Stock Trend Estimator

Capstone Project, Machine Learning Engineer Nanodegree

Lucas Li

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)
- London FTSE-100 (^FTSE 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, the stock trend estimator 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 stock trend estimator, the following problems are 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.

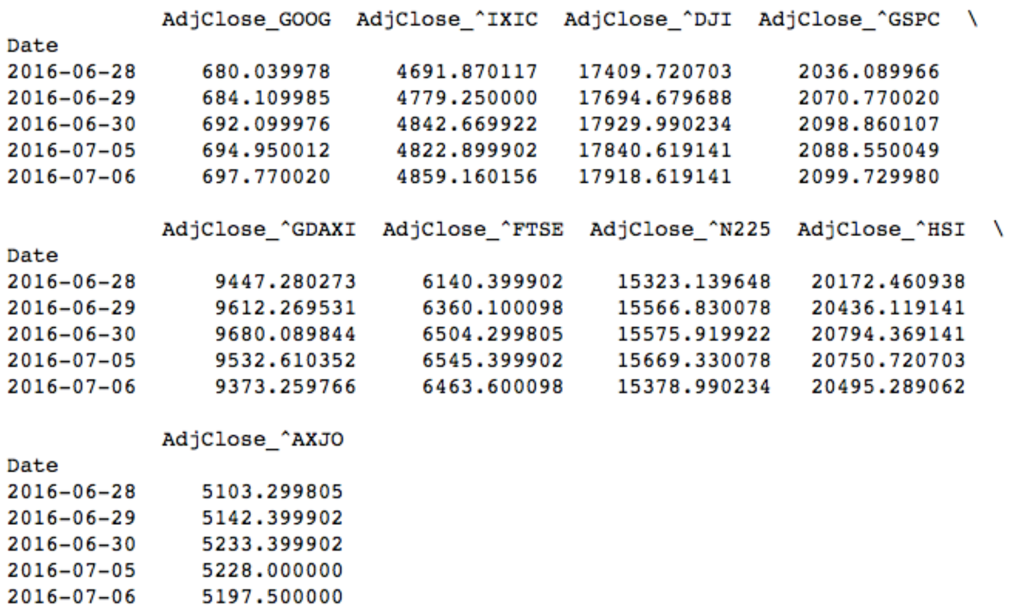
Analysis

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:



```
            AdjClose_GOOG  AdjClose_^IXIC  AdjClose_^DJI  AdjClose_^GSPC  \
Date
2016-06-28     680.039978     4691.870117   17409.720703     2036.089966
2016-06-29     684.109985     4779.250000   17694.679688     2070.770020
2016-06-30     692.099976     4842.669922   17929.990234     2098.860107
2016-07-05     694.950012     4822.899902   17840.619141     2088.550049
2016-07-06     697.770020     4859.160156   17918.619141     2099.729980

            AdjClose_^GDAXI  AdjClose_^FTSE  AdjClose_^N225  AdjClose_^HSI  \
Date
2016-06-28      9447.280273     6140.399902    15323.139648   20172.460938
2016-06-29      9612.269531     6360.100098    15566.830078   20436.119141
2016-06-30      9680.089844     6504.299805    15575.919922   20794.369141
2016-07-05      9532.610352     6545.399902    15669.330078   20750.720703
2016-07-06      9373.259766     6463.600098    15378.990234   20495.289062

            AdjClose_^AXJO
Date
2016-06-28     5103.299805
2016-06-29     5142.399902
2016-06-30     5233.399902
2016-07-05     5228.000000
2016-07-06     5197.500000
```

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.

