Predicting Price Movements of Publicly Traded Securities

A Comparative Analysis of Conventional Statistical Modeling and Supervised Machine Learning

Raoul R. Wadhwa Dahwi Kim

September 25, 2020

Contents

1 Introduction

1.1 Statistics in Finance

The stock market constantly provides a large quantity of time-series pricing data every day for many companies. It is only natural that analysis of these time series has caught the attention of mathematicians, statisticians, and computer scientists worldwide. Efforts to predict price movements in the stock market have also benefited from the inherent financial incentive available to those able to successfully do so. However, prior to the early 1900s, efforts to predict price movements were limited to financial *speculation*, rather than true financial *security analysis*. Whereas security analysis is a valid discipline and an active field of research, financial speculation resembles gambling too closely to be considered financial analysis. [2]

In the 20th century, stock market analysis was split into two fields: security analysis, encapsulated by Ben Graham's The Intelligent Investor; and technical analysis, encapsulated by John Murphy's Technical Analysis of the Financial Markets. [1, 3] In both camps, the mathematics used is limited to elementary algebra and arithmetic, with some application of differentiation. However, the invention and widespread usage of the computer led to the creation of a third field within financial analysis, namely quantitative finance. This field ignores the business aspects underlying financial securities, and chooses instead to view time-series of prices as trajectories of a predictable process (e.g. Markov chain). High-frequency trading uses statistical analysis of price trajectories in the very short-term (on the order of milliseconds) to generate large profits from small discrepancies in price across stock indexes by increasing trading volume across the same period of time. This strategy famously resulted in the flash crash where the price of a financial security varied hundreds of dollars in a single hour.

Although quantitative finance is highly profitable, it is only a viable strategy for large firms with sufficient cash reserves to overcome the fixed costs of trading commissions, that add up to large values due to trading frequency (potentially hundreds every second). In this paper, we explore the potential application of either conventional statistics or machine learning algorithms to a longer term holding period (on the order of days, instead of milliseconds), and conduct a comparative analysis to see which paradigm, if either, predicts short-term price movements more accurately.

1.2 Logistic Regression and Single-Layer Perceptron

Logistic regression is an established statistical procedure to classify a set of quantitative independent variables to binary categories, making it ideal for use in a comparative analysis with a modern machine learning approach, namely the single-layer perceptron. [9] Essentially, the logistic regression procedure fits a logistic curve to the predictors, where the output of the curve is the probability that the target is `1` given the input variables. The Wald z-score can be used to test each independent variable as a significant predictor of the target. However, as in multiple linear regression, cross-correlation between independent variables remains a potential issue. [5]

The single-layer perceptron is a linear threshold unit that, when alone, constitutes the

simplest artificial neural network. [8] A severe limitation of the single-layer perceptron is that, as an iterative procedure, convergence is only guaranteed if the independent variables are linearly separable in $\mathbb{R}^n$, where n is the number of independent variables. However, in this report, we use the single-layer perceptron as the minimum component required to classify an algorithm as machine learning. In this manner, if machine learning is shown to be an improvement over conventional statistics, we have demonstrated that even at a minimal implementation machine learning provides advantages over logistic regression when applied to public security analysis in finance.

1.3 Goals

The primary goal of this paper is to compare predictions of the prices of publicly traded securities in the near future using a single-layer perceptron versus logistic regression. A secondary goal of this paper is to validate the application of machine learning in the financial discipline.

Currently, statistical models, most notably the Black-Scholes model, are heavily used in options pricing. Although high-frequency trading strategies have taken advantage of Gaussian statistics for extremely short-term trading (on the order of microseconds to milliseconds), trade commissions limit these strategies to institutions with large financial reserves. Validation of the single-layer perceptron for short-term trading (on the order of a few days) would give the ordinary person access to a multitude of strategies previously out of reach.

2 Materials and Methods

2.1 Web Scraping and Data Analysis

A list of all ticker symbols currently trading on the New York Stock Exchange (NYSE) is publicly available. The Java programming language was used to build a database of financial values about each ticker symbol (Appendix D). Yahoo! Finance, like nearly every database, is incomplete, and missing cases were eliminated prior to use by considering only complete cases. In addition to the ticker symbol (ID) and target variables, the complete data included 53 financial variables for 4,828 securities traded publicly on NYSE. These variables were then narrowed down to only significant predictors in the logistic regression model (Appendix A). The data was stored in Wickham and Grolemund's "tidy" format as a CSV file. [7] The R programming language was used for exploratory data analysis and data visualization with the help of the `tidyverse` package (Appendix B). [4, 6]

The target variable was calculated by comparing the price of the security when the data was collected to the price of the security two weeks post-data collection. If the price of the security increased over this time period, the target for the case was set to `1`, otherwise it was set to `0`. It should be noted that the web formatting for Yahoo! Finance's database has been changed since the start of this project, and the Java code might be out-of-date as a result; however, the data sets were manually validated at the time of collection, so data integrity is not of concern.

2.2 Statistical Modeling and Perceptron

Given the choice of either `0` or `1` as our binary target, we use logistic regression as the control conventional statistical procedure for comparative analysis with the single-layer perceptron. The target variable's value was determined by looking at each symbol's price 7 days after the data was collected, and comparing it to the original price; price decreases correspond to a target of `0`; price increases correspond to a target of `1`. The statistical procedure was implemented in R's `base` package under the `glm` method with the `binomial` family parameter (Appendix B). [4] In line with current statistical procedures, the entire data set is used to fit the logistic function. Rather than using the Likelihood Ratio Test to test the logit fit, we use the null and residual deviances as a measure of prediction capability. Although the Likelihood Ratio Test is appropriate for most statistical purposes, the null and residual deviance values grant us a measure of how well the logistic model would generalize to a larger population. This is more analogous to measuring the accuracy of machine learning algorithms on a testing data set, rather than simply the training data set, and should provide a better comparison.

A single-layer perceptron was implemented in the Python programming language (Appendix C). As is appropriate for machine learning algorithms, the entire data set was split into three smaller sets using an R program (Appendix B, code 3): a training set (70%), a validation set (20%), and a testing set (10%). The single-layer perceptron was trained on the training set and then checked and improved on the validation set. Our perceptron model parameters are learning rate, initial weights, initial bias, and dataset on which the single-layer perceptron is performed. The initial learning rate was 0.1, and the initial weights and

bias were 0.5. Accuracy of the single-layer perceptron was measured by the percent correct predictions from the testing set alone. The output of the algorithm is formatted in CSV file that contains the accuracy rate of the perceptron on testing set, and the resulting graphs were plotted to make comparisons.

