Minding the Gap: Using a Neural Net to Classify the Price Pattern Following a Gap Discontinuity

Michael Szczepaniak

May 15, 2008

Contents

1 Introduction **1**

2 Background and Definitions **2**

3 Pre-Analysis Data Preparation **3**

4 Linear Discriminant Analysis (LDA) Classification **5**

4.1 Implementation of LDA 5

4.2 R code for LDA 5

5 Neural Network Classification of Gap Data **8**

5.1 Implementation of NNC 8

5.2 R code for NNC 9

6 Results **13**

6.1 Comparing LDA with NNC on the Three Selected Dow Components 13

6.2 A Personal Curiosity: Using One Stocks NNC to Make Predictions Based on Another Stocks Data 15

7 Discussion **16**

8 Additional R code used in this work **18**

1 Introduction

This paper examines the feasibility of profitably trading what is referred to here as a *gap* strategy [1]. This was done by attempting to predict the price patter over a 5-day period following a 1% or greater gap (up or down) in price using a neural net classifier (NNC). Even with limited training, the NNC showed an overall accuracy of between 74% and 83%. The NNC results were benchmarked against the performance of a classifier constructed using linear discriminant analysis (LDA) which showed an overall accuracy of between 33% and 37%.

Three Dow 30 component stocks (Alcoa, GM, and Intel) were selected for analysis because they showed the greatest number of gaps over the selected time period. Pricing and volume data was collected over a period through and including January 2006 through April 2008 [2]. A 50%-50% training versus test split of the sparse sample data sets ($N_{max} = 152$ to $N_{min} = 106$) was selected. Five post-gap pricing patterns were created and assigned to the days in which gaps occurred and the NNC and LDA outputs were designed to output one of these five classes based on the maginitude and direction of the gap, the price (open, high, low, and close), and volume over a 5-day period preceding the gap.

2 Background and Definitions

US stocks trade from 9am to 4pm EST. Between the closing price of the previous day and the opening price of the next, prices can show discontinuities or gaps. In the context of this paper, a gap is deemed to have occured if the opening price is different from the closing price of the previous trading session. More specifically, a gap in this paper refers to a $\geq 1\%$ or $\leq -1\%$ difference between the pre-gap close and the post-gap opening price. Figure 1 shows an example of price gaps as seen on a typical bar chart.