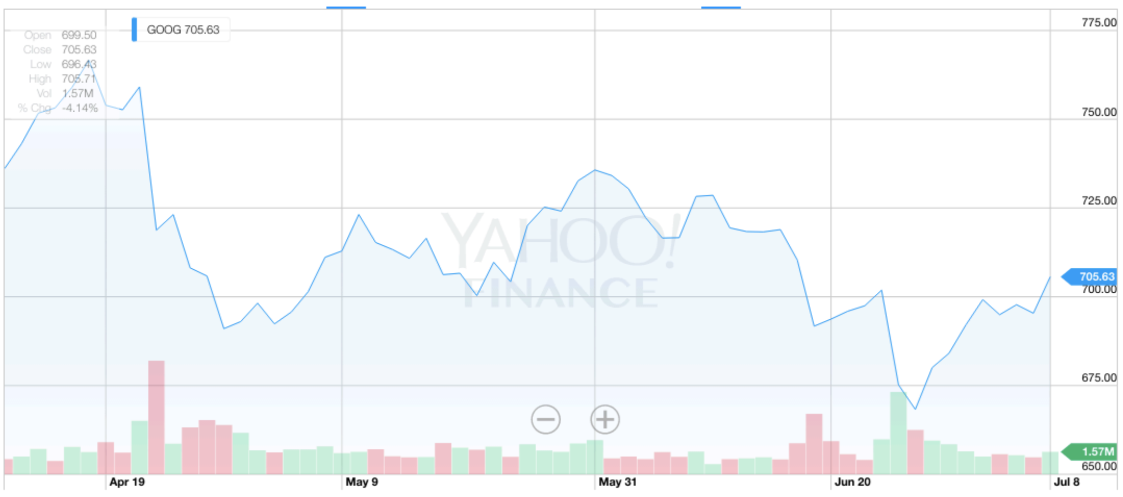
Take an observation of the data collected, the weekend or the public holidays are excluded, as the transaction of that day is not performed. In this regard, we treat the next working day as second day in stock prediction model and ignore "empty" days without a transaction record.

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.

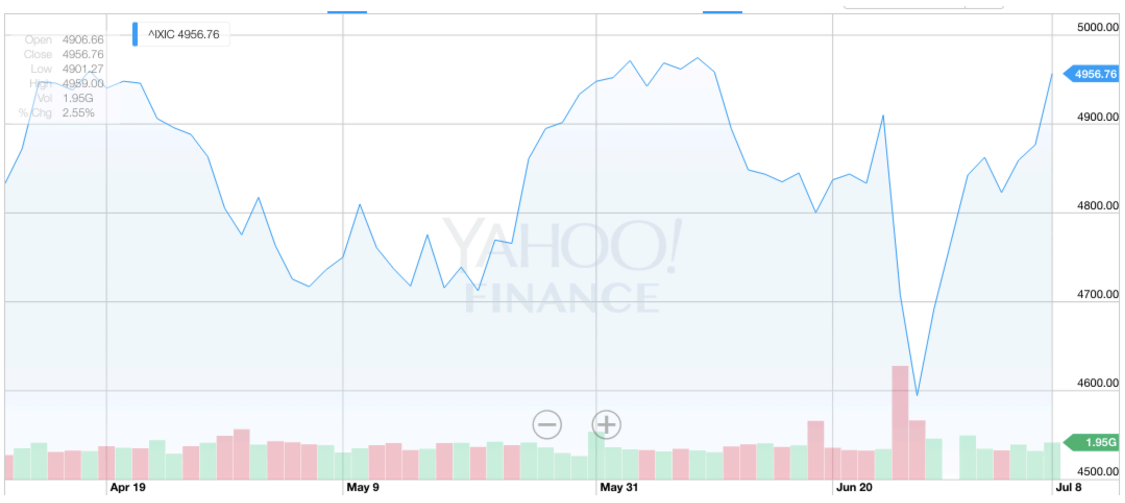
Data are of different scales. In the following implementation stage, they are processed by a natural logarithmic scaling in order to bring them to scale. Also, some quantities need to be normalized in order to have a valid differentiation, such as volume, and stock gain rate.

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)



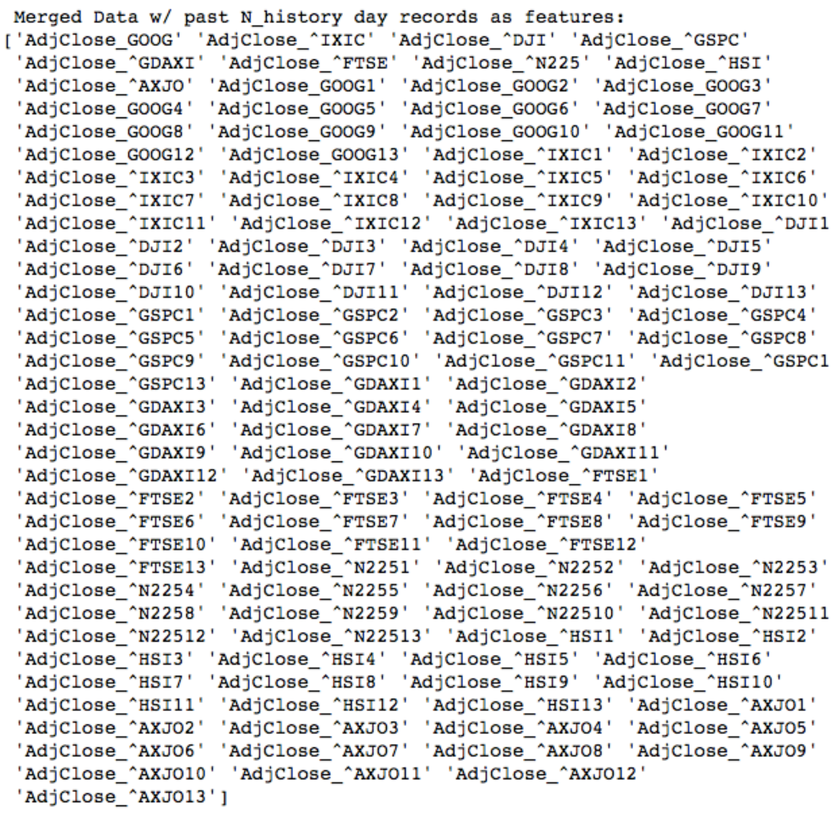
Indices NASDAQ Composite Daily Adjusted Close Price in USD, (source: Yahoo! Finance)

