Analysis on Prediction of Stock Change

—Practical Applications of k-NN, SVM, ARIMA Model

Abstract

Traditional regression method will have limitations on stock market prediction, Data Mining algorithms are now under more applications in finding relevant factors and making predictions on stock market data. In this report, three data mining methods of k-NN, SVM, ARIMA are applied to predict the change of stock price, a sample of 128 Australian stocks data in year 2018 is used for practical analysis, and the result shows:

1) k-NN has a high accuracy of prediction when training dataset and testing dataset are in the same steady change period.
2) SVM has higher robustness working on large dataset, it has a relatively high accuracy but is hard to find short-term trend.
3) ARIMA can work well if there exist seasonal pattern, the uncertain time series dataset will increase uncertainty in predictions of ARIMA model.

Table of Contents

Introduction

Data Mining Applications in Stock Market Prediction

Prediction on financial stock market has always been a fascinating and challengeable research area. Most traditional predicting methods can be included into Fundamental Analysis which focus more on a company's financial statements and Technical Analysis which make analysis based on past and current market data.

Data Mining is an exploring and analyzing process which aims at discovering meaningful patterns and associated rules beyond massive data by applying the principal algorithms and suitable techniques. Due to the nature of high uncertainty of stock market and limitations on some traditional predicting methods (for example, traditional regression methods assume all sample points identically independent distributed, which are not appropriate in stock market predicting as data are correlated), more and more models established by data mining algorithms are under application of prediction on financial stock market .

In this report, three data mining algorithms (k-NN, SVM, ARIMA) will be discussed and applied on predicting the price change of stock. We will focus on the question that how well the change of stock market can be predicted by these models, and give relevant discussions and recommendations based on comparison of these different models.

Research Object and Data Collection

Research object of this report are the individual stocks in stock market. We selected (not randomly) 128 Australian stocks as sample, derived their daily stock prices and volumes over the period 1 January 2018 to 31 December 2018.[1]

There are 20 attributes in total in the dataset, along which *Date, Open, Close, High, Low and Volume* are directly obtained and transformed with a *yyyymmdd* format for the *Date*, decimal numbers for the prices in $AU, and a non-negative integer for the *Volume.* And the rest 14 attributes are:

- *Code* : Abbreviated character of code for each 128 stock

[1] Data was derived by screen-scraping the Wall Street Journal stock prices at quotes.wsj.com. For example for Australian company BHP, this page provides most of the data: http://quotes.wsj.com/AU/XASX/BHP/historical-prices.

- *Sector, SubSector* : Sector and Subsector categories for each stock[2]
- *Weekday, Month* : Character strings of weekday and month from *Date*
- *DayofMonth, Year, WeekofYear, DayofYear* : Numbers of day of month, year, week of year, day of year from *Date*
- *Close-Open* : *Close – Open*
- *Change* : "up" if *Close-Open* is zero or positive, "down" otherwise
- *High-Low* : *High - Low*
- *HMLOL* : *(High – Low) / Low*
- *PriorClose* : The *Close* value of the stock on the Date prior to this Date. On the earliest day for each stock, the *PriorClose* is set instead to the value of *Close* from the same day.

Here is a brief summary of this dataset,

- Total 31403 observations, observations on each stock code no larger than 253,
- 8 different sectors and 19 different subsetors, sector *Technology* has most observations of 7821 and subsector *Mining_&_Metals* has most observations of 5930,
- Observations on different weekdays are not equal, observations from Monday to Friday are : 6201, 6331, 6213, 6456, 6202,
- *Open, Close, High, Low, Close-Open, High-Low, Volume, HMLOL, PriorClose* are all right skewed,

This dataset is well-structured and has high quality with no missing values. Considering it was derived by screen-scraping, it is rational to assume there is no manual input error and all data are valid. Thus we will use this dataset for our practical analysis.