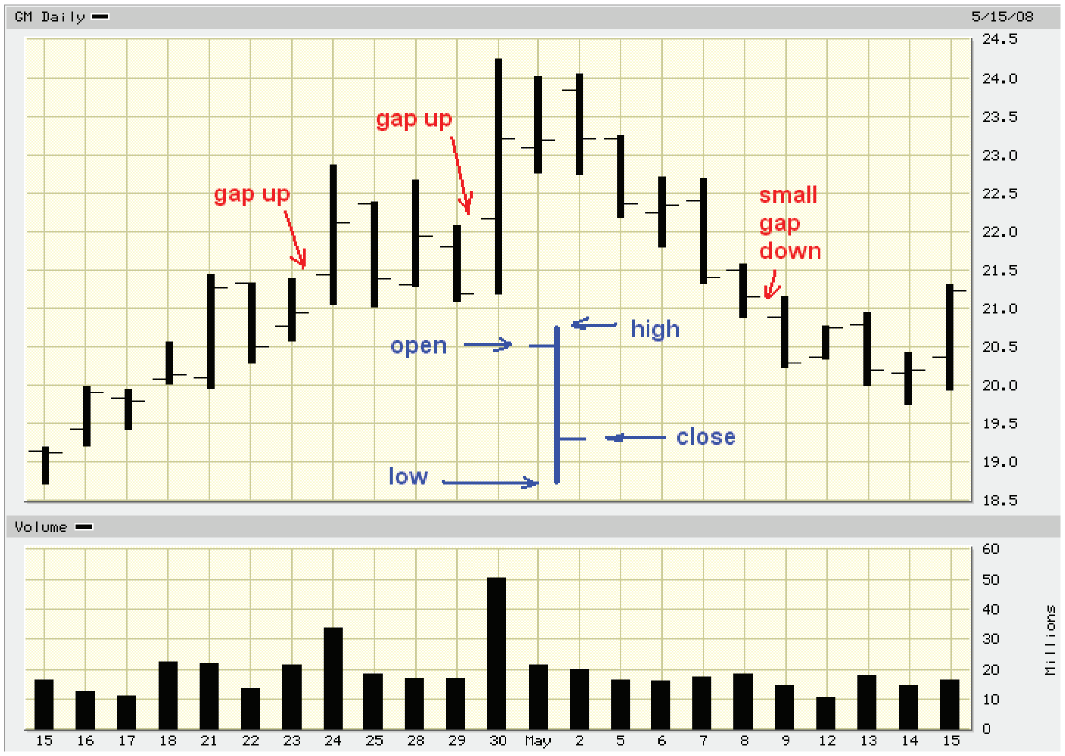

Figure 1: 30 Day barchart for GM showing gaps

Once a gap has occurred, price can either *fill* or *fail to fill* this gap over a certian time period. A gap is said to have filled if its price manages to reach its closing price prior to the gap over a certain specified period of time. Conversely, if the price fails to reach the pre-gap closing price, it is said to have failed to fill. The time period selected in this work is 5 trading days.

Post-gap pricing patterns are first characterized as having filled the gap or failed to fill the gap within the specifed period following a gap. The first three classes are assigned to filled gap pricing trajectories based on whether price finished above, within, or below the gap zone after having filled. The gap zone is defined by prices within the range between the pre-gap closing price and the post-gap opening price.

The third and fourth classes are assigned to pricing trajectories that failed to fill within the specifed time period following a gap. Prices that fail to fill can either finish inside the gap zone or outside the gap zone. There are only two types of these classes because if a price fails to fill its gap, it can either finish inside the gap zone or below the gap zone if the gap was down or above the gap zone if the gap was up.

The list below is a summary of the classes used in this analysis:

class 1: price fills the gap and finished above the gap zone

class 2: price fills the gap and finished within the gap zone

class 3: price fills the gap and finished below the gap zone

class 4: price fails to fill the gap and finished within the gap zone

class 5: price fails to fill the gap and finished outside the gap zone

3 Pre-Analysis Data Preparation

The following 5 steps were applied as part of the data preparation process prior to classification:

1. Query the data source for a paricular dow component over a specfied time frame.
2. Download the data for the selected dow component into a comma separated variable (csv) file.
3. Repeat 1. and 2. for the remaining dow components (30 total).
4. Run the R function `read.dow30` which does the following: reads each data file in a loop and creates a dataframe for each csv file read, adds 3 addtional columns to each dataframe (`gaps.size`, `gap.frac`, and `is.gap`), populates each of these columns, adds each dataframe to a list, and then returns the list.
5. Run the R function `makeInputsToClassify` passing in a particular dow components data frame as a parameter which creates the input data matrix which is later passed to either the LDA classifier or the NNC.

The functions `read.dow30` and `makeInputsToClassify` are listed and described in section 8. The function `makeInputsToClassify` calls the function `classifyGap` which is at the heart of the pre-classification data preparation. This function is listed below with descriptions defined with in-line comments.

```
################################################################################
# Returns an integer from 1 to 5 where:
# 1 - means that the gap filled and finished above the gap zone
# 2 - means that the gap filled and finished inside the gap zone
# 3 - means that the gap filled and finished below the gap zone
# 4 - means that the gap was NOT filled and finished inside the gap zone
# 5 - means that the gap was NOT filled and finished outside the gap zone
#
# p1 - pre-gap closing price
# p2 - post-gap opening price
# postpmin - minimum price during the specified period following the gap
# postpmax - maximum price during the specified period following the gap
# ppclose - closing price on the last day of the specified period
#
# PRECONDITION: p1 != p2, ppclose <= postpmax, and ppclose >= postpmin
################################################################################
classifyGap <- function(p1,p2,postpmin,postpmax,ppclose) {
  gapfilled <- gapFill(p1,p2,postpmin,postpmax)
  if(gapfilled) {  # must be class 1, 2, or 3 if it fills
#print("gap filled")
    if(finishAbove(p1,p2,ppclose))
      retval <- 1
    else {         # finished in or below gap zone
      if(finishBelow(p1,p2,ppclose))
        retval <- 3
      else
        retval <- 2
    }
  }
  else {           # gap was not filled , must be class 4 or 5
#print("gap did NOT fill")
    if(finishIn(p1,p2,ppclose))
      retval <- 4
    else
      retval <- 5
  }
  retval
}
```

The `classifyGap` function takes in 5 prices: pre-gap closing price (p1), post-gap opening price (p2), minimum price during the specified period following the gap (postpmin), maximum price during the specified period following the gap (postpmax), and closing price on the last day of the specified period (ppclose). The function returns a number 1 through 5 corresponding to a class which is defined in the comment section. The function calls 4 helper functions which are listed in section 8: `gapFill`, `finishAbove`, `finishBelow`, and `finishIn`. The function `gapFill` returns TRUE if the price fills at some point during the specified period, otherwise, it returns FALSE. The functions `finishAbove`, `finishBelow`, and `finishIn` return TRUE if the closing price on the last day of the period is above the gap zone, below the gap zone, or in the gap zone respectively, otherwise, it returns FALSE.

4 Linear Discriminant Analysis (LDA) Classification

4.1 Implementation of LDA

As descibed in an earlier work [5], the equation for the linear discriminant classifier is shown in equation (1).

$$\delta_k(x) = x^T \Sigma^{-1} \mu_k - \frac{1}{2} \mu_k^T \Sigma^{-1} \mu_k + \log P(C = k) \tag{1}$$

where μ_k are the class means, $P(C = k)$ is the probability of class k and Σ is the average covariance defined by equation (2).

$$\Sigma = \frac{1}{N - K} \sum_{k=1}^{K} \sum_{x \in X_k} (x - \mu_k)(x - \mu_k)^T \tag{2}$$

There will be K (the number of classes) values for $\delta_k(x)$ computed for each sample. The k that produces the highest value $\delta_k(x)$ determines the class.

4.2 R code for LDA

The R function `LDAclassifier` was used to select the best LDA classifier out of 100 iteration and is shown below. The results were tallied over 30 trials to get a reading on the overall average accuracy.

```r
################################################################################
# Runs LDA classifier ntrials number of times and saves the results the
# gave the highest testing accuracy.  The best results were then run over
# best.trials number of iterations to obtain an average accuracy for the
# selected classifier.
#
# all.data is N x (D+1) with the classes in the first column
# tsf - the training set fraction
# name.lda - sets the name of the saved output file
################################################################################
LDAclassifier <- function(all.data=XT,tsf=0.8,ntrials=100,
                          best.trials=30,name.lda="lda.default")  {

  Nclass <- length(unique(all.data[,1])) # calc the number of classes

  r.train <- matrix(0,Nclass,3)  # create place to hold training results
  r.test <- matrix(0,Nclass,3)   # create place to hold testing results

  for(i in 1:ntrials) {

    ## establish the random set of training rows based on the tsf
    trainRows <- trainSetRows(all.data, tsf)
    ## the test set was everything that wasn't the training set...
```

```
testRows <- testSetRows(seq(1:nrow(all.data)),trainRows)

## load the independent var's for training and test sets
data.train <- all.data[trainRows,]
data.test <- all.data[testRows,]
## make sure there is at least 1 sample from each class in the
## training data set
while(!containsAllClasses(data.train,Nclass)) {
  cat("There's at least 1 class without a sample. Reselect data sets.\n")
  trainRows <- trainSetRows(all.data, tsf)
  testRows <- testSetRows(seq(1:nrow(all.data)),trainRows)
  data.train <- all.data[trainRows,]
  data.test <- all.data[testRows,]
}

N.train.samples <- nrow(data.train)
N.test.samples <- nrow(data.train)
pc <- pofc(data.train)          # probability of the class
muk <- class_means(data.train)  # class means
## calc the avg covariance matrix
avg.sig <- AVGvar(N.train.samples,Nclass,data.train,muk)
## calc the N x D matrix of classifier probabilities for training set
lda.delta.train <- deltLD(data.train[,-1],N.train.samples,
                          Nclass,pc,muk,avg.sig)
## calc the N x D matrix of classifier probabilities for testing set
lda.delta.test <- deltLD(data.test[,-1],N.test.samples,
                         Nclass,pc,muk,avg.sig)

classes.train <- classifyOuts(t(lda.delta.train))
classes.test <- classifyOuts(t(lda.delta.test))

## tally results
T.train <- as.matrix(t(data.train[,1]))
T.test <- as.matrix(t(data.test[,1]))
r.train <- r.train + classAccuracy2(classes.train, T.train, Nclass)
trial.test <- classAccuracy2(classes.test, T.test, Nclass)
r.test <- r.test + trial.test
correct.test <- sum(trial.test[,3])
## save the pc, muk, and avg.sig with the best test results
if(i == 1)  {
  correct.best <- correct.test
  best.lda <- list(pc=pc,muk=muk,avg.sig=avg.sig)
  data.train.best <- data.train
}
if(i > 1 && correct.test > correct.best) {
  correct.best <- correct.test
```

```
      best.lda <- list(pc=pc,muk=muk,avg.sig=avg.sig)
      save(best.lda,file=name.lda)
      data.train.best <- data.train
    }
cat("completed trial ",i,"\n")
  }
cat("preparing to run trials with the best lda parms obtained so far...\n")
  # now run best.trials number of trials with the best lda parms

  r.train <- matrix(0,Nclass,3)
  r.test <- matrix(0,Nclass,3)
  tsf <- 0.5  # reset to randomly take half of the data to test

  for(j in 1:best.trials) {

    ## make sure there is at least 1 sample from each class in the
    ## training data set
    while(!containsAllClasses(data.train,Nclass)) {
      cat("There's at least 1 class without a sample. Reselect data sets.\n")
      trainRows <- trainSetRows(all.data, tsf)
      testRows <- testSetRows(seq(1:nrow(all.data)),trainRows)
      data.train <- all.data[trainRows,]
      data.test <- all.data[testRows,]
    }
    cat("got valid data sets, running trial",j," with the best LDA parms\n")
    data.train <- data.train.best  # use the set that got you the best net

    N.train.samples <- nrow(data.train)
    N.test.samples <- nrow(data.train)

    pc <- best.lda$pc            # probability of the class
    muk <- best.lda$muk          # class means
    avg.sig <- best.lda$avg.sig  # avg covariance
    ## calc the N x D matrix of classifier probabilities
    lda.delta.train <- deltLD(data.train[,-1],N.train.samples,
                              Nclass,pc,muk,avg.sig)
    lda.delta.test <- deltLD(data.test[,-1],N.test.samples,
                             Nclass,pc,muk,avg.sig)
    ## classify the results
    classes.train <- classifyOuts(t(lda.delta.train))
    classes.test <- classifyOuts(t(lda.delta.test))

    ## tally the results
    r.train <- r.train + classAccuracy2(classes.train, T.train, Nclass)
    trial.test <- classAccuracy2(classes.test, T.test, Nclass)
    r.test <- r.test + trial.test
  } # close trials with the best lda parms

  ## class column needs to be recovert after getting summed
  r.train[,1] <- r.train[,1]/best.trials
  r.test[,1] <- r.test[,1]/best.trials
  ## add the accuracy column in the results matrix
  r.train <- cbind(r.train, accuracy=0)
  r.test <- cbind(r.test, accuracy=0)
  ## make sure you don't divide by zero when calc'ing accuracy
  for(j in 1:Nclass) {
    if(r.train[j,2] == 0)
      r.train[j,4] <- 0
    else
      r.train[j,4] <- r.train[j,3]/r.train[j,2]

    if(r.test[j,2] == 0)
      r.test[j,4] <- 0
    else
      r.test[j,4] <- r.test[j,3]/r.test[j,2]
  }

  results <- list(r.train=r.train,r.test=r.test,best.lda=best.lda)
  results
}
```

The function `LDAclassifier` calls several helper functions that were described earlier [5] and listed in section 8. What is different here is the call to the function `containsAllClasses` in the while-loops. This function returns TRUE if the training set contains samples from all the classes. It returns FALSE if one or more classes are not represented in the training set. This is discussed further in the section 7.

5 Neural Network Classification of Gap Data

5.1 Implementation of NNC

As descibed in an earlier work [6], the neural network (NN) used for regression was modified for classification by minimizing the negative natural log of the likelihood of the data (versus of the squared error in the regression case). The new function to be minimize is shown in equation (3).

$$-l(v,w) = -\sum_{n=1}^{N}\sum_{k=1}^{K-1} t_{i,k} \ln y_{i,k} \tag{3}$$

where $y_{i,k}$ is the normalized output from the regression NN as shown in equation (4).

$$y_{j,n} = \frac{e^{a_{j,n}}}{\sum_{K}^{k=1} e^{a_{k,n}}} \tag{4}$$

In equation (4), the a terms are the actual pre-normalized outputs from the regression NN.

5.2 R code for NNC

The R function `NNclassifier` was used to select the best classifier net out of 100 iteration and is shown below. The results were tallied over 30 trials to get a reading on the overall average accuracy.

```
##############################################################################
# Returns the results of the best NN classifier over 100 iterations. Best
# nets results are then averaged over 30 iterations.
#
# data.train & data.test are N x (D+1) with classes in the first column
# Assumes that data is normalized before being passed in
# tsf = training set fraction
##############################################################################
NNclassifier <- function(wPenalty=0,nIterations=1000,
                         nHiddens=10,weightPrecision=0,
                         errorPrecision=0.0001,
                         all.data=XT,tsf=0.50,ntrials=100,
                         best.trials=30,name.net="net.default")
{

  Nclass <- length(unique(all.data[,1])) # calc the number of classes

  correct.test <- 0
  r.train <- matrix(0,Nclass,3)
  r.test <- matrix(0,Nclass,3)
  for(i in 1:ntrials) {

    ## establish the random set of training rows based on the tsf
    trainRows <- trainSetRows(all.data, tsf)
    ## the test set was everything that wasn't the training set...
    testRows <- testSetRows(seq(1:nrow(all.data)),trainRows)

    ## load the independent var's for training and test sets
    data.train <- all.data[trainRows,]
    data.test <- all.data[testRows,]
    ## make sure there is at least 1 sample from each class in the
    ## training data set (not an issue for NN, but do anyway for
    ## consistent comparison to LDA)
    while(!containsAllClasses(data.train,Nclass)) {
      cat("There's at least 1 class without a sample. Reselect data sets.\n")
      trainRows <- trainSetRows(all.data, tsf)
      testRows <- testSetRows(seq(1:nrow(all.data)),trainRows)
      data.train <- all.data[trainRows,]
      data.test <- all.data[testRows,]
    }

    ## remove the classes and transpost to get the data into rows
    ## so the NN can handle it ...
    X.train <- t(as.matrix(data.train[,-1]))
    X.test <- t(as.matrix(data.test[,-1]))

    T.train <- t(as.matrix(data.train[,1]))
    T.test <- t(as.matrix(data.test[,1]))
    ## create indicator variables
    Tiv.train <- makeIndicatorVars(T.train)
    Tiv.test <- makeIndicatorVars(T.test)

    ## train the network
    X <- X.train
    T <- Tiv.train
    net <- nnTrainSCG(X,T,nHiddens,wPenalty,0.5,
                      nIterations=nIterations,
                      weightPrecision=weightPrecision ,
                      errorPrecision=errorPrecision)

    ## compute the output probabilities
    outs.train <- nnOutput(net,X.train)
    outs.test <- nnOutput(net,X.test)
    # now classify the output
    classes.train <- classifyOuts(outs.train)
    classes.test <- classifyOuts(outs.test)

    ## tally
    r.train <- r.train + classAccuracy2(classes.train, T.train, Nclass)
    trial.test <- classAccuracy2(classes.test, T.test, Nclass)
    r.test <- r.test + trial.test
    correct.test <- sum(trial.test[,3])
    ## save the net with the best test results
    if(i == 1)  {
      correct.best <- correct.test
      best.net <- net
      data.train.best <- data.train
    }
    if(i > 1 && correct.test > correct.best) {
      correct.best <- correct.test
      best.net <- net
      save(best.net,file=name.net)
      data.train.best <- data.train
    }
cat("completed trial ",i,"\n")
  }
  ## after exiting this loop, we have the best net over ntrials for a
  ## given tsf
```

```
cat("preparing to run trials with the best net obtained so far...\n")
  ## now run best.trials number of trials with the best net

  r.train <- matrix(0,Nclass,3)
  r.test <- matrix(0,Nclass,3)
  tsf <- 0.5  # reset to randomly take half of the data to test

  for(j in 1:best.trials) {

    ## make sure there is at least 1 sample from each class in the
    ## training data set (not an issue for NN, but do anyway for
    ## consistent comparison to LDA)
    while(!containsAllClasses(data.train,Nclass)) {
      cat("There's at least 1 class without a sample. Reselect data sets.\n")
      trainRows <- trainSetRows(all.data, tsf)
      testRows <- testSetRows(seq(1:nrow(all.data)),trainRows)
      data.train <- all.data[trainRows,]
      data.test <- all.data[testRows,]
    }
    cat("got valid data sets, running trial",j," with the best net\n")
    data.train <- data.train.best  # use the set that got you the best net

    ## remove the classes and transpos3 to get the data into rows
    ## so the NN can handle it...
    X.train <- t(as.matrix(data.train[,-1]))
    X.test <- t(as.matrix(data.test[,-1]))

    T.train <- t(as.matrix(data.train[,1]))
    T.test <- t(as.matrix(data.test[,1]))

    Tiv.train <- makeIndicatorVars(T.train)
    Tiv.test <- makeIndicatorVars(T.test)

    # compute the output probabