adj.close_i - adj.close_{i-\boldsymbol{\Delta}}}{adj.close_{i-\boldsymbol{\Delta}}}$$

Return of day i with **Δ** is defined as price gain from day i-**Δ** to day i.

$$avgReturn_{i,\boldsymbol{\Delta}} = \frac{return_i + return_{i-1} + return_{i-2} + \cdots + return_{i-\Delta+1}}{\Delta}$$

avgReturn of day i with Δ is defined as averaged return from day i-Δ+1 today i.

For example we can evaluate the recent stock dynamics by include $return_j$, $reutrn_{i,\Delta}$, and $avgReturn_{i,\Delta}$ in the feature, with j ranges in i - forecast*return_eval_list, and Δ ranges in forecast*return_eval_list, and return_eval_list = [1,2,…,N_history]. With N_history = 15, 15+15+15=45 more features will be included. However this is not the most efficient way. We have the assumption that history far from now affects future less than recent history. Therefore the increasing step of the number in the return_eval_list should be gradually enlarged. Here I used Fibonacci Series = [1,2,3,5,8,13….N_history] here to further downsize the feature number. As a result, suppose N_history = 15 and forecast = 3, the program generates the below return information for Google's stock price:

```
Return data
              return   return6   return9  return15  return24  return39  \
Date
2016-08-15 -0.002321  0.014036  0.017755  0.059313  0.125230  0.089340
2016-08-16 -0.009609 -0.006494  0.005512  0.046315  0.101342  0.080983
2016-08-17 -0.004226 -0.002367  0.008704  0.054260  0.090646  0.097908
2016-08-18 -0.006314 -0.008620  0.007633  0.052924  0.078902  0.124010
2016-08-19 -0.002213 -0.011801 -0.008693  0.045364  0.081509  0.117787

            avgReturn6  avgReturn9  avgReturn15  avgReturn24  avgReturn39
Date
2016-08-15    0.007045    0.013826     0.011880     0.014998     0.006537
2016-08-16    0.003429    0.008098     0.010717     0.013958     0.006621
2016-08-17    0.000875    0.003567     0.010563     0.012747     0.006835
2016-08-18   -0.002909    0.002458     0.010161     0.010969     0.007621
2016-08-19   -0.003803    0.000869     0.010100     0.010207     0.008463
```

Another quantity is volume of the stock. Volume data need to be normalized so that the corresponding volume feature knows how large or how small today's volume is. Define:

$$volume\_norm_i = \frac{volume_i - avg\_volume_i}{std\_volume_i}$$

$$avg\_volume_i = mean\{volume_i, volume_{i-1}, \dots volume_{i-N_{history}*forcast+1}\}$$

$$std\_volume_i = std\{volume_i, volume_{i-1}, \dots volume_{i-N_{history}*forcast+1}\}$$

With these quantities as features, day i is able to preserve most important information of recent stock price dynamics. In total there are 22 features generated:

[0 1 2 3 4 5 6 7 8 9 'return' 'return6' 'return9' 'return15' 'return24' 'return39' 'avgReturn6' 'avgReturn9' 'avgReturn15' 'avgReturn24' 'avgReturn39' 'Volume_norm']

Where 0,1,…9 represents the downsized features generated by PCA.

For labels,

$$y_i = sgn\{return_{i+forcast,forcast}\}$$

It is generated by shifting $return_{i+forecast,forecast}$ by –forecast rows. Note the tailing forecast rows should be dropped, as they do not have corresponding labels. For example the end date is 2016/08/20 and forecast = 5, rows with date 2016/08/20, 2016/08/19, 2016/08/18, 2016/08/17, 2016/08/16 should be dropped.

After searching for outliers beyond 1.5x interquartile range (IQR), there are no outliers. This is because the data for everyday is real trading data and eliminates the irrational data possibilities.

Implementation

Given the generated features and RISE/FALL labels, the overall data is split into training set and test (validation) set:

```
# TODO: Split the data into training and testing sets using the given feature as the target
from sklearn.cross_validation import train_test_split

X_train, X_test, y_train, y_test = train_test_split(data, y, test_size=0.25, random_state=42)
```

The test set is set to be 0.25 of overall data.

A classifier training function which performs both normal training and cross-validation grid search training is implemented as below:

```
def train_classifier(clf, X_train, y_train, gridSearch=False, parameters=None):

    print X_train.shape, y_train.shape
    if not gridSearch:
        clf.fit(X_train, y_train)
    else:
        f1_scorer = make_scorer(f1_score, pos_label=1)
        accu_scorer = make_scorer(accuracy_score)
        fa_scorer = make_scorer(fa_score)
        grid_obj = GridSearchCV(clf, parameters, scoring = fa_scorer)
        grid_obj.fit(X_train, y_train)
        print "GridSearch Best Parameters: ", grid_obj.best_params_, '=', grid_obj.best_score_
        clf = grid_obj.best_estimator_

    return clf
```

The cross-validation grid search object is created using the FA score, which evaluates the minimum of F1 score and accuracy score to ensure balanced TPR and TNR.