Methods[3]

K-Nearest Neighbourhood (k-NN) Classifier

Prediction of stock market change can be divided into two parts: a). predict whether the stock will go up or down, b). predict how much the price value will change. If we focus on a), then the question will become a binary classification problem, so we can apply classification algorithms in data mining to make predictions, that is, based on given training dataset $\mathrm{T} = \{(X_1, y_1), (X_2, y_2), \cdots, (X_l, y_l)\} \in (X \times Y)^l$, *where* $X_i \in R^n$, $y_i \in \{1, -1\}$, find a

[2] Information was taken from the Wall Street Journal web site, from http://quotes.wsj.com/companylist.

[3] All methods are performed with R, the software is RStudio, running at Windows 7, see more technique details in appendix.

function $g(x)$ in $X_i \in R^n$ to obtain decision function,

$$f(x) = \text{sign}(\text{g(x)})$$

to infer y value of any X_i.

K-Nearest Neighbourhood (k-NN) classifier is a non-parametric lazy-learner classification method which are based on learning by analogy. KNN classifier searches k nearest neighbours of the unknown tuple, and assign the unknown tuple to the majority in its k nearest neighbours (assign mean value if doing real-valued prediction). Here, neighbourhood will be measured by Minkowski distance $Dist(X_1, X_2) = \sqrt[P]{\sum_{i=1}^{n}(x_{1i} - x_{2i})^p}$ (usually p=2 for Euclidean distance, n is the dimensions of attributes), and if x_{1i}, x_{2i} are categorical variables, we can assume $x_{1i} - x_{2i} = 0$ if $x_{1i} = x_{2i}$ and $x_{1i} - x_{2i} = 1$ otherwise. For example, set k=3, p=2, the k nearest tuples of test tuple

	Code	SubSector	Weekday	Open	High	Low	Close	Volume	Close.Open	High.Low	HMLOL	PriorClose
1	3DP	Internet/Online	Tuesday	0.145	0.145	0.14	0.14	840117	-0.005	0.005	0.0357143	0.14

are

	Code	SubSector	Weekday	Open	High	Low	Close	Volume	Close.Open	High.Low	HMLOL	distance
29812	WMC	Computer_Services	Friday	0.042	0.042	0.035	0.038	840000	-0.004	0.007	0.200	117.0131
21409	MEU	Mining_&_Metals	Wednesday	0.017	0.018	0.016	0.016	840000	-0.001	0.002	0.125	117.0131
16556	ENB	Electric/Gas_Utilities	Tuesday	0.006	0.006	0.006	0.006	840363	0.000	0.000	0.000	246.0042

KNN can be very slow as it need to calculate and compare every training tuple, time complexity is $O(N)$ if we have N training tuples in finding k-nearest neighbours for one test tuple, and if we have M test tuples, it grows to $O(NM)$, which can be an extreme long period for real prediction, so reducing the training size is applied in practical analysis when fitting the model in R.

So here is the algorithm,

1) Data pre-process, choose all data of subsector *Investing/ Securities_Companies* to reduce data size,
2) Data pre-process, select categorical variables *Weekday, Month,* and continuous variables *Open, High, Low, Close, Volume, PriorClose, HMLOL,*
3) Data pre-process, z-score scale of continuous variables *Open, High, Low, Close, Volume, PriorClose, HMLOL* for standardization,
4) Divide dataset into 5 parts *D1, D2, D3, D4, D5*, every time take *Di* as test dataset and rest as training dataset, do 5-fold cross validation to check mean accuracy.
5) For each tuple in test dataset, get its nearest k neighbours with smallest $Dist(X_1, X_2) = \sqrt[2]{\sum_{i=1}^{n}(x_{1i} - x_{2i})^2}$ (set each attribute weight to

1), if there are more neighbours have same distance with kth one, then include these tuples as well. Assign the majority of Change in these neighbours, if the numbers of "up" and "down" are the same, assign the predicted value to "up", change different k and obtain model assessment.