2.3 Comparative Analysis

Comparative analysis between the conventional statistical and machine learning algorithms was conducted using the measures described in Section 2.2. Specifically, the null and residual deviances will be used to evaluate the predictive capability of the logistic regression model, and the test set accuracy will be used to evaluate the predictive capability of the single-layer perceptron. Comparative analysis of the difficulty and efficiency of the implementations will also be conducted to provide a practical overview of which method to use under which circumstances, assuming the models provide sufficiently good predictive capability.

3 Results

3.1 Logistic Regression to Predict Price Momentum

Of the 54 variables collected for each company, 2 variables (Ticker Symbol and Target) were not part of the set of independent variables used as predictors. The 52 remaining variables were then incorporated into a logistic regression model generated by the generalized linear model function (`glm()`; Appendix B). Variables that were not significant predictors of the Target variable were sifted out, leaving behind 13 significant predictors that were incorporated into a new logistic regression model.

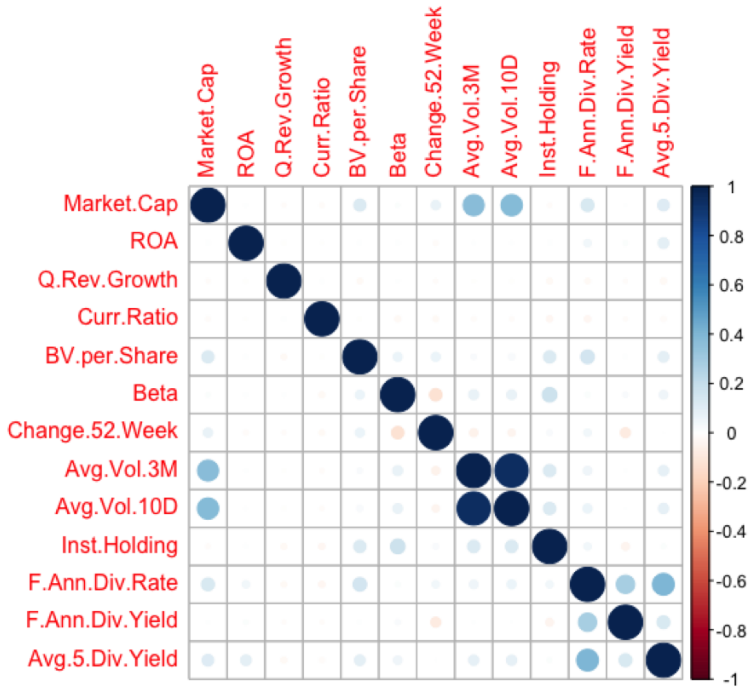

Figure 1: A correlation matrix to detect multicollinearity between significant predictors in logistic regression model. Circle size and color correspond to correlation strength. Only `Avg.Vol.3M` and `Avg.Vol.10D` exhibited a risk of multicollinearity due to high correlation. To correct for this, `Avg.Vol.3M` was removed from the model as the 10-day timescale of `Avg.Vol.10D` better matched the financial data.

A correlation matrix of these 13 variables revealed a large correlation between the average 3-month trading volume and the average 10-day trading volume, and a small correlation between forward annual dividend rate and two associated variables (Figure 1). To correct for the cross-correlation, two more variables were removed from the logistic regression model: average 3-month trading volume and forward annual dividend rate. Since 10 days is closer to our prediction timescale than 3 months, the average 10-day trading volume was preserved; since the forward annual dividend rate was correlated to the other two variables stronger than they were correlated to each other, it was removed.

Table 1: Table of coefficients for the final logistic regression model. The third column provides the level of significance for each coefficient as a predictor to the target variable. The 11 variables included in this table survived the two preprocessing sifting steps.

Coefficient Name	Coefficient Value	Significance (p-value)
Intercept	-8.3079	0.0010
Market Cap	-4.1229	0.0104
Return on Assets	51.9994	0.0272
Quarterly Revenue Growth	-27.6501	0.0042
Current Ratio	-41.2720	0.0000
Book Value per Share	4.6063	0.0143
Beta	3.5653	0.0051
52 Week Change	-6.9892	0.0000
Average 10-day Volume	9.0007	0.0000
% Held by Institutions	4.1616	0.0000
Forward Annual Dividend Yield	38.0765	0.0000
5-Year Average Dividend Yield	2.3670	0.0000

Table 1 gives a summary of the logistic regression model coefficients and predictor significance values. Overall, the model had a null deviance of 6692.3 and a residual deviance of 5889.9. Given the very low improvement achieved by a multiple variable logistic regression model of 11 independent variables, we can state with confidence that although the statistical model gave a list of significant predictors, none of the predictors were actually effective in predicting the target. This conclusion of further supported by the model's extremely large Akaike's IC value of 5913.9. Note that although the residual deviance indicates a degree of predictability higher than chance alone, the large Akaike's IC value reduces the predictability score by indicating that the logistic regression does not correctly capture the process that produces the data; this could be a result of overfitting, overcomplexity in the model, or similar statistical issues.

3.2 Estimation Using a Single-Layer Perceptron

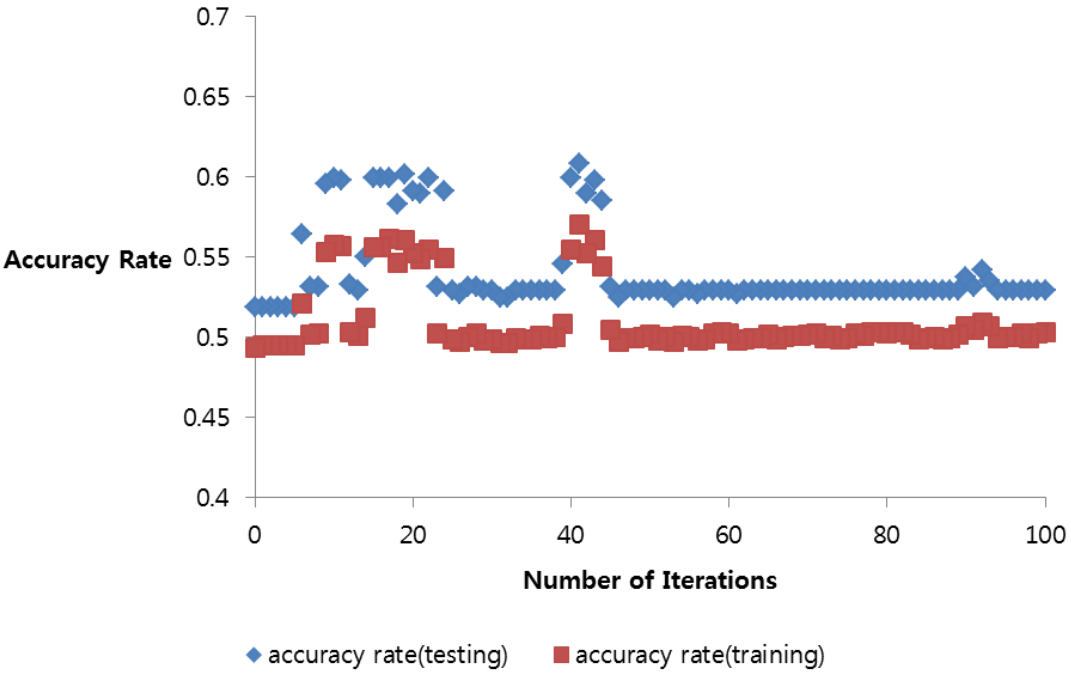

Figure 2: Change in accuracy rate of the single-layer perceptron as a function of the number of iterations for which it is run. The color of the point refers to the data set on which the perceptron was tested (testing versus training). Since the accuracy rate of the testing data set was always near the accuracy rate of the training data set, the single-layer perceptron is not being overfitted to the training set. The results of the perceptron on the validation data set are not visualized here.

Out of 54 variables, we used 14 variables including the target. Our input variables were learning rate, initial weights, initial bias, the number of iterations. Starting with changing the learning rate, the result stayed the same. Changing the initial weights and the initial bias respectively had no impact on the outcome of our perceptron algorithm. Varying the number the iterations changed the accuracy rate of the data.

Looking at the Figure 2, after 45 iterations the accuracy rate of the single-layer perceptron for both testing and training dataset stayed approximately constant. Therefore, we set the maximum iterations to 100. Also, it seems like the accuracy rate of the single-layer perceptron converges even though the final weights and bias do not converge.

Table 2: Table of coefficients for the single-layer perceptron model at 41th iteration. The 13 variables included in this table survived the first preprocessing sifting step.

Coefficient Name	Coefficient Value
Bias	-0.1000
Market Cap	-0.2252
Return on Assets	0.6375
Quarterly Revenue Growth	-0.8933
Current Ratio	-0.9636
Book Value per Share	0.1016
Beta	0.2019
52 Week Change	-0.3954
Average 3-month Volume	0.0503
Average 10-day Volume	0.1436
% Held by Institutions	0.3655
Forward Annual Dividend Rate	0.0555
Forward Annual Dividend Yield	2.0489
5-Year Average Dividend Yield	0.2405

Table 2 gives a summary of the single-layer perceptron model coefficients at 41st iteration. Starting with the initial bias of 0.5 and the initial weights of 0.1, we achieved the highest accuracy rate of 0.6079 at 41st iteration. The bias turned out to be -0.1, and the weight of each categorical variables is given in the table 2.

4 Conclusions

4.1 Goals

The primary goal of this paper was to compare the predictive capability of a logistic regression model to a single-layer perceptron. Based on the results section, we see that the single-layer perceptron is superior to the logistic regression model in terms of predictive capability. This is because the logistic regression model provides negligible improvement over random chance. In contrast, at its peak, the single-layer perceptron provided a larger prediction accuracy on the testing data set.

Our secondary goal was to look at the application of machine learning in the discipline of finance. We can state with confidence that machine learning is a very useful tool in financial security analysis, especially the analysis of publicly traded companies. The single-layer perceptron is the simplest neural network, and only converges when the input variables are linearly separable. As this condition was unmet by our training data, the single-layer perceptron used in this paper did not converge, and required an analysis of predictive capability based on iteration (Figure 2). However, in addition to the improvement provided by the single-layer perceptron relative to logistic regression, the multiple-layer perceptron would provide an additional qualitative jump in results that would translate to quantitatively increased predictive capability.

In summary, we successfully achieved both the goals set for this project: first, we demonstrated that even basic machine learning algorithms (in this case, the single-layer perceptron) provide an increase in predictive capability in the security analysis of publicly traded companies; second, as a consequence of our primary goal, we have illustrated that machine learning tools do provide valuable additions to the financial discipline, and should be considered for use by the common investor as a simpler alternative to current quantitative financial investing.

4.2 Implications of Comparative Analysis

It should be noted that although the single-layer perceptron provided better predictive capability than the logistic regression model, it did not do so on its own. Optimization of the single-layer perceptron was conducted using results from the logistic regression model. Specifically, the single-layer perceptron provided the best results with a low number of iterations when the insignificant predictor columns were removed from the data set; however, the significance of each predictor was determined using logistic regression. Additionally, although the variation in the accuracy rate of the single-layer perceptron appears to decrease as the number of iterations increases, the weight and bias coefficients do not converge, indicating that the single-layer perceptron does not converge for the financial data set. Thus, to gain maximum predictive accuracy, iteration analysis needed to be conducted on the single-layer perceptron, adding an additional step to its analysis. However, despite the caveats of using machine learning algorithms for prediction, the improvements provided are worth the extra effort. Due to the use of statistics in identifying significant predictors, we conclude that using machine learning algorithms with supporting statistical tools provides better accuracy than using either individually. Although this resembles the ensemble method of combining weak

models for a better result, it is qualitatively different since the two methods being combined here are of separate regimes: the statistical regime and the machine learning regime.

Acknowledgements

We thank Ildikó Stark, teaching assistant, for assisting us in writing the code for the single-layer perceptron, and Dr. Péter Érdi, professor, for teaching the Machine Learning course at Kalamazoo College.

References

[1] Benjamin Graham. *The Intelligent Investor*. Harper & Row Publishers, 4th edition, January 1986.

[2] Benjamin Graham and David Dodd. *Security Analysis*. McGraw-Hill Education, 6th edition, September 2008.

[3] John J. Murphy. *Technical Analysis of the Financial Markets: A Comprehensive Guide to Trading Methods and Applications*. New York Institute of Finance, January 1999.

[4] R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2015.

[5] Galit Shmueli, Nitin R. Patel, and Peter C. Bruce. *Data Mining for Business Intelligence: Concepts, Techniques, and Applications in Microsoft Office Excel with XLMiner*. Wiley, 2nd edition, October 2010.

[6] Hadley Wickham. *tidyverse: Easily Install and Load 'Tidyverse' Packages*, 2017. R package version 1.1.1.