After the training function implementation, it is simple to apply a type of classifier, for example:

```
from sklearn import svm

clf_SVM = svm.SVC(C = 36, gamma = 4, degree = 2, kernel='poly', probability=True)

param_SVM = {'C': [1,6,36], 'gamma': [0.5,2,4], 'degree': [2,3]}
clf_SVM, f1_SVM = train_predict(clf_SVM, X_train, y_train, X_test, y_test, gridSearch=True, parameters = param_SVM)

plotROC(clf_SVM, X_test, y_test)

clf_best, score_best = compBestClf(clf_SVM, f1_SVM, clf_best, score_best)
```

The adaptive boosting classifier is applied to the data using GridSearchCV. 'n_estimators' and 'learning_rate' parameter are swept to achieve the best fit while not overfitting. However for naïve Bayes, there is no need to perform GridSearchCV because there are few parameters to tune:

```
from sklearn.naive_bayes import GaussianNB

clf_NB = GaussianNB()

clf_NB, f1_NB = train_predict(clf_NB, X_train, y_train, X_test, y_test)

plotROC(clf_NB, X_test, y_test)

# compare best clf
clf_best, score_best = compBestClf(clf_NB, f1_NB, clf_best, score_best)
```

Classifier algorithms adaptive boosting, random forest, Gaussian naïve Bayes, decision tree, support vector machine and k-nearest neighbors are applied to seek for best classifier fit by comparing their FA score on validation set. It is advantageous to implement multiple candidate classifiers due to the stock trend behavior is different from case to case (industry, data time, forecast days), one particular classifier may not be a perfect fit for all the stock trend behavior predictions. Therefore, for a company stock, these classifiers will all try to fit and predict data, the one who achieves best FA score finally get the qualification to predict the final result.

Refinement

The initial solution is brute-force and simple: make the history stock data to be features of today and perform a support vector machine (SVM) classifier to learn the historic dynamics of stock and predict. The initial solution results in thousands of features (because there are thousands of days in the history) per day, which takes too long time for SVM to learn and converge. Also, the validation result is barely better than a random guess.

Therefore a medium solution is come up that more features such as world market indices (NASDAQ, Dow Jones, etc.) are involved and principal component analysis (PCA) is performed to downsize the feature dimensionality while preserving most of information. As a result, the amount of features is reduced to 10 and SVM is able to train fast and make predictions. However the solution does not induce a marginal improvement.

A final solution is to introduce the concept of return and average return of Fibonacci-Series-numbered days to account for the characteristics of history stock trend. Moreover, not only SVM, but also classifier algorithms such as adaptive boosting, random forest, Gaussian naïve Bayes, decision tree, and k-nearest neighbors are applied to seek for best classifier fit by comparing their FA score on validation set. The reason of implementing multiple candidate classifiers is that the stock trend behavior is different from case to case (industry, data time, forecast days); one particular classifier may not be a perfect fit for all the stock trend behavior predictions.

The estimator has been trained on machine-learning models on historic data and predict stock price trend with confidence. Multiple company estimation is supported and results are summarized in a printed table:

```
//////////////////////////////////////////////
        PREDICTION SUMMARY
//////////////////////////////////////////////
FORCAST [days]: 5
END TIME: None

SYMBOL          TREND           MODEL FA SCORE          FALL/RISE CONFIDENCE
GOOG            FALL            0.736280487805          [[ 0.54105093  0.45894907]]
AAPL            FALL            0.707877461707          [[ 0.53503137  0.46496863]]
FB              RISE            0.768518518519          [[ 0.46906317  0.53093683]]
T               RISE            0.699124726477          [[ 0.24064529  0.75935471]]
CSCO            RISE            0.719912472648          [[ 0.43802483  0.56197517]]
GE              FALL            0.754385964912          [[ 0.51743759  0.48256241]]
INTC            RISE            0.706783369803          [[ 0.39879557  0.60120443]]
QCOM            RISE            0.693654266958          [[ 0.47097467  0.52902533]]
C               RISE            0.714442013129          [[ 0.4713214  0.5286786]]
BAC             RISE            0.72647702407           [[ 0.45870076  0.54129924]]
JPM             RISE            0.666301969365          [[ 0.47785479  0.52214521]]
GS              FALL            0.727571115974          [[ 0.53259418  0.46740582]]
```

Results

Model Evaluation and Validation

GOOG stock [2000/01/01-2016/08/20], forecast = 3 days					
Data size=2665, test set portion=0.25, feature dimension=22					
Classifiers	AdaBoost	RandomForest	GuassianNB	DecisionTree	KNN
Train accuracy	0.674	0.932	0.537	0.779	0.629
Train F1 score	0.711	0.939	0.635	0.808	0.643
Train FA score	0.674	0.932	0.537	0.779	0.629
Test accuracy	0.587	0.645	0.488	0.578	0.533
Test F1 score	0.658	0.719	0.609	0.651	0.568
Test FA score	0.587	0.645	0.488	0.578	0.533
FA score variance	<0.001	0.292	0.02	0.001	0.07
Test ROC AUC	0.57	0.69	0.44	0.57	0.54
GridSearchCV best params.	n_esti.=50 learning_rate =1	n_esti.=160 max_depth=8	N/A	min_split=2 max_depth=8	N_neighbors=16

Note FA score = min {F1, accuracy}

The reason that I didn't include **SVM** in my results is due to my system limitations (as per my knowledge). I have included the SVM code as a screenshot in classifier example and have also provided the same in attached python notebook. My system configuration is 8 GB ram and Intel i7 processor. Also I ran it on a machine with 16GB ram and on AWS with 32 GB ram but the program ran for more than 10 hours on each of the machines but no output. But if you could please run this and provide me the results then that would be much appreciated.

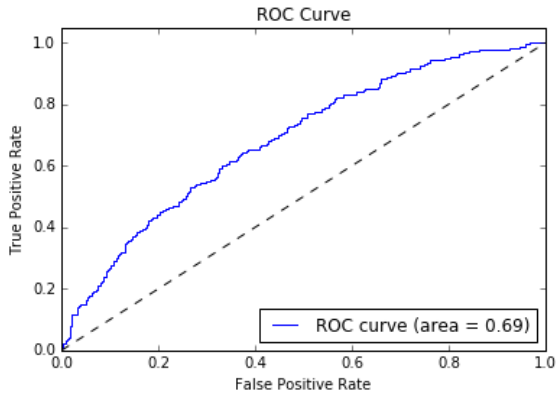
The above 5 models are implemented and compared by their performance: Thanks to grid search technique, all the models do not have a very large variance so that the models are not over-fitting. However the variance of random forest

is as large as 29%, note this is the best situation in the parameter sweep space. Random forest algorithm performs the best in terms of validation set FA score.

Random forest is an averaging algorithm based on randomized decision trees. The algorithms are perturb-and-combine techniques [1] specifically designed for trees. A diverse set of classifiers is created by introducing randomness in the classifier construction. The prediction of the ensemble is given as the averaged prediction of the individual classifiers. Therefore random forest is more capable to capture and collect the random-occurring non-linear factors that affect the future stock prices the most.

The quality of model is examined by alternating input features and evaluate receiver operating characteristic (ROC) curve. When number of PCA analysis features is reduced to 4, the algorithm can still retain an accuracy of 55%, which proves it is a robust model. Another way of examining model quality is to plot the ROC:

```
Best Classifier is:
RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
            max_depth=8, max_features='auto', max_leaf_nodes=None,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, n_estimators=160, n_jobs=-1,
            oob_score=False, random_state=None, verbose=0,
            warm_start=False)
Best Classifier's Score on test:  0.644776119403
ROC AUC: 0.69
```




```
FORCAST [days]: 3
END TIME: 2016-08-20 00:00:00

SYMBOL          TREND           MODEL FA SCORE          FALL/RISE CONFIDENCE
GOOG            RISE            0.644776119403          [[ 0.43731777  0.56268223]]
Predict:
GOOG stock after 3 days of 2016-08-19 00:00:00 is going to (with confidence FALL/RISE = [[ 0.43731777  0.56268223]] ): RISE
```

ROC curves have true positive rate on the Y-axis, and false positive rate on the X-axis. This means that the top left corner of the plot is the “ideal” point - a false positive rate of zero, and a true positive rate of one. Since it is ideal to maximize the true positive rate while minimizing the false positive rate, a steep ROC curve and a larger area under the curve (AUC) is usually better. The implemented random forest algorithm achieves 0.69 AUC, which is a reasonable number for stock prediction and proves the quality of the model.

Justification

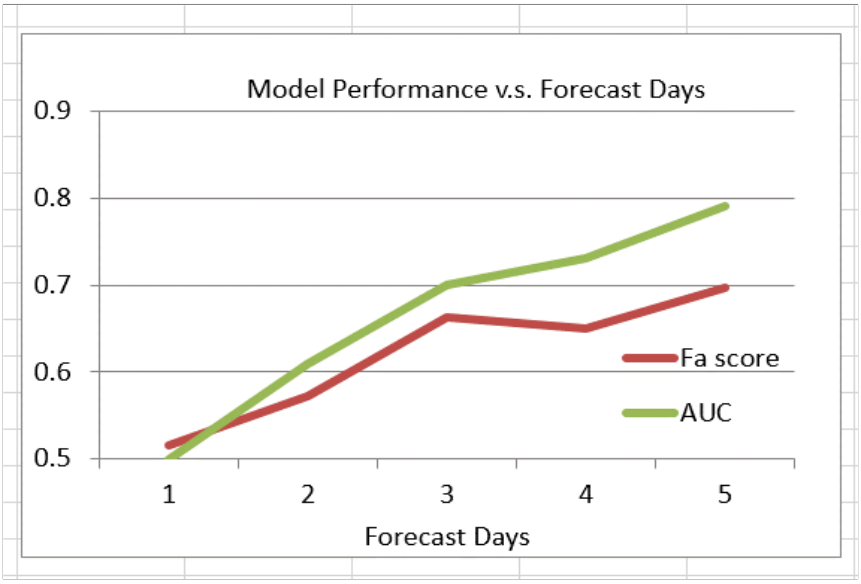
Previous session has explained and concluded that a better-performance model has higher FA score than other models and has a marginal increase of accuracy by 50% random guess. The random forest classifier achieves highest FA score =

0.65 among all the classifiers, which involves accuracy score = 0.65 and F1 score = 0.72. The 65% accuracy on the validation set has 15% more than random guess 50%, which is a marginal probability indicating that the classifier performs well.

Conclusion

Free-Form Visualization

Take Google stock as example, the classifier results in different accuracy and ROC area under curve (AUC) with different forecast date queries:



As can be seen from the above figure, the horizontal axis is the number of days to forecast the stock, the vertical axis is the FA score and AUC it achieved in the test set. An interesting point is that as forecast days increases, the achieved FA score and AUC has an increasing trend. It reflects that the more days the classifier forecasts, the more accurate it gets. This phenomenon is expected because in the near future the prediction result is easily disturbed by many factors or noises while in the far future the trend is mainly determined by the long-term slowly-changing reasons such as company policy, economic situation, etc.

Reflection

This project implemented supervised learning techniques to empower stock analysis and price trend estimation. The core interesting work enabling the system are summarized as below:

1. Feature generation from data. How to extract essential information from data and generate features differentiates the performance of many machine learning works. Designers should apply domain knowledge to transform and cover core information using a limited number of features. A situation that often occurs is that the feature space is too large that the implemented classifiers suffer long training time and not being able to converge. The introduction of PCA and ICA lends a powerful hand for the issue and transforms data to a few synthesized features while pertaining most information.

2. Machine learning model selection. In this project a classifier instead of a regression is implemented because stock price is a complex statistical problem and predicting price rise or fall simplifies the problem model from regression to binary-output classification. The rise or fall trend prediction can be more accurate than predicting prices and can provide more trustable and valuable prediction result and indeed provide a practical investment suggestion in reality. Metrics should be selected to effectively compare performance of different algorithms. Appropriate parameters should be searched to get best training result and prevent over-fitting. For example in this project, a customized FA score is proposed to better measure accuracy as well as true positive rate (TPR) and true negative rate (TNR) balance. By using this metric, failed classifier such as the one outputting constantly positive label is sifted out.

Throughout the entire project, I gained a firm understanding of machine learning knowledge and rich practice experience to tackle with numerous implementation problems and design trade-offs.

Improvement

Another possible direction of improvement of the current algorithm is to implement regression. It is interesting to evaluate with the same features how well will the regression model predict the future price compared to current work. Though the predicted price may not be very close to the actual price, the amount of predicted price gain indeed can act as a price trend predictor indicating the prediction confidence.

In order to further verify the system, a trading strategy could be followed up and an automatic trading system can be designed. If an automatic trading system can successfully predict stock trend using the classifier in the project and make profit, it proves the stock prediction works effectively. Otherwise it is a good test vehicle to test the potential design shortcomings and address the issues.

References

[1] L. Breiman, “Arcing Classifiers”, Annals of Statistics 1998.
[2] Ren, He, and Quan Yang. "Predicting and Evaluating the Popularity of Online News."
[3] Hall, Mark A. Correlation-based feature selection for machine learning. Diss. The University of Waikato, 1999.