Support Vector Machine (SVM) Classifier

Support Vector Machine (SVM) is one of the most successful classification methods to overcome "dimension disasters" and over-fitting. Tay and Cao (2002) proved that 5 financial time series data can be predicted by SVM, and has a better performance in standard MSE, standard absolute error, trend accuracy than artificial neural networks.[4]

For N-dimensional training tuple case, let $X = (x_1, x_2, \cdots, x_n)$, if mapping is not required, then $y_i(w_0 + \mathrm{W}X^T) \geq 1 \; \forall i, \; where \; W = \{w_1, w_2, \cdots, w_n\}$ is a weight vector and $w_0 \; a \; scalar \; (bias) \; and \; y_i \in \{-1, +1\}$, and hereby get decision function $d(X) = \sum_{i=1}^{l} y_i \alpha_i X_i X^T + b_0$ where *l* is the number of support vectors and α and b_0 are automatically determined by the optimisation algorithm.

Stock data will first be applied a nonlinear mapping to a sufficiently high dimension $\phi: x \to f$, since optimization function and classification function all involve inner product operation $X_i X^T$ in sample space, $\langle \phi(x_i) \cdot \phi(x_j) \rangle$ will also be required in the mapping high dimensional eigenspace. From Mercer's theorem,$\langle \phi(x_i) \cdot \phi(x_j) \rangle = k(x_i, x_j)$, which means, by applying suitable kernel functions, the decision function can be converted to $d(X) = \sum_{i=1}^{l} y_i \alpha_i k(x_i, x_j) + b_0$.

Time complexity of SVM is more based on the length of supporting vector, but not entirely depend on size of training data. For categorical variables, one possible dealing is to assign new 0-1 variables, for example, if $x_i \in \{a, b, c\}$ and $x_1 = a$, then we create $\{L_{11} = 1, L_{12} = 0, L_{13} = 0\}$ for x_1, but it will greatly increase time complexity as more vectors will there be if more categories x_i have, so we only focus on numeric variables in this SVM model.

And here is the algorithm,

[4] Tay, F. E., & Cao, L. J. (2002). Modified support vector machines in financial time series forecasting. Neurocomputing, 48(1-4), 847-861.

1) Data pre-process, select variables *Open, High, Low, Close, Volume, PriorClose, HMLOL* and apply z-score scale for standardization,
2) Holdout of randomly chosen 70% of samples as training datasets, rest 30% as testing datasets, select Gaussian radial function as kernel function, train SVM model from training data, do random subsampling for n times, then get mean accuracy on predicting $y_i \in \{"up", "down"\}$.
3) Model Assessment

Auto-Regressive Integrated Moving Average (ARIMA) Model

As discussed previously, question will change to a regression question if we want to predict value change, traditional regression methods like SLR or logistic regression are not well performed as they assume sample points i.i.d.

Time Series Analysis is a useful tool to predict correlated data like stock market data, it is based on the assumption that y_t has correlation with y_{t-1}, y_{t-2}, etc, so we can predict y_t based on former observations.

Autoregressive Integrated Moving Average (ARIMA) model can be used to deal with non-stationary time series data. Its algorithm develops from ARMA model, and an *ARMA(p,q)* model is given by

$$\left(1 - \sum_{i=1}^{p} \alpha_i L^i\right) y_t = \left(1 + \sum_{i=1}^{q} \theta_i L^i\right) \varepsilon_t,$$

where *L* is the lag operator, the α_i are the parameters of the autoregressive part of the model, the θ_i are the parameters of the moving average part and the ε_t are error terms which are generally assumed to be independent, identically distributed variables sampled from $N(0, \sigma^2)$[5]. An *ARIMA(p,d,q)* model can be expressed as

$$\left(1 - \sum_{i=1}^{p'} \phi_i L^i\right) (1 - L)^d y_t = \left(1 + \sum_{i=1}^{q} \theta_i L^i\right) \varepsilon_t,$$

where $p' = p - d$.