[7] Hadley Wickham and Garrett Grolemund. *R for Data Science: Import, Tidy, Transform, Visualize, and Model Data*. O'Reilly Media, January 2017.

[8] Wikipedia, the free encyclopedia. *Feedforward neural network*, 2017.

[9] Wikipedia, the free encyclopedia. *Logistic regression*, 2017.

A Table of Variable Names

The following table lists the financial variables that correspond to variable names in the correlation plot (Figure 1). Note that only significant predictors in the logistic regression model are included, although far more (54 variables) were scraped from Yahoo! Finance. Each variable satisfied the condition $p < 0.1$.

Variable Name	Corresponding Financial Variable
Market.Cap	Intraday market cap on NYSE
ROA	Return on assets
Q.Rev.Growth	Quarterly Revenue Growth (year-on-year)
Curr.Ratio	Current Ratio
BV.per.Share	Book Value per Share
Beta	Beta
Change.52.Week	% Change in Price over Past 52 Weeks
Avg.Vol.3M	Average Volume Traded (past 3 months)
Avg.Vol.10D	Average Volume Traded (past 10 days)
Inst.Holding	% Shares Held by Institutions
F.Ann.Div.Rate	Forward Annual Dividend Rate
F.Ann.Div.Yield	Forward Annual Dividend Yield
Avg.5.Div.Yield	Average Dividend Yield (past 5 years)

B Data Analysis Source Code - R

Given the dataset from the source code in Appendix D, we needed to normalize the dataset. This was accomplished using the `Normalize.R` source shown below. After this, modeling the data using logistic regression and data visualizations were created using the `LogReg.R` file. Note that in order to run the code for the single-layer perceptron, the `split.R` script must be run to produce the training, validation, and testing data sets.

Code 1: Normalize.R source code.

```
# read in data set
data <- read.csv("Database.csv")

# function that normalizes a column
norm.col <- function(vec) {
    min.val <- min(vec)
    max.val <- max(vec)

    return((vec-min.val)/(max.val-min.val))
}

# normalize every column (x - min) / (max - min) except ticker
    symbol and target
for (i in 1:ncol(data)) {
    if (names(data)[i] != "Ticker.Symbol" && names(data)[i] != "
        Target") {
        data[[i]] <- as.numeric(data[[i]])
        data[[i]] <- norm.col(data[[i]])
    }
}

# write out to NormData.csv
write.csv(data, "NormData.csv", row.names=FALSE)
```

Code 2: LogReg.R source code.

```
# import required libraries
library(corrplot)

# read in data
norm.data <- read.csv("NormData.csv")

# make logistic regression with all variables
logreg.1 <- glm(Target ~ . - Ticker.Symbol, data = norm.data,
    family="binomial")
summary(logreg.1) # see which variables are significant

```

```
# logistic regression with only significant predictors
logreg.2 <- glm(Target ~ Market.Cap..intraday. +
                         Return.on.Assets..ttm.. +
                         Qtrly.Revenue.Growth..yoy.. +
                         Current.Ratio..mrq.. +
                         Book.Value.Per.Share..mrq.. +
                         Beta. +
                         X52.Week.Change +
                         Avg.Vol..10.day. +
                         X..Held.by.Institutions +
                         Forward.Annual.Dividend.Rate +
                         Forward.Annual.Dividend.Yield +
                         X5.Year.Average.Dividend.Yield,
                  data = norm.data, family="binomial")

logreg.2$null.deviance

# subset data from significant variables for correlation matrix
sub.norm <- subset(norm.data, select=c("Market.Cap..intraday.",
                                       "Return.on.Assets..ttm..",
                                            "Qtrly.Revenue.Growth..
                                               yoy..",
                                            "Current.Ratio..mrq..",
                                            "Book.Value.Per.Share..
                                               mrq..",
                                            "Beta.",
                                            "X52.Week.Change",
                                            "Avg.Vol..3.month.",
                                            "Avg.Vol..10.day.",
                                            "X..Held.by.
                                               Institutions",
                                            "Forward.Annual.
                                               Dividend.Rate",
                                            "Forward.Annual.
                                               Dividend.Yield",
                                            "X5.Year.Average.
                                               Dividend.Yield"))
names(sub.norm) <- c("Market.Cap",
                     "ROA",
                     "Q.Rev.Growth",
                     "Curr.Ratio",
                     "BV.per.Share",
                     "Beta",
                     "Change.52.Week",
                     "Avg.Vol.3M",
```

```
                                "Avg.Vol.10D",
                                "Inst.Holding",
                                "F.Ann.Div.Rate",
                                "F.Ann.Div.Yield",
                                "Avg.5.Div.Yield")

M <- cor(sub.norm)
corrplot(M, mar=c(1,1,1,1))
```

Code 3: split.R source code.

```
#Read in the data
data<-read.csv("Tester.csv", nrow=4820)

#permute the index using sample method
permutation <- sample(1:nrow(data), nrow(data), replace = FALSE)

#sets break points at 70th percentile and 90th percentile
breaks <- c(0.7*nrow(data), 0.9*nrow(data))

#separates data into three parts: 70%, 20%, and 10% of the data
first.sample <- 1:breaks[1]
second.sample <- (breaks[1]+1):breaks[2]
third.sample <- (breaks[2]+1):nrow(data)

one <- data[permutation[first.sample],]
two <- data[permutation[second.sample],]
three <- data[permutation[third.sample],]

##Export the output into csv file
write.csv(one, "Training.csv", row.names = FALSE)
write.csv(two, "Validation.csv", row.names = FALSE)
write.csv(three, "Testing.csv", row.names = FALSE)
```

C Single-Layer Perceptron Source Code - Python

The single-layer perceptron was coded using Python, as shown in the source code below.

Code 4: perceptron.py source code.

```
1  import numpy as np
2
3  ##reads the data given as a parameter line by line
4  ##stores the input value of each categorical variables into a
       vector
5  def readin(data):
6          with open(data, "r") as fo:
7                  dataset = []
8                  linenum = 0
9                  #for each line in the input(csv file), we spilt
                       the line by comma delimeter
10                 for line in fo:
11                         if linenum==0:
12                                 linenum += 1
13                                 continue
14                         newdata=line.split(",")
15                         colnum = 0
16                         tempdata = []
17                         #for each numeric value in each line, we
                               store it into a vector
18                         for s in newdata:
19                                 if colnum==0:
20                                         colnum += 1
21                                         continue
22                                 s=float(s)
23                                 colnum += 1
24                                 tempdata.append(s)
25                         dataset.append(tempdata)
26                         linenum += 1
27
28         return dataset
29
30 ##predicts the target value using the equation of dot product of
       input vector and weight vector plus bias
31 def predict(data,weights,bias):
32         input=np.array(data[:-1])
33         weight=np.array(weights)
34         if input.dot(weight)+bias>0:
35                 return 1
36         else:
```

```
37                  return 0
38
39  ##updates the bias and the weights using the error and the
        learning rate, which is achieved by actual target minus the
        predicted target
40  def update(learningrate, data, weights, bias):
41          error=data[-1]-predict(data,weights,bias)
42          newbias=bias+learningrate*error
43          newWeight=np.array(weights)+learningrate*error*np.array(
                data[:-1])
44          return (newbias,newWeight)
45
46  ##goes through N iterations, passed by a parameter, and updates
        bias and weights every iteration
47  def perceptron(learningrate, dataset, weights, bias, N):
48          for i in range(N):
49                  for data in dataset:
50                          (bias,weights)=update(learningrate,data,
                                weights,bias)
51                          #print(bias,weights)
52          return (bias,weights)
53
54
55
56  ##Read in training dataset
57  data="Training.csv"
58  dataset=readin(data)
59
60  ##Set initial conditions: learning rate, weights, bias, and number
        of iterations
61  lrate = 0.1
62  weig = [0.3] * (len(dataset[0])-1)
63  bias = 0.5
64  N = 150
65
66  output = open("ResultTester2.csv","w")
67  output.write("Number of iterations,accuracy rate(testing),accuracy
        rate(training)\n")
68
69
70  while N >= 0:
71          print(N)
72          if N < 0: continue
73          ##implements perceptron algorithm to acquire final bias
                and weights for N iterations
```

```
74          a = perceptron(lrate,dataset,weig,bias,N)
75          data2=readin("Testing.csv")
76          data3=readin("Training.csv")
77
78          # get accuracy on testing data set
79          count = 0
80          correct = 0
81          for line in data2:
82                  actual = line[-1]
83                  pred = predict(line, a[1], a[0])
84                  count += 1
85                  if actual == pred:
86                          correct += 1
87          output.write(str(N)+","+str(correct/count))
88
89          # get accuracy on training data set
90          count = 0
91          correct = 0
92          for line in data3:
93                  actual = line[-1]
94                  pred = predict(line, a[1], a[0])
95                  count += 1
96                  if actual == pred:
97                          correct += 1
98          output.write(","+str(correct/count)+"\n")
99          N-=1
```

D Web Scraping Source Code - Java

The Java programming language was used to scrape financial data from Yahoo! Finance, with the framework shown in Figure 3. The following code listings show the source code for the various classes used for web scraping.

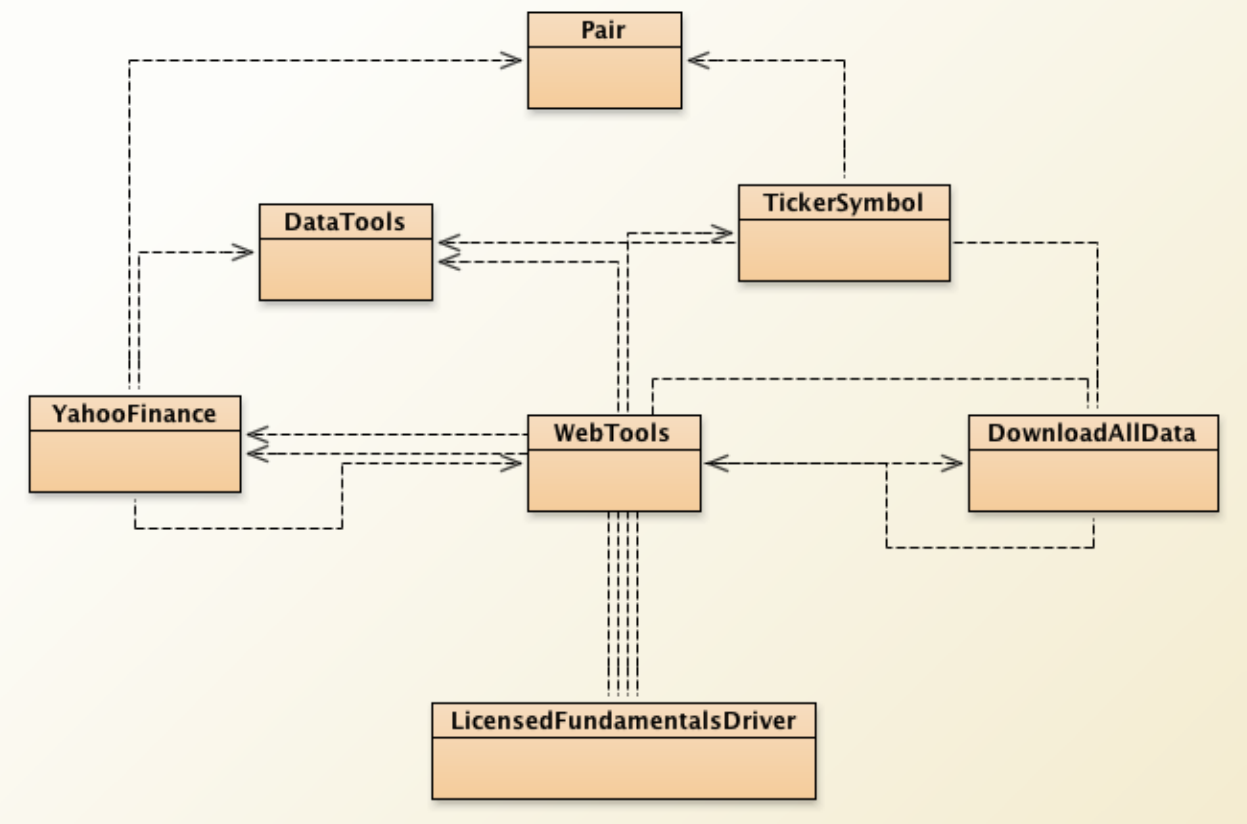

Figure 3: The dependency chart for the Java classes as visualized in the BlueJ IDE. The displayed classes were built from scratch for this project.