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 8 indices that are considered in the model (NASDAQ Composite, Dow Jones Industrial Average, Standard and Poor's 500, Frankfurt DAX , London FTSE-100 , 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 8 market indices, we have (1+8)*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 13 days, which is day i, day i-1, day i-2… to day i-13. Thus for each day, to capture the recent market dynamics we need to include into features 13 days of history data of stock prices and those 8 market indices. In total, it includes 14 * (1 + 8) = 126 features in one day, which is an acceptable for some classifiers.



```
 Merged Data w/ past N_history day records as features:
['AdjClose_GOOG' 'AdjClose_^IXIC' 'AdjClose_^DJI' 'AdjClose_^GSPC'
 'AdjClose_^GDAXI' 'AdjClose_^FTSE' 'AdjClose_^N225' 'AdjClose_^HSI'
 'AdjClose_^AXJO' 'AdjClose_GOOG1' 'AdjClose_GOOG2' 'AdjClose_GOOG3'
 'AdjClose_GOOG4' 'AdjClose_GOOG5' 'AdjClose_GOOG6' 'AdjClose_GOOG7'
 'AdjClose_GOOG8' 'AdjClose_GOOG9' 'AdjClose_GOOG10' 'AdjClose_GOOG11'
 'AdjClose_GOOG12' 'AdjClose_GOOG13' 'AdjClose_^IXIC1' 'AdjClose_^IXIC2'
 'AdjClose_^IXIC3' 'AdjClose_^IXIC4' 'AdjClose_^IXIC5' 'AdjClose_^IXIC6'
 'AdjClose_^IXIC7' 'AdjClose_^IXIC8' 'AdjClose_^IXIC9' 'AdjClose_^IXIC10'
 'AdjClose_^IXIC11' 'AdjClose_^IXIC12' 'AdjClose_^IXIC13' 'AdjClose_^DJI1
 'AdjClose_^DJI2' 'AdjClose_^DJI3' 'AdjClose_^DJI4' 'AdjClose_^DJI5'
 'AdjClose_^DJI6' 'AdjClose_^DJI7' 'AdjClose_^DJI8' 'AdjClose_^DJI9'
 'AdjClose_^DJI10' 'AdjClose_^DJI11' 'AdjClose_^DJI12' 'AdjClose_^DJI13'
 'AdjClose_^GSPC1' 'AdjClose_^GSPC2' 'AdjClose_^GSPC3' 'AdjClose_^GSPC4'
 'AdjClose_^GSPC5' 'AdjClose_^GSPC6' 'AdjClose_^GSPC7' 'AdjClose_^GSPC8'
 'AdjClose_^GSPC9' 'AdjClose_^GSPC10' 'AdjClose_^GSPC11' 'AdjClose_^GSPC1
 'AdjClose_^GSPC13' 'AdjClose_^GDAXI1' 'AdjClose_^GDAXI2'
 'AdjClose_^GDAXI3' 'AdjClose_^GDAXI4' 'AdjClose_^GDAXI5'
 'AdjClose_^GDAXI6' 'AdjClose_^GDAXI7' 'AdjClose_^GDAXI8'
 'AdjClose_^GDAXI9' 'AdjClose_^GDAXI10' 'AdjClose_^GDAXI11'
 'AdjClose_^GDAXI12' 'AdjClose_^GDAXI13' 'AdjClose_^FTSE1'
 'AdjClose_^FTSE2' 'AdjClose_^FTSE3' 'AdjClose_^FTSE4' 'AdjClose_^FTSE5'
 'AdjClose_^FTSE6' 'AdjClose_^FTSE7' 'AdjClose_^FTSE8' 'AdjClose_^FTSE9'
 'AdjClose_^FTSE10' 'AdjClose_^FTSE11' 'AdjClose_^FTSE12'
 'AdjClose_^FTSE13' 'AdjClose_^N2251' 'AdjClose_^N2252' 'AdjClose_^N2253'
 'AdjClose_^N2254' 'AdjClose_^N2255' 'AdjClose_^N2256' 'AdjClose_^N2257'
 'AdjClose_^N2258' 'AdjClose_^N2259' 'AdjClose_^N22510' 'AdjClose_^N22511
 'AdjClose_^N22512' 'AdjClose_^N22513' 'AdjClose_^HSI1' 'AdjClose_^HSI2'
 'AdjClose_^HSI3' 'AdjClose_^HSI4' 'AdjClose_^HSI5' 'AdjClose_^HSI6'
 'AdjClose_^HSI7' 'AdjClose_^HSI8' 'AdjClose_^HSI9' 'AdjClose_^HSI10'
 'AdjClose_^HSI11' 'AdjClose_^HSI12' 'AdjClose_^HSI13' 'AdjClose_^AXJO1'
 'AdjClose_^AXJO2' 'AdjClose_^AXJO3' 'AdjClose_^AXJO4' 'AdjClose_^AXJO5'
 'AdjClose_^AXJO6' 'AdjClose_^AXJO7' 'AdjClose_^AXJO8' 'AdjClose_^AXJO9'
 'AdjClose_^AXJO10' 'AdjClose_^AXJO11' 'AdjClose_^AXJO12'
 'AdjClose_^AXJO13']
```

