Stock Price Prediction

Xiaochun Fan

2024-08-27

Introduction

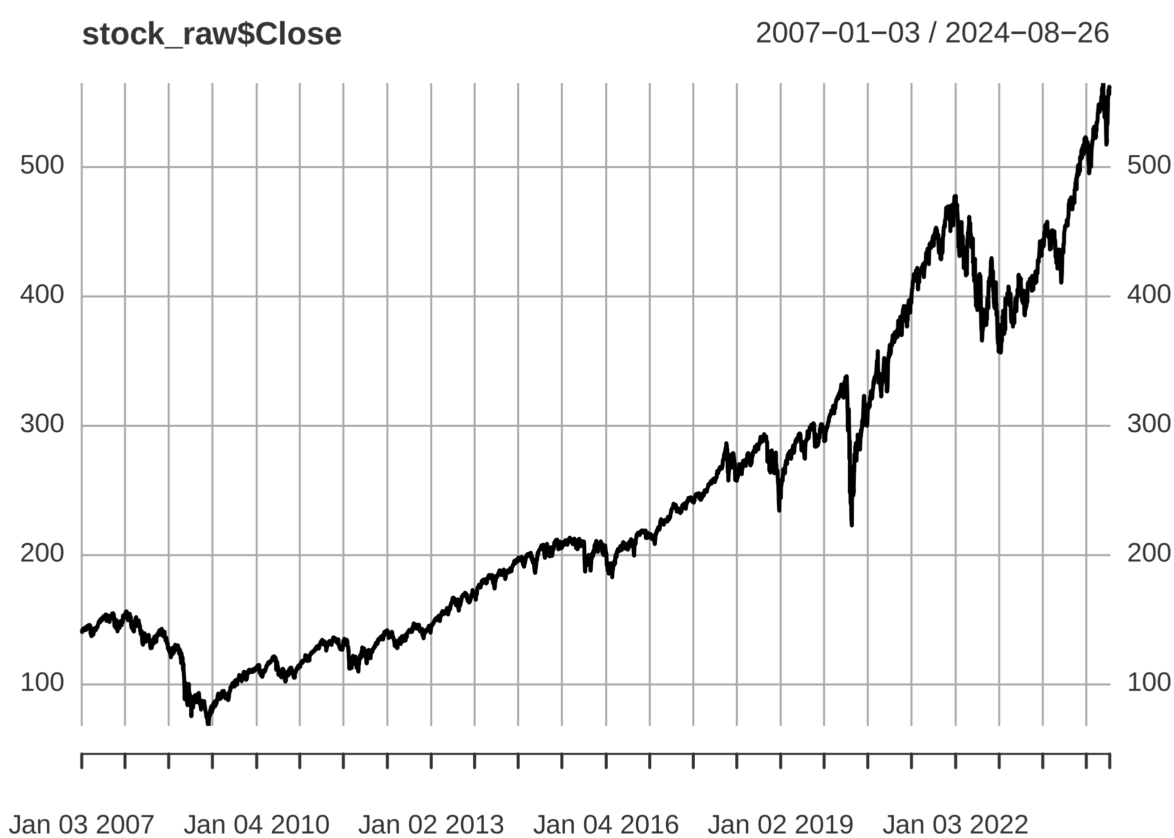
This project will be attempting to predict stock price moving direction based on historical stock data. Historic data will be gathered from Yahoo Finance. Stock pricing data is extremly noisy and not an easy task to predict, therefore this project will just focus on general methods instead of results.

Step 1: Load and Transform Data

Stock pricing data can be loaded directly from Yahoo Finance using 'getSymbols' function from 'tidyquant' package.

```
# loading stock pricing data
stock_raw <- getSymbols(Symbols = 'SPY', auto.assign = FALSE) %>%
  `colnames<-`(c('Open', 'High', 'Low', 'Close', 'Volume', 'Adjusted_Close'))
```

We can plot historic SPY price data:



This project will be focusing on predicting directions, a binary value of 1 or 0, with 1 meaning stock moves up. Therefore, the absolute value of the stock price isn't that important, it's the relationship between pricing data that we will be focusing on.