Code 5: Pair.java source code.

```
1  //Shallow immutable class implementing generics for a pair of
       objects
2  public class Pair<F,S>
3  {
4      //members
5      private final F first;
6      private final S second;
7
8      //constructor
9      public Pair(F setFirst, S setSecond)
10     {
11         first = setFirst;
```

```
12          second = setSecond;
13      }
14
15      //getters
16      public F getFirst()
17      {
18          return first;
19      }
20      public S getSecond()
21      {
22          return second;
23      }
24  }
```

Code 6: DataTools.java source code.

```
1   import java.util.ArrayList;
2   import java.io.File;
3   import java.io.PrintWriter;
4   import java.io.FileWriter;
5   import java.io.IOException;
6   import java.util.Scanner;
7   import java.util.Collections;
8
9   //class contains common methods to help dealing with data
10  public class DataTools
11  {
12      //private constructor to prevent instantiation
13      private DataTools(){}
14
15      //given a start string and end string, returns the part in the
            middle
16      public static String getPartBetween(String all, String start,
            String end) throws IllegalArgumentException
17      {
18          int beginStrIndex = all.indexOf(start)+start.length();
19          int endStrIndex = all.indexOf(end);
20
21          if (beginStrIndex==-1)
22              throw new IllegalArgumentException("Can't find 'start'
                    String");
23          else if (endStrIndex==-1)
24              throw new IllegalArgumentException("Can't find 'end'
                    String");
25          else if (endStrIndex<=beginStrIndex)
```

```
26          {
27              System.out.println(beginStrIndex+" "+endStrIndex);
28              throw new IllegalArgumentException("'start' String
                    appears after 'end String");
29          }
30
31          return all.substring(beginStrIndex,endStrIndex);
32      }
33
34      //return list of all indices of occurrence of string in larger
            string
35      public static ArrayList<Integer> allIndicesOf(String all,
            String search)
36      {
37          ArrayList<Integer>answer = new ArrayList<Integer>();
38
39          int currIndex = all.indexOf(search),
40              totalSoFar = 0;
41          while (currIndex>=0)
42          {
43              answer.add(totalSoFar+currIndex);
44              all = all.substring(currIndex+2);
45              totalSoFar += currIndex+2;
46              currIndex = all.indexOf(search);
47          }
48
49          return answer;
50      }
51
52      //wrapper method to output an html source to a file
53      public static void writeHTMLToFile(File fout, String
            htmlSource) throws IOException
54      {
55          PrintWriter out = new PrintWriter(new FileWriter(fout));
56
57          out.println(htmlSource);
58
59          out.close();
60      }
61
62      //remove commas
63      public static String noCommas(String before)
64      {
65          StringBuilder sb = new StringBuilder("");
66
```

```
67          for (int i=0; i<before.length(); i++)
68              if (before.charAt(i)!=',')
69                  sb.append(before.charAt(i));
70
71          return sb.toString();
72      }
73
74      //remove html tags in data
75      public static String noHTMLTags(String before)
76      {
77          String htmlTagRegex = "\\<[^\\<\\>]*\\>";
78
79          return before.replaceAll(htmlTagRegex,"");
80      }
81
82      //remove whitespace function to use in DownloadData class
83      public static String noWhitespace(String before)
84      {
85          StringBuilder after = new StringBuilder("");
86
87          char temp;
88          for (int i=0; i<before.length(); i++)
89          {
90              temp = before.charAt(i);
91              if (temp!=' ' && temp!='\n' && temp!='\r' && temp!='\t
                    ')
92                  after.append(temp);
93          }
94
95          return after.toString();
96      }
97
98      //get the list of ticker symbols from the CSV file
99      protected static ArrayList<String> readInTickers(File
            csvTickerList) throws IOException
100     {
101         Scanner fin = new Scanner(csvTickerList).useDelimiter(",\n
                ");
102         ArrayList<String>tickerList = new ArrayList<String>();
103
104         String tempLine, tempTicker;
105         int firstCommaIndex;
106         while (fin.hasNext())
107         {
108             tempLine = fin.nextLine();
```

```
109             firstCommaIndex = tempLine.indexOf(",");
110             tempTicker = tempLine.substring(0,firstCommaIndex);
111             if (!tempTicker.contains("^") && !tempTicker.contains(
                    ".") && !tempTicker.equalsIgnoreCase("symbol"))
112                 tickerList.add(DataTools.noWhitespace(tempTicker))
                        ;
113         }
114
115         return tickerList;
116     }
117
118     //sort each individual arraylist and then use a merge idea to
            avoid repeats
119     //remove tickers shared between two indices
120     //combine each ticker to uppercase before so don't have to use
            "equalsIgnoreCase"
121     protected static ArrayList<String> combineNoRepeats(ArrayList<
            String> a, ArrayList<String> b)
122     {
123         //can't just initialize with another arraylist's contents
                b/c have to convert to uppercase individually
124         ArrayList<String>firstCombine = new ArrayList<String>();
125         firstCombine.ensureCapacity(a.size()+b.size());
126         for (String curr:a)
127             firstCombine.add(curr.toUpperCase());
128         for (String curr:b)
129             firstCombine.add(curr.toUpperCase());
130
131         Collections.sort(firstCombine);
132
133         ArrayList<String>answer = new ArrayList<String>();
134         answer.ensureCapacity(firstCombine.size());
135
136         answer.add(firstCombine.get(0));
137         for (int i=1; i<firstCombine.size(); i++)
138         {
139             if (!firstCombine.get(i).equals(firstCombine.get(i-1))
                    )
140                 answer.add(firstCombine.get(i));
141         }
142
143         return answer;
144     }
145
146     //remove all characters that are not digits or periods
```

```
147     public static String digitDotOnly(String a)
148     {
149         StringBuilder sb = new StringBuilder("");
150         for (int i=0; i<a.length(); i++)
151             if (a.charAt(i)>='0' && a.charAt(i)<='9' || a.charAt(i
                )=='.')
152                 sb.append(a.charAt(i));
153         return sb.toString();
154     }
155
156     //quick method for correct significant digits
157     public static String dotUntilNum(String a, int desiredLen)
158     {
159         int numAdd = desiredLen-a.length();
160         StringBuilder sb = new StringBuilder(a);
161         for (int i=0; i<numAdd; i++)
162             sb.append('.');
163         return sb.toString();
164     }
165 }
```