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-Nesrest 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.

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):
    """
    Downloads Stock from Yahoo Finance.
    Computes daily Returns based on Adj Close.
    Returns pandas dataframe.
    """
    df = get_data_yahoo(symbol, start, end)

    df.rename(columns = {'Adj Close' : 'AdjClose' + '_' + 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.

```
## merge stock prices and indices and generate past N_history day records as features
## -----------------------------------------------

meg_data = pd.concat(raw_data, axis=1, join='inner')
print volume_data.tail()
print meg_data.tail()
for i in range(len(raw_data)):
    for j in range(N_history):
        t = forcast * (j + 1);
        meg_data[meg_data.columns.values[i] + str(t)] = meg_data.iloc[:, i].shift(t)
```

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 = 2, it means the model is to predict stock trend the day after tomorrow.

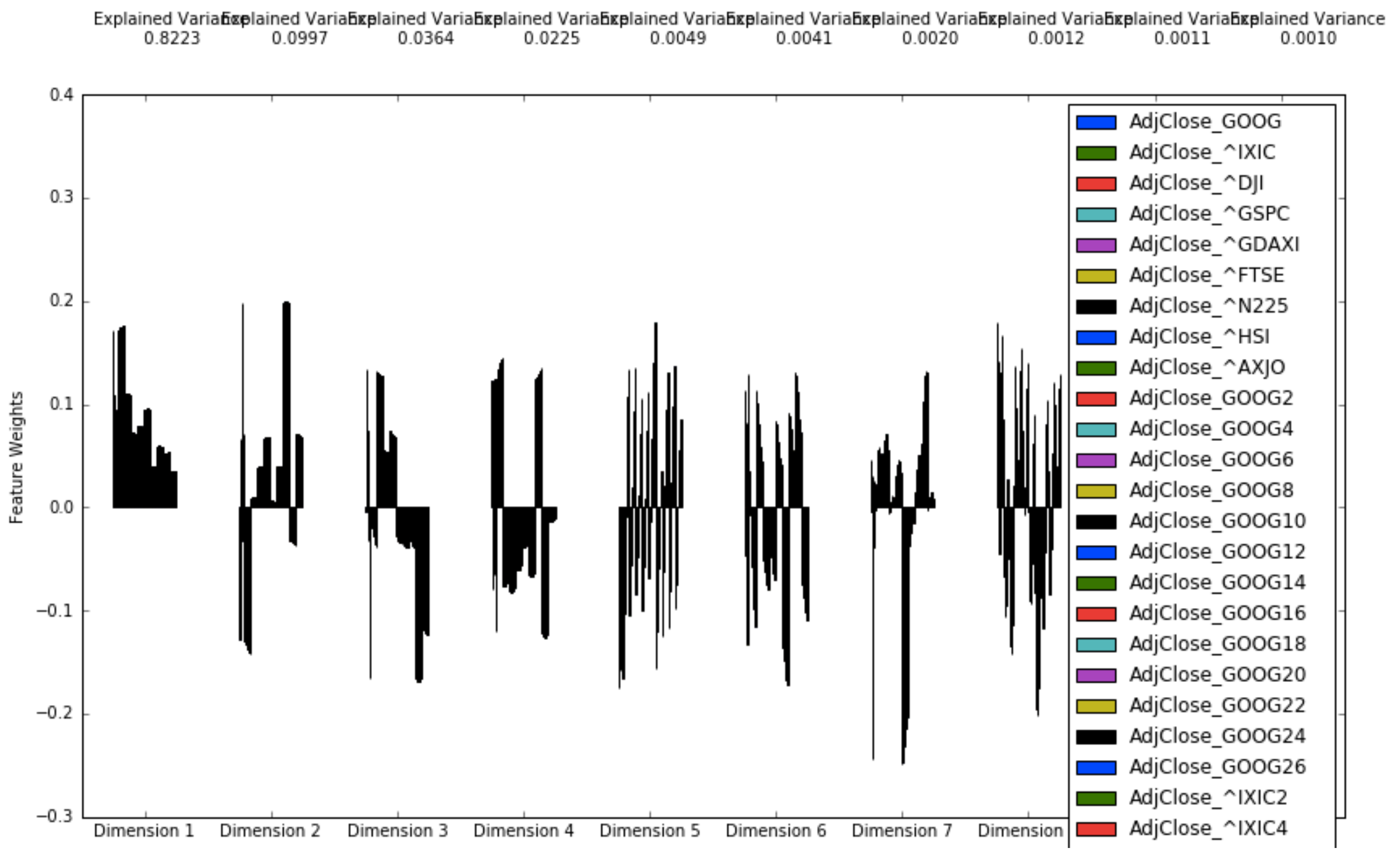
N_history: the number of days you observe in the history to capture data recent dynamics. For example when forecast = 1 and N_history = 13 and suppose the observed day is day i, it means day i-1, i-2,…,i-13 trading record (adj.