The steps of fitting a time series ARIMA model,

1) Data pre-process, randomly select a stock, get its *Close* data as dataset,
2) Visualization to identify stationarity, difference non-stationary time series data into stationary series,
3) Fit ARIMA model, determine p and q using ACF and PACF,

[5] See Wiki for more details, Autoregressive integrated moving average, https://en.wikipedia.org/wiki/Autoregressive_integrated_moving_average

4) Parameter estimation on ARIMA model(usually least square method),
5) Hypothesis testing to identify whether residual series are white noise,
6) predicting

Analysis

Practical Results Analysis[6]

☐ k-NN

By applying k-NN with k=10 and 5-fold cross validations on predicting the change of *Investing/ Securities_Companies* stocks in R, we get 5 confusion matrix shown as below,

Figure 1. Confusion Matrix of k-NN Classifier in 5-fold cross validation

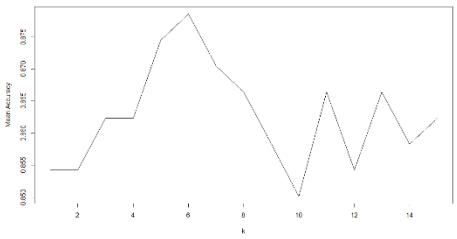
1.	real-up	real-down	2.	real-up	real-down
pre-up	215	25	pre-up	208	35
pre-down	2	5	pre-down	2	2
3.	real-up	real-down	4.	real-up	real-down
pre-up	167	80	pre-up	183	63
pre-down	0	1	pre-down	0	1
5.	real-up	real-down			
pre-up	244	4			
pre-down	0	0			

From this we can get:

1) Mean accuracy is *0.8294*, stand error is *0.1207*,
2) Mean precision is *0.9963*, mean recall is *0.8311*,
3) Confusion Matrix 3 and 4 have bad performances on predicting change when there exist many real "down" circumstances, considering down: up = 215: 1018, up/(down + up)= 0.8256285 in our dataset, it may be caused by unbalanced design.

Below is how mean accuracy changes as k changes (from 1 to 15), we can see mean accuracy can reach the peak when k is equal to 6.

Figure 2. Mean Accuracy Change with k parameter



[6] All result here can be reproduced as random seed is set, see appendix for more detailed codes.

☐ SVM

We fit a SVM model in R by *SVM* function in the package "e1071", and set the parameters *type* of y_i = C for *character*, *kernel = "radial"* for Gaussian radial function, n=5 for 5 times of random subsampling, then we get 5 confusion matrix shown as below,

Figure 3. Confusion Matrix of SVM Classifier in 5 times random subsampling

1.	real-up	real-down	2.	real-up	real-down
pre-up	6281	2700	pre-up	6276	2500
pre-down	203	237	pre-down	293	352
3.	real-up	real-down	4.	real-up	real-down
pre-up	6247	2361	pre-up	6180	2431
pre-down	388	425	pre-down	361	449
5.	real-up	real-down			
pre-up	6252	2584			
pre-down	250	335			

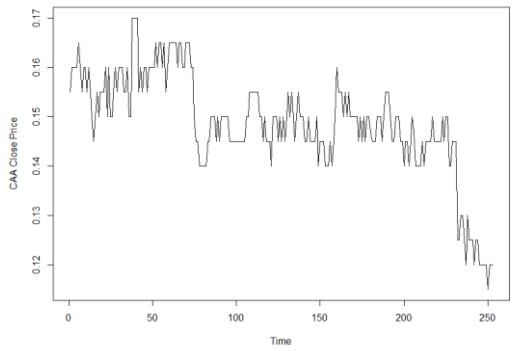
From this we can get:

1) Mean accuracy is *0.7013*, stand error is *0.0062,*
2) Mean precision is *0.9544*,mean recall is *0.7131,*
3) Unlike ordered samples in cross validation, random subsampling eliminates periodic trends, and confusion Matrix are more similar and steady.