The following code create relationships between pricing changes and create prediction targets.

```
# Compute price changes and create prediction target (y)
stock <- stock_raw %>%
  data.frame() %>%
  mutate(change = Close - Open, change_pct = change * 100 / Open,
         high_pct = (High - Open) * 100 / Open,
         low_pct = (Low - Open) * 100 / Open) %>%
  mutate(direction = ifelse(change_pct > 0, 1, 0), y = lead(direction),
         y = factor(y)) %>%
  mutate(direction = NULL) # remove the known "direction" column.
```

Here's a snippet of the transformed data:

```
head(stock)
```

```
##              Open   High    Low  Close   Volume Adjusted_Close     change
## 2007-01-03 142.25 142.86 140.57 141.37 94807600       101.2101 -0.8800049
## 2007-01-04 141.23 142.05 140.61 141.67 69620600       101.4248  0.4400024
## 2007-01-05 141.33 141.40 140.38 140.54 76645300       100.6159 -0.7900085
## 2007-01-08 140.82 141.41 140.25 141.19 71655000       101.0812  0.3699951
## 2007-01-09 141.31 141.60 140.40 141.07 75680100       100.9953 -0.2399902
## 2007-01-10 140.58 141.57 140.30 141.54 72428000       101.3317  0.9599915
##            change_pct   high_pct    low_pct y
## 2007-01-03 -0.6186326 0.42882292 -1.1810142 1
## 2007-01-04  0.3115503 0.58061839 -0.4389968 0
## 2007-01-05 -0.5589815 0.04952386 -0.6721835 1
## 2007-01-08  0.2627433 0.41897196 -0.4047772 0
## 2007-01-09 -0.1698325 0.20522861 -0.6439768 1
## 2007-01-10  0.6828791 0.70422925 -0.1991740 1
```

Step 2: Spliting Data

Since stock pricing is a time-series data, we want to split the data into training and testing sets before creating more predictors. A lot of the predictors will be based on averaging certain period of past data, therefore if predictors are created before spliting data, it's possible that information from training data could leak into testing data due to averaging.

The code below splits data into training set for model training, ensemble_test set for ensemble models and select the best ensembling method, and final_test set for final evaluation.

```
# split data before creating predictors
train <- stock[1:3000, ]
ensemble_test <- stock[3001:4000, ]
final_test <- stock[4001:nrow(stock), ]
```

Step 3: Generator Predictors

Directly using daily stock data as predictors probably won't be very helpful, since it lacks relationship to past data. Therefore in this step, we will generate predictors based on rolling averages and recent high/lows of historic data from certain time periods.

Next, we remove the columns containing absolute values of the stock pricing data, and only leave relavent pricing data as predictors.