close price for company, price of indices) are added to the meg_data. If forecast = 2 and N history = 13, it means day i-2, i-4,…,i-26 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 14 * (1+8) columns. It added N_history=13 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:

```
## PCA
## -----------------------------------------------

# TODO: Scale the data using the natural logarithm
meg_data = meg_data.dropna()
log_data = np.log(meg_data)
#print "log data: ", log_data

# TODO: Apply PCA by fitting the good data with the same number of dimensions as features
pca = PCA(n_components = PCA_n)
pca.fit(log_data)
print "PCA explained variance ratio = ", pca.explained_variance_ratio_

# Generate PCA results plot
#pca_results = rs.pca_results(log_data, pca)

# TODO: Transform the data using the PCA fit above
reduced_data = pca.transform(log_data)
reduced_data = pd.DataFrame(reduced_data, index=log_data.index)
#print reduced_data.shape, reduced_data
```

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 includes PCA_n = 10 dimensions in order to preserve the information. Note the 10th dimension has the explained variance to be 0.001. 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.

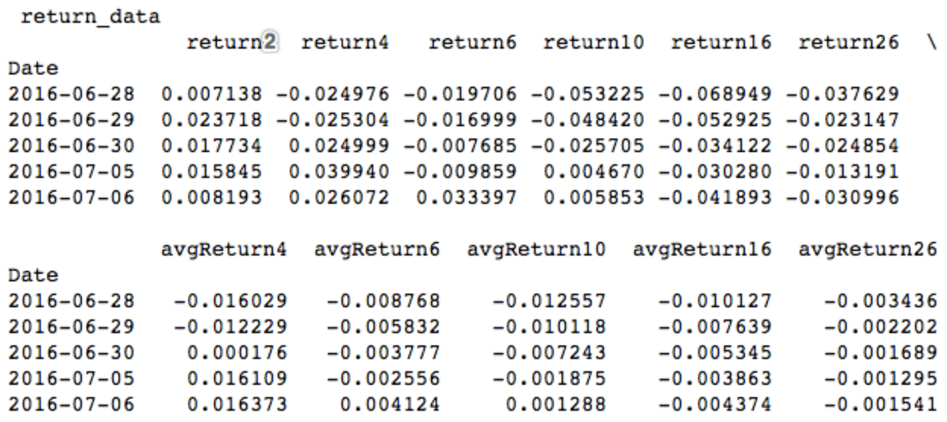
$$return_{i,\boldsymbol{\Delta}} = \frac{adj.close_i - adj.close_{i-\boldsymbol{\Delta}}}{adj.close_{i-\boldsymbol{\Delta}}}$$

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 to day i.