Code 7: TickerSymbol.java source code.

```
1  import java.util.ArrayList;
2
3  //class for a ticker symbol (publicly traded security)
4  public class TickerSymbol
5  {
6      //member variables
7      private final String SYMBOL;                    //always
           uppercase, alpha, 3-5?
8      private ArrayList<Pair<String,String>> data;    //fundamentals
           (mostly?)
9
10     //constructor
11     public TickerSymbol(String setSymbol)
12     {
13         SYMBOL = setSymbol;
14         data = new ArrayList<Pair<String,String>>();
15     }
16
17     //keep track of var w/ max # of attributes
18     private static int maxNumAttributes = 0;
19
20     //getters
```

```
21      public String getSymbol()
22      {
23          return SYMBOL;
24      }
25
26      //null return value indicates result not found
27      public String getValue(String key)
28      {
29          for (Pair<String,String> curr:data)
30              if (key.equals(curr.getFirst()))
31                  return curr.getSecond();
32          return null;
33      }
34
35      //add an attribute to the ticker symbol (multiple methods)
36      public void addAttribute(String key, String value)
37      {
38          Pair<String,String>temp = new Pair<String,String>(key,
                value);
39          this.addAttribute(temp);
40      }
41      public void addAttribute(Pair<String,String> toAdd)
42      {
43          data.add(toAdd);
44          maxNumAttributes = Math.max(maxNumAttributes,data.size());
45      }
46
47      //wrapper method to add multiple attributes
48      public void addAttributes(ArrayList<Pair<String,String>> toAdd
            )
49      {
50          for (Pair<String,String>curr:toAdd)
51              this.addAttribute(curr);
52      }
53
54      //utility methods
55      public int numAttributes()
56      {
57          return data.size();
58      }
59      public static int totalAttributes()
60      {
61          return maxNumAttributes;
62      }
63
```

```
64      //print out the header line for the csv file
65      public String csvHeader()
66      {
67          StringBuilder sb = new StringBuilder();
68          if (data.size()>0)
69              sb.append("Ticker Symbol");
70          for (int i=0; i<data.size(); i++)
71          {
72              sb.append(',');
73              sb.append(data.get(i).getFirst());
74          }
75          return sb.toString();
76      }
77
78      //return csv data line in order of header line
79      public String csvData()    //no newline; no ordering
80      {
81          StringBuilder sb = new StringBuilder();
82          if (data.size()>0)
83              sb.append(this.getSymbol());
84          for (int i=0; i<data.size(); i++)
85          {
86              sb.append(',');
87              sb.append(""+data.get(i).getSecond()+"");    //
                    potentially add quotes to deal with annoying data
88          }
89          return sb.toString();
90      }
91  }
```

Code 8: YahooFinance.java source code.

```
1   import java.util.ArrayList;
2   import java.io.IOException;
3
4   //class that takes care of everything with yahoo finance database
5   public class YahooFinance
6   {
7       //private constructor to prevent instantiation
8       private YahooFinance(){}
9
10      private static final String HEAD_START_STRING = "<td class=\"
            yfnc_tablehead1\" width=\"74%\">",
11                                 DATA_START_STRING = "td class=\"
                                       yfnc_tabledata1\">",
```

```
12                                      HEAD_END_STRING_1 = "<font",
13                                      HEAD_END_STRING_2 = "</td>",
14                                      DATA_END_STRING = "</td>";
15
16      //wrapper method to extract all values from Key Statistics
            page
17      public static ArrayList<Pair<String,String>> getAllKeyStats(
            String ticker) throws IOException
18      {
19          return getAllKeyStatsFromSource(getKeyStatsHTML(ticker));
20      }
21
22      //actual method that extracts all values from Key Statistics
            page on Yahoo! Finance database
23      private static ArrayList<Pair<String,String>>
            getAllKeyStatsFromSource(String source)
24      {
25          ArrayList<Pair<String,String>>keyStats = new ArrayList<
                Pair<String,String>>();
26          //ArrayList<Integer>headLocations = DataTools.allIndicesOf
                (source,HEAD_START_STRING);
27
28          Pair<String,String>tempPair;
29          String tempFirst, tempSecond;
30          String tempSource = source;
31          int firstHeadIndex = tempSource.indexOf(HEAD_START_STRING)
                ;
32          int endHeadIndex, firstDataIndex, endDataIndex;
33          while (firstHeadIndex>=0)
34          {
35              //table head
36              tempSource = tempSource.substring(firstHeadIndex);
37              firstHeadIndex = HEAD_START_STRING.length();
38              endHeadIndex = closerStringIndex(tempSource.indexOf(
                    HEAD_END_STRING_1),tempSource.indexOf(
                    HEAD_END_STRING_2));
39              tempFirst = tempSource.substring(firstHeadIndex,
                    endHeadIndex);
40
41              //table data
42              firstDataIndex = tempSource.indexOf(DATA_START_STRING)
                    ;
43              tempSource = tempSource.substring(firstDataIndex);
44              firstDataIndex = DATA_START_STRING.length();
45              endDataIndex = tempSource.indexOf(DATA_END_STRING);
```

```
46              tempSecond = tempSource.substring(firstDataIndex,
                    endDataIndex);
47
48              //update
49              tempPair = new Pair<String,String>(DataTools.
                    noHTMLTags(DataTools.noCommas(tempFirst)),DataTools
                    .noHTMLTags(DataTools.noCommas(tempSecond)));
50              keyStats.add(tempPair);
51              firstHeadIndex = tempSource.indexOf(HEAD_START_STRING)
                    ;
52          }
53
54          return keyStats;
55      }
56
57      private final static String MEAN_RECOMMENDATION_START = "Mean␣
            Recommendation␣(this␣week):";
58      private final static String MEAN_NUM_BROKER_START = "No.␣of␣
            Brokers";
59      private final static int DATA_AFTER_HEADER = 45+
            MEAN_RECOMMENDATION_START.length();     //number of
            characters past which it's guaranteed data is within range
            of header
60
61      //method (and wrapper) that extracts mean analyst
            recommendation from analyst opinion page
62      public static ArrayList<Pair<String,String>> getAnalystOpinion
            (String ticker) throws IOException
63      {
64          return getAnalystOpinionFromSource(getAnalystOpinionHTML(
                ticker));
65      }
66      private static ArrayList<Pair<String,String>>
            getAnalystOpinionFromSource(String source)
67      {
68          //add analyst opinion then no. brokers
69          ArrayList<Pair<String,String>>answer = new ArrayList<Pair<
                String,String>>();
70
71          String tempFirst, tempSecond;
72          int headIndex = source.indexOf(MEAN_RECOMMENDATION_START),
73              endHeadIndex = headIndex+MEAN_RECOMMENDATION_START.
                    length();
74          int dataIndex = endHeadIndex+10, endDataIndex =
                DATA_AFTER_HEADER+headIndex;
```

```
75
76          //some tickers don't have an analyst opinion
77          if (headIndex<0)
78          {
79              Pair temp = new Pair<String,String>(
                    MEAN_RECOMMENDATION_START, "");
80                  temp2= new Pair<String,String>(
                        MEAN_NUM_BROKER_START, "");
81              answer.add(temp);
82              answer.add(temp2);
83              return answer;
84          }
85
86          tempFirst = MEAN_RECOMMENDATION_START;
87          tempSecond= DataTools.digitDotOnly(DataTools.noHTMLTags(
                source.substring(dataIndex, endDataIndex))).substring
                (1);    //remove first character b/c for some reason a
                '1' is always prepended (too lazy to actually check it
                out)
88          answer.add(new Pair<String,String>(tempFirst,tempSecond));
89
90          //add no. of brokers
91          headIndex = source.indexOf(MEAN_NUM_BROKER_START);
92          endHeadIndex = headIndex+MEAN_NUM_BROKER_START.length();
93          dataIndex = endHeadIndex+10;
94          endDataIndex = DATA_AFTER_HEADER+headIndex;
95
96          tempFirst = MEAN_NUM_BROKER_START;
97          tempSecond= DataTools.digitDotOnly(DataTools.noHTMLTags(
                source.substring(dataIndex, endDataIndex))).substring
                (1);    //remove first character b/c for some reason a
                '1' is always prepended (too lazy to actually check it
                out)
98          answer.add(new Pair<String,String>(tempFirst,tempSecond));
99
100         return answer;
101     }
102
103     //utility method to find attributes within source
104     private static int closerStringIndex(int index1, int index2)
105     {
106         if (index1<0 && index2<0) return index2;
107         if (index2<0) return index1;
108         return Math.min(index1,index2);
109     }
```

```
110
111     protected static final String KEY_STATS_PAGE = "http://finance
            .yahoo.com/q/ks?s=TICKER+Key+Statistics";
112     //download source for keystats page from yahoo! finance for
            given ticker symbol
113     private static String getKeyStatsHTML(String ticker) throws
            IOException
114     {
115         return WebTools.getSource(KEY_STATS_PAGE.replace("TICKER",
                ticker));
116     }
117
118     protected static final String ANALYST_OPINION_PAGE = "http://
            finance.yahoo.com/q/ao?s=TICKER+Analyst+Opinion";
119     //download source for analyst opinion page from Yahoo! finance
            for given ticker symbol
120     private static String getAnalystOpinionHTML(String ticker)
            throws IOException
121     {
122         return WebTools.getSource(ANALYST_OPINION_PAGE.replace("
                TICKER",ticker));
123     }
124 }
```