Here's a snippet of the processed data:

```r
head(train)
```

```
##                 change change_pct   high_pct    low_pct y      sma_10    ema_10
## 2007-04-27  0.4400024  0.2951254 0.43598442 -0.1676840 0  0.90617391 0.9684440
## 2007-04-30 -1.3500061 -0.9021693 0.06683113 -0.9556219 1 -0.02967381 0.1148229
## 2007-05-01  0.2500000  0.1684409 0.70745389 -0.5053227 1  0.11972920 0.3028355
## 2007-05-02  0.6399994  0.4298183 0.70517333 -0.1007347 1  0.54901542 0.7223356
## 2007-05-03  0.3800049  0.2533873 0.28671913 -0.1600357 1  0.87729255 1.0286144
## 2007-05-04  0.1699982  0.1127683 0.24543623 -0.3515746 1  1.09925867 1.1474246
##              evwma_10       high_10     low_10      vol_10     sd_10   sma_20
## 2007-04-27 1.8426635 -0.0018056863 0.02481102  0.07202247 0.2667440 2.335985
## 2007-04-30 0.9209878 -0.0101828163 0.01301499 -0.01294445 0.4243741 1.307231
## 2007-05-01 1.0236711 -0.0076007594 0.01553775  0.22011602 0.4248587 1.340552
## 2007-05-02 1.4663254 -0.0027417660 0.02126517 -0.20108356 0.4224560 1.718934
## 2007-05-03 1.8293379 -0.0003324762 0.02201538 -0.18993562 0.4075754 2.032263
## 2007-05-04 1.9885088 -0.0013251852 0.02385364 -0.03975625 0.4026241 2.180956
##              ema_20 evwma_20       high_20     low_20       vol_20     sd_20
## 2007-04-27 1.995908 3.157798 -0.0018056863 0.05998797  0.159538123 0.3131756
## 2007-04-30 1.064357 2.215776 -0.0101828163 0.04592351  0.112116044 0.3905420
## 2007-05-01 1.191788 2.286294 -0.0076007594 0.03874349  0.312870666 0.3889178
## 2007-05-02 1.598384 2.720389 -0.0027417660 0.04266410 -0.062167135 0.3858228
## 2007-05-03 1.925808 3.094565 -0.0003324762 0.04662461 -0.082077621 0.3862619
## 2007-05-04 2.077526 3.287070 -0.0013251852 0.05022530  0.002640366 0.3824272
##              sma_40   ema_40 evwma_40       high_40     low_40      vol_40
## 2007-04-27 4.117233 3.067948 4.675397 -0.0018056863 0.08546779 -0.01577015
## 2007-04-30 3.153276 2.147282 3.788026 -0.0101828163 0.07782045 -0.07415870
## 2007-05-01 3.210465 2.280451 3.908638 -0.0076007594 0.08017756  0.19519557
## 2007-05-02 3.609064 2.709990 4.376486 -0.0027417660 0.08552891 -0.22477628
## 2007-05-03 3.948956 3.076379 4.791784 -0.0003324762 0.09045564 -0.22444842
## 2007-05-04 4.143088 3.274514 5.033637 -0.0013251852 0.09389079 -0.09391309
##               sd_40   sma_80   ema_80    evwma_80       high_80     low_80
## 2007-04-27 0.5169080 4.124172 4.124172  0.00000000 -0.0018056863 0.08546779
## 2007-04-30 0.5125861 3.264123 3.240419 -0.82486504 -0.0101828163 0.07782045
## 2007-05-01 0.5056599 3.452528 3.401622 -0.55700060 -0.0076007594 0.08017756
## 2007-05-02 0.5004985 3.938992 3.865746  0.02769594 -0.0027417660 0.08552891
## 2007-05-03 0.5005477 4.380366 4.275432  0.55978745 -0.0003324762 0.09045564
## 2007-05-04 0.5005295 4.659918 4.522470  0.92340848 -0.0013251852 0.09389079
##                 vol_80     sd_80
## 2007-04-27  0.14114943 0.5963491
## 2007-04-30  0.07810014 0.6011853
## 2007-05-01  0.30174959 0.6005937
## 2007-05-02 -0.07811451 0.5982210
## 2007-05-03 -0.08724340 0.5981808
## 2007-05-04  0.02103075 0.5976698
```

Step 4: Model Training

This step will train some models using training dataset.

This step will take some time to run, depends on computer performance.

This optional code below can use multiple cores to speed up training process:

```
# setup parallel computation for multi-core computers
if (detectCores() > 1){
  num_core <- detectCores() - 1
  pl <- makeCluster(num_core)
  registerDoParallel(pl)
} else {
  registerDoSEQ()
}
```

Train some models:

```
# setup training control
control <- trainControl(method = 'timeslice', initialWindow = 500,
                        horizon = 300, fixedWindow = FALSE, skip = 49)

# train some models
fit <- list()
fit[['knn']] <- train(y ~ ., data = train, trControl = control, method = 'knn',
                      tuneGrid = data.frame(k = seq(1, 15, 2)))
fit[['rf']] <- train(y ~ ., data = train, trControl = control, method = 'rf',
                     tuneGrid = data.frame(mtry = seq(2, ncol(train), 2)))
fit[['glm']] <- train(y ~ ., data = train, trControl = control, method = 'glm')
fit[['rpart']] <- train(y ~ ., data = train, trControl = control, method = 'rpart',
                        tuneGrid = data.frame(cp = seq(0.005, 0.05, 0.005)))
fit[['xgbTree']] <- train(y ~ ., data = train, trControl = control, method = 'xgbTree')
fit[['earth']] <- train(y ~ ., data = train, trControl = control, method = 'earth')
```

Step 5: Ensemble Models

This step will compare prediction results from 6 models and using the majority to decide which prediction to be used.

This code below creates a function that takes in the number of models to use, and output the prediction results.

For accuracy metrics, we should focus more on precision, instead of overall accuracy. Also, only using accuracy to test performance may not work as intended, because the magnitude of price changes were ignored.

Therefore, this function also calculates the dollar amount the stock has changed during the testing data period, and compared to the predicted result ('portfolio' vs 'buy_and_hold' metrics).

```
result <- function(x, test_data){
  pred <- list()
  for (f in fit){
    pred[[f$method]] <- predict(f, test_data)
  }

  ensemble <- pred %>%
    data.frame() %>%
    mutate(count = rowSums(. == 1)) %>%
    mutate(y_hat = ifelse(count >= x, 1, 0),
           y_hat = lag(y_hat),
           y_hat = factor(y_hat, levels = levels(test_data$y))) %>%
    na.omit()

  test_data <- test_data %>%
```

```r
    mutate(y = lag(y)) %>%
    na.omit()

  ensemble <- cbind(test_data, ensemble) %>%
    select(y, y_hat, change)
  conf <- confusionMatrix(ensemble$y_hat, ensemble$y, positive = '1')
  ensemble <- ensemble %>%
    mutate(y_hat = as.numeric(as.character(y_hat))) %>%
    summarise(portfolio = sum(ensemble$change * y_hat),
              buy_and_hold = sum(ensemble$change),
              precision = conf$byClass['Precision'],
              sensitivity = conf$byClass['Sensitivity'],
              overall_accu = conf$overall['Accuracy'])

  return(ensemble)
}
```

Next, we will test which ensemble parameter performs best:

```r
ensemble_n <- seq(1, length(fit), 1)
lapply(ensemble_n, function(x) result(x, ensemble_test))
```

```
## [[1]]
##   portfolio buy_and_hold precision sensitivity overall_accu
## 1   29.8203      29.8203 0.5423913           1    0.5423913
##
## [[2]]
##   portfolio buy_and_hold precision sensitivity overall_accu
## 1   29.8203      29.8203 0.5423913           1    0.5423913
##
## [[3]]
##   portfolio buy_and_hold precision sensitivity overall_accu
## 1  31.55031      29.8203 0.5442329   0.9739479    0.5434783
##
## [[4]]
##   portfolio buy_and_hold precision sensitivity overall_accu
## 1  97.46007      29.8203 0.5510471   0.8436874    0.5423913
##
## [[5]]
##   portfolio buy_and_hold precision sensitivity overall_accu
## 1  119.7101      29.8203 0.5589354   0.5891784        0.525
##
## [[6]]
##   portfolio buy_and_hold precision sensitivity overall_accu
## 1  102.5202      29.8203 0.5934959   0.2925852    0.5076087
```

Step 6: Final Testing

After selecting ensemble parameters (5, in this case), we will test it on the final testing dataset, using code below:

```r
# Final testing
result(5, final_test)
```

```
##   portfolio buy_and_hold precision sensitivity overall_accu
## 1  54.23987     62.69995  0.559322   0.6502463    0.5152355
```

Conclusion

It's clear the accuracy is low, and seems very inconsistent (performance from ensemble_test dataset do not translate well to final_test dataset). The prediction method can potentially be improved by selecting / excluding some models, adding more predictors, adding pricing data from other stocks/instruments, and using higher resolution data.