For example we can evaluate the recent stock dynamics by include return_j, $\text{reutrn}_{i,\Delta}$, and $\text{avgReturn}_{i,\Delta}$ 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 = 13, 13+13+13=39 more features will be included. How ever 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 = 13 and forecast = 2, the program generates the below return information for Google's stock price:



```
 return_data
              return2  return4   return6  return10  return16  return26  \
Date
2016-06-28  0.007138 -0.024976 -0.019706 -0.053225 -0.068949 -0.037629
2016-06-29  0.023718 -0.025304 -0.016999 -0.048420 -0.052925 -0.023147
2016-06-30  0.017734  0.024999 -0.007685 -0.025705 -0.034122 -0.024854
2016-07-05  0.015845  0.039940 -0.009859  0.004670 -0.030280 -0.013191
2016-07-06  0.008193  0.026072  0.033397  0.005853 -0.041893 -0.030996

            avgReturn4  avgReturn6  avgReturn10  avgReturn16  avgReturn26
Date
2016-06-28   -0.016029   -0.008768    -0.012557    -0.010127    -0.003436
2016-06-29   -0.012229   -0.005832    -0.010118    -0.007639    -0.002202
2016-06-30    0.000176   -0.003777    -0.007243    -0.005345    -0.001689
2016-07-05    0.016109   -0.002556    -0.001875    -0.003863    -0.001295
2016-07-06    0.016373    0.004124     0.001288    -0.004374    -0.001541
```

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:

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

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

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

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 'return2' 'return4' 'return6' 'return10' 'return16' 'return26' 'avgReturn4' 'avgReturn6' 'avgReturn10' ' avgReturn16' 'avgReturn26' '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 $\text{return}_{\text{i+forecast,forecast}}$ by –forecast rows. Note the tailing forecast rows should be dropped, as they do not have corresponding labels. For example the end date is 2016/07/06 and forecast = 5, rows with date 2016/07/06, 2016/07/05, 2016/07/04, 2016/07/03,2016/07/02 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 fa_score(ytrue, ypred, pos_label=1):
    return min(f1_score(ytrue, ypred, pos_label=1), accuracy_score(ytrue, ypred))

def train_classifier(clf, X_train, y_train, gridSearch=False, parameters=None):
    ''' Fits a classifier to the training data. '''

    # Start the clock, train the classifier, then stop the clock
    start = time()

    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(min_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_

    end = time()

    # Print the results
    print "Trained model in {:.4f} seconds".format(end - start)
    return clf
```

The cross-valiadation grid search (GridSearchCV) object is created using the FA scorer, 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.ensemble import AdaBoostClassifier

# TODO: Initialize the three models
clf_Bst = AdaBoostClassifier(n_estimators=20)

# TODO: Execute the 'train_predict' function for each classifier and each training set size
param_Bst = {'n_estimators': [2, 3, 5, 10, 20, 50], 'learning_rate': [0.1, 0.5, 1]}
clf_Bst, f1_Bst = train_predict(clf_Bst, X_train, y_train, X_test, y_test, gridSearch=True, parameters = param_Bst)

plotROC(clf_Bst, X_test, y_test)

# compare best clf
clf_best, score_best = compBestClf(clf_Bst, f1_Bst, clf_best, score_best)
```

An 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.

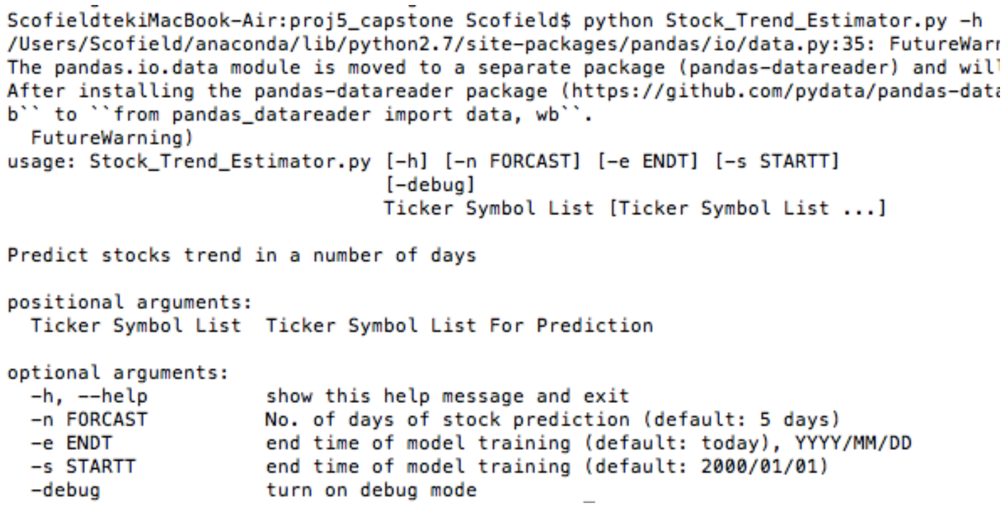
Refinement

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.

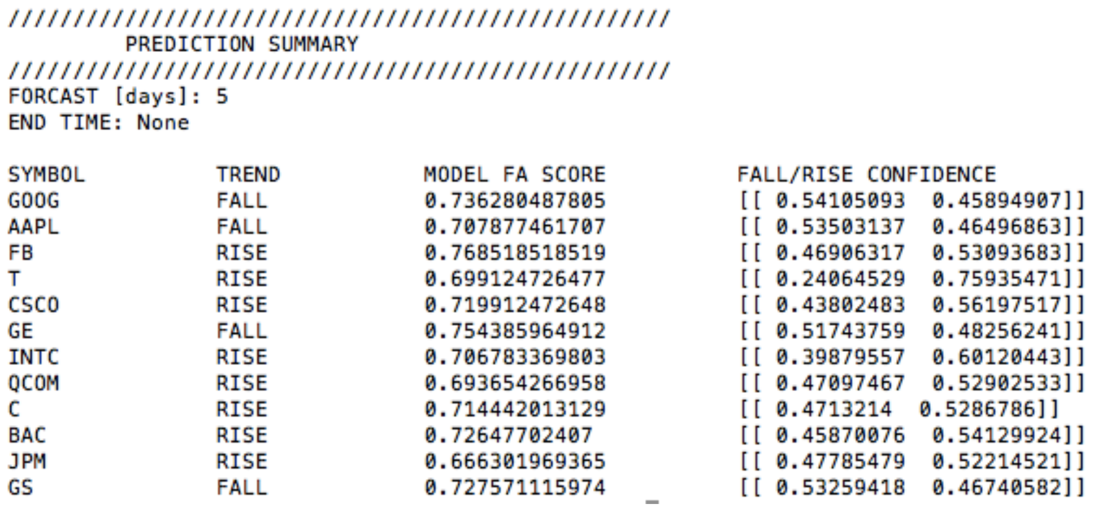
A final solution is to introduce the concept of return and average return of Fibonacci-Series-numbered days to account for the characteristics of history stock trend. Moreover, not only SVM, but also classifier algorithms such as adaptive boosting, random forest, Gaussian naïve Bayes, decision tree, 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.

Moreover, a user-friendly command-line interface is designed:



```
ScofieldtekiMacBook-Air:proj5_capstone Scofield$ python Stock_Trend_Estimator.py -h
/Users/Scofield/anaconda/lib/python2.7/site-packages/pandas/io/data.py:35: FutureWar
The pandas.io.data module is moved to a separate package (pandas-datareader) and wil
After installing the pandas-datareader package (https://github.com/pydata/pandas-dat
b`` to ``from pandas_datareader import data, wb``.
  FutureWarning)
usage: Stock_Trend_Estimator.py [-h] [-n FORCAST] [-e ENDT] [-s STARTT]
                                [-debug]
                                Ticker Symbol List [Ticker Symbol List ...]

Predict stocks trend in a number of days

positional arguments:
  Ticker Symbol List  Ticker Symbol List For Prediction

optional arguments:
  -h, --help          show this help message and exit
  -n FORCAST          No. of days of stock prediction (default: 5 days)
  -e ENDT             end time of model training (default: today), YYYY/MM/DD
  -s STARTT           end time of model training (default: 2000/01/01)
  -debug              turn on debug mode
```

The estimator will train 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]]
```

*Note: “END TIME = None” means today (2016/07/06)

Results

Model Evaluation and Validation

GOOG stock [2000/01/01-2016/07/06], forecast = 2 days						
Data size=2665, test set portion=0.25, feature dimension=22						
Classifiers	AdaBoost	RandomForest	GuassianNB	DecisionTree	SVM	KNN
GridSearch time	6.3s	31s	0.0035s	1.6s	6h	0.4s
Predict time	1.2ms	109ms	0.5ms	0.4ms	34ms	19ms
Train accuracy	0.544	0.874	0.54	0.544	0.64	0.608
Train F1 score	0.702	0.894	0.63	0.702	0.701	0.657
Train FA score	0.544	0.874	0.54	0.544	0.64	0.608
Test accuracy	0.544	0.582	0.52	0.543	0.555	0.538
Test F1 score	0.701	0.677	0.604	0.701	0.63	0.58
Test FA score	0.544	0.582	0.52	0.543	0.555	0.538
FA score variance	<0.001	0.292	0.02	0.001	0.09	0.07
Test ROC AUC	0.52	0.62	0.52	0.52	0.56	0.55
GridSearchCV best params.	n_esti.=2 learning_rate =0.1	n_esti.=80 max_depth=8	N/A	min_split=2 max_depth=2	Poly, deg =2, C=36, gamma=4	N_neighb ors=32

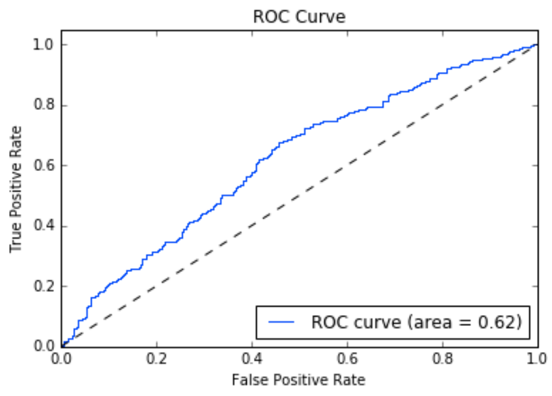
*Note FA score = min{F1, accuracy}

The above 6 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). 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=80, n_jobs=-1,
            oob_score=False, random_state=None, verbose=0,
            warm_start=False)
Best Classifier's Score on test:  0.584707646177
output labels belong to :  [-1  1]
ROC AUC: 0.62
```



```
Predict:
GOOG stock after 2 days of 2016-07-06 00:00:00
is going to (with confidence FALL/RISE = [[ 0.45574552  0.54425448]] ): 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 0.62 AUC, which is a reasonable number for stock prediction and proves the quality of the model.

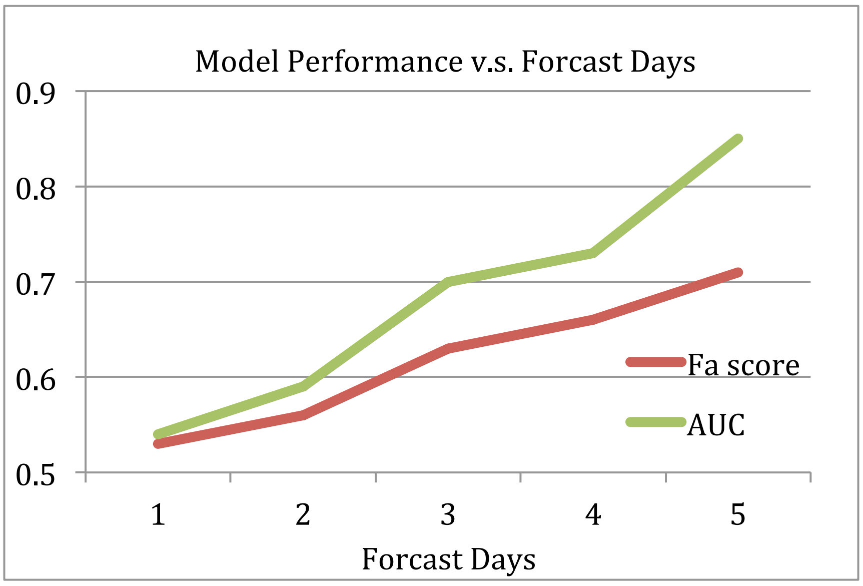
Justification

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 = 0.58 among all the classifiers, which involves accuracy score = 0.58 and F1 score = 0.68. The 58% accuracy on the validation set has 8% 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:



Model Performance v.s. Forcast Days

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 validation 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 regressor 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.

Through out 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 a regressor. 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.