☐ ARIMA

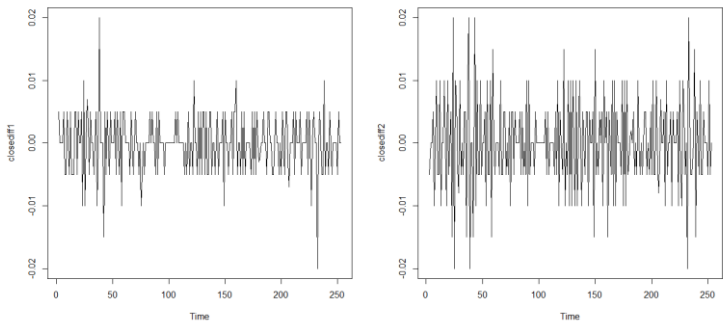
Randomly select a stock as sample data, draw its time series plot, train first 90% of the data as training data,

Figure 4. Time Series Data of "CAA" Close Price



The data is obviously non-stationary, so do the differences:

Figure 5. 1st Difference plot and 2nd difference plot



Here we will choose difference = 1. And fit an ARIMA model with the auto.arima function in forecast package.

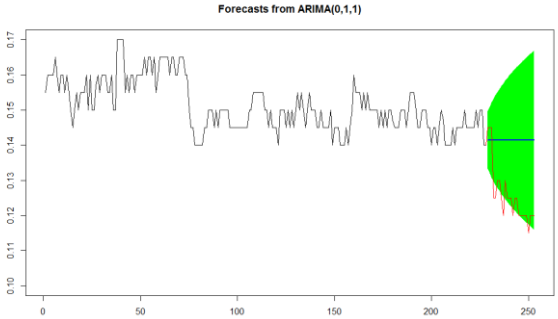
```
Series: Close_data[1:round(0.9 * nrow(Code_data))]
ARIMA(0,1,1)

Coefficients:
          ma1
      -0.3950
s.e.   0.0667

sigma^2 estimated as 1.726e-05:  log likelihood=923.09
AIC=-1842.17   AICc=-1842.12   BIC=-1835.32
```

So our model is ARIMA(0,1,1), then apply this model to make prediction on the rest 10% of data:

Figure 6. Predictions of ARIMA(0,1,1) Model



We can find the performance is not as good as two former classifiers because this model didn't predict short-term change but only give prediction intervals (green shaded area for 95% PI), although some real values are included in this interval, it will still have no practical meaningful help for real investment strategies.

Model Comparison and Assessment

From analysis in the last part, it can be concluded:

1) K-NN has the largest predicting accuracy among these 3 models, from the nature of algorithm of k-NN, it can attain greatly high accuracy as long as there are enough training datasets, but the time cost will be massive, which make it not realistic in analyzing the full size of real unordered massive stock data.
2) SVM has the most steady prediction outcome, I believe it is because a). larger size of 70% of dataset are used for training, b). random subsampling will unlikely to be influenced by periodic change, which is common in stock market.
3) ARIMA model performs not well because a). it is unable to predict sudden change, b). prediction interval becomes larger and larger as time increases, although this may indicate a relatively high accuracy, but it is a proof of higher uncertainty and not realistic in true stock market investment.

Discussion

Insufficiency and Recommendation

These three models have different strength and disadvantages:

- k-NN predicts well when training size is large but time cost is massive, the improvement of k-NN on stock market predicting in the future can be partial distance (skip a given tuple if distance of any subsets of this tuple is over threshold) and editing (prune tuples that can be proved to be invalid).
- SVM performs well on large dataset, but may have poor generalization ability in some cases of inappropriate training datasets, besides, too large training dataset will cover the short-term trend, not enough training dataset will make it unable to find movement trend. The improvement of SVM on stock market predicting in the future can be a). specialized kernel function on stock market, b). select highly relevant training dataset, for example, the last 50 days of stock market data, c). select balanced dataset.
- ARIMA model will be appropriate when there is obvious seasonal pattern, but as discussed previously, uncertain time series data will cause more uncertainty in prediction, which makes the model hard to find abrupt trends, the improvement of ARIMA on stock market predicting in the future can be finding more time series data to check whether the uncertainty in y_t can be interpreted.

Appendix

Code of K-Nearest Neighbourhood (k-NN) Classifier

```
stock = read.csv("stock_prices_2018.csv",header = T)
attach(stock)

#improved knn

dist_p <- function(p, train_noy, trainline_number, test_noy){
  sum_dist_p <- 0
  for (i in 1:ncol(test_noy)){
    if (is.factor(test_noy[1, i]) == TRUE){
      if (train_noy[trainline_number, i] == test_noy[1, i]){
        sum_dist_p <- sum_dist_p + 0
      }
      else{
        sum_dist_p <- sum_dist_p + 1
      }
    }
    else{
      sum_dist_p <- sum_dist_p + (train_noy[trainline_number, i] - test_noy[1, i])^p
    }
  }
  sum_dist_p^(1/p)
}

find_k_nearest <- function(train, test, k, p){
  k_nearest <- data.frame()
  for (i in 1:nrow(train)){
    if (i <= k){
      k_nearest <- rbind(k_nearest,data.frame(train[i, -ncol(train)], dist_p(p, train[, -ncol(train)],
i, test[, -ncol(test)]), train[i, ncol(train)]))
    }else{
      if (dist_p(p, train[, -ncol(train)], i, test[, -ncol(test)]) == max(k_nearest[, ncol(k_nearest)-
1])){
        k_nearest <- rbind(k_nearest,data.frame(train[i, -ncol(train)], dist_p(p, train[, -
ncol(train)], i, test[, -ncol(test)]), train[i, ncol(train)]))
        }
      if (dist_p(p, train[, -ncol(train)], i, test[, -ncol(test)]) < max(k_nearest[, ncol(k_nearest)-
1])){
        k_nearest <- k_nearest[-which.max(k_nearest[, ncol(k_nearest)-1]), ]
        k_nearest <- rbind(k_nearest,data.frame(train[i, -ncol(train)], dist_p(p, train[, -
ncol(train)], i, test[, -ncol(test)]), train[i, ncol(train)]))
```

```
        }
      }
    }
    colnames(k_nearest)[ncol(k_nearest)-1] <- 'distance'
    colnames(k_nearest)[ncol(k_nearest)] <- 'Change'

    sort_k_nearest <- k_nearest[order(k_nearest$distance),]
    if (k == nrow(sort_k_nearest)){
      return(sort_k_nearest)
    }else{
      for (i in k:(nrow(sort_k_nearest)-1)){
        if (sort_k_nearest[i, ncol(k_nearest)-1] != sort_k_nearest[i+1, ncol(k_nearest)-1]){
          return(sort_k_nearest[1:i, ])
        }
      return(sort_k_nearest)
      }
    }
}

knn <- function(train, test, k, p){
    k_nearest <- find_k_nearest(train, test, k, p)
    if (sum(k_nearest[, ncol(k_nearest)] == "up") < sum(k_nearest[, ncol(k_nearest)] == "down")){
      predict_change <- "down"
    }else{
      predict_change <- "up"
    }
    predict_change
}

set.seed(8410)
sample_sector <- sample(unique(SubSector), 1, replace = F)
selected_lines <- which(SubSector == sample_sector)

#5 fold cross validation
for (i in (1:5)){
    scale_stock_contin <- scale(data.frame(Open, High, Low, Close, Volume, PriorClose, HMLOL))

    testline <- selected_lines[round((i-
1)/5*length(selected_lines)):round((i)/5*length(selected_lines))]
    trainline <- selected_lines[-match(testline, selected_lines)]

    knn_train_subsector <- data.frame(Weekday, Month, scale_stock_contin, Change)[trainline, ]
    knn_test_subsector <- data.frame(Weekday, Month, scale_stock_contin, Change)[testline, ]
```

```
    seq_up_down <- c(0,0,0,0)
    for (j in(1:nrow(knn_test_subsector))){
      predict_change <- knn(train = knn_train_subsector, test=knn_test_subsector[j, ], k=10, p=2)

      if (predict_change == knn_test_subsector[j, ncol(knn_test_subsector)]){
        if (predict_change == "up"){
          seq_up_down[1] <- seq_up_down[1]+1
        }
        if (predict_change == "down"){
          seq_up_down[4] <- seq_up_down[4]+1
        }
      }else{
        if (predict_change == "up"){
          seq_up_down[3] <- seq_up_down[3]+1
        }
        if (predict_change == "down"){
          seq_up_down[2] <- seq_up_down[2]+1
        }
      }
    }
    #confusion matrix
    result = matrix(seq_up_down,c(2,2))
    colnames(result) <- c("real-up","real-down")
    rownames(result) <- c("pre-up","pre-down")
    print(result)
}
```

Code of Support Vector Machine (SVM) Classifier

```
library(e1071)

n=5
set.seed(1)
for (i in 1:n){
    seq_accuracy = c(0,0,0,0)
    scale_stock_contin <- scale(data.frame(Open, High, Low, Close, Volume, PriorClose, HMLOL))
    trainline <- sample(nrow(stock), round(0.7*nrow(stock)), replace = F)

    train <- data.frame(scale_stock_contin, Change)[trainline, ]
    test <- data.frame(scale_stock_contin, Change)[-trainline, ]

    stock_svm <- svm(Change ~ Open+High+Low+Close+Volume+PriorClose+HMLOL, data =
train, type = 'C',kernel = 'radial')
    pre_change <- predict(stock_svm, newdata = test[ , -ncol(test)])
```

```
    accuray_ <- data.frame(pre_change, Change[-trainline])

    seq_accuracy[1] <- length(which(accuray_[ ,1] == 'up' & accuray_[,2] == 'up'))
    seq_accuracy[2] <- length(which(accuray_[ ,1] == 'down' & accuray_[,2] == 'up'))
    seq_accuracy[3] <- length(which(accuray_[ ,1] == 'up' & accuray_[,2] == 'down'))
    seq_accuracy[4] <- length(which(accuray_[ ,1] == 'down' & accuray_[,2] == 'down'))

    result = matrix(seq_accuracy,c(2,2))
    colnames(result) <- c("real-up","real-down")
    rownames(result) <- c("pre-up","pre-down")
    print(result)
}
```

Code of Auto-Regressive Integrated Moving Average (ARIMA)

```
set.seed(1)
Code_sample <- sample(unique(Code),1)
Code_data <- stock[which(Code == Code_sample), ]

Close_data <- ts(Code_data$Close,start=1)
plot.ts(Close_data,ylab = "CAA Close Price")

par(mfrow = c(1,2))
closediff1 <- diff(Close_data,differences=1)
plot.ts(closediff1)

closediff2 <- diff(Close_data,differences=2)
plot.ts(closediff2)

acf(closediff2,lag.max=20)
acf(closediff2,lag.max=20,plot=FALSE)

pacf(closediff2,lag.max=20)
pacf(closediff2,lag.max=20,plot=FALSE)

closearima <- arima(Close_data[1:round(0.9*nrow(Code_data))],order=c(4,2,1))
closearima

closearimaauto <- auto.arima(Close_data[1:round(0.9*nrow(Code_data))])
closearimaauto

library(forecast)
closeforecast <- forecast(object = closearimaauto,h = round(0.1*nrow(Code_data)),level=c(95))
closeforecast
```

```
par(mfrow = c(1,1))
plot(closeforecast,shadecols = "green",ylim = c(0.1,0.17))
lines(c(round(0.9*nrow(Code_data)):nrow(Code_data)),Close_data[round(0.9*nrow(Code_data)):
nrow(Code_data)], col ="red")

tsdiag(closearima)
acf(closearima$residuals, lag.max = 20)
Box.test(closeforecast$residuals, lag=20, type = "Ljung-Box")

plot.ts(closeforecast$residuals)
```