Code 9: WebTools.java source code.

```
1  import java.net.*;
2  import java.io.*;
3
4  //class takes care of all the source download aspects
5  public class WebTools
6  {
7      //downloads the html source of the passed url
8      public static String getSource(String url) throws IOException
9      {
10         URL pageLocation = new URL(url);
11         BufferedReader in = new BufferedReader(new
               InputStreamReader(pageLocation.openStream()));
12
13         String inputLine;
14         StringBuilder sb = new StringBuilder("");
15         while ((inputLine = in.readLine()) != null)
16         {
17             sb.append(inputLine);
18             sb.append("\n");
```

```
19          }
20          return sb.toString();
21      }
22
23      //outputs the html source at the given url to the given file
24      public static void writeSourceToFile(String url, File fout)
            throws IOException
25      {
26          PrintWriter out = new PrintWriter(new FileWriter(fout));
27          String source = WebTools.getSource(url);
28
29          out.println(source);
30
31          out.close();
32      }
33  }
```

Code 10: DownloadAllData.java source code.

```
1   import java.io.*;
2   import java.util.*;
3
4   //class takes care of downloading all the html source files
5   //from the Yahoo! Finance database
6   public class DownloadAllData
7   {
8       //driver method using list of tickers as input
9       public static void main(String[] args) throws IOException
10      {
11          //declare input stuff
12          File nyseCSV = new File("Files/nyse-list.csv");
13          File nasdaqCSV=new File("Files/nasdaq-list.csv");
14
15          ArrayList<String> nyseList = DataTools.readInTickers(
                nyseCSV);
16          ArrayList<String> nasdaqList = DataTools.readInTickers(
                nasdaqCSV);
17          ArrayList<String> allTickers = DataTools.combineNoRepeats(
                nyseList,nasdaqList);
18
19          //declare output stuff
20          File temp;
21
22          //get & output desired info page-by-page (don't lose all
                data in case of crash; buffer every few pages to avoid
```

```
            overflow & preserve data)
23          //output NYSE
24          for (String currTicker:allTickers)
25          {
26              System.out.println(currTicker);
27              temp = new File("Files/"+currTicker+".txt");
28              WebTools.writeSourceToFile(YahooFinance.KEY_STATS_PAGE
                    .replace("TICKER",currTicker),temp);
29          }
30
31          //close output and input streams
32      }
33
34
35  }
```

Code 11: LicensedFundamentalsDriver.java source code.

```
 1  import java.io.*;
 2  import java.util.*;
 3
 4  //driver for all the work
 5  public class LicensedFundamentalsDriver
 6  {
 7      //driver initial method
 8      public static void main(String[] args) throws IOException
 9      {
10          PrintWriter fout = new PrintWriter(new FileWriter("Scraped
                ␣Dataset.csv"));
11          File nyseList = new File("nyse-list.csv"),
12               nasdaqList=new File("nasdaq-list.csv");
13          ArrayList<String>allSymbols = DataTools.combineNoRepeats(
                DataTools.readInTickers(nyseList),DataTools.
                readInTickers(nasdaqList));
14
15          //TickerSymbol[] allTickers = new TickerSymbol[allSymbols.
                size()];
16          TickerSymbol current;
17          for (int i=0; i<allSymbols.size(); i++)
18          {
19              System.out.print(DataTools.dotUntilNum(allSymbols.get(
                    i),15));
20              current = new TickerSymbol(allSymbols.get(i));
21              current.addAttributes(YahooFinance.getAllKeyStats(
                    current.getSymbol()));      //add key stats
```

```
22                  current.addAttributes(YahooFinance.getAnalystOpinion(
                        current.getSymbol())); //add analyst opinion
23                  System.out.print("PARSED.........");
24                  if (i==0)
25                      fout.println(current.csvHeader());
26                  fout.println(current.csvData());
27                  System.out.println("PRINTED TO FILE");
28                  if (i%50==0 && i>0)
29                  {
30                      fout.flush();
31                      System.out.println("OUTPUT BUFFER FLUSHED AT N="+i
                            );
32                  }
33              }
34              System.out.println("\n\nTASK COMPLETED");
35              fout.close();
36          }
37
38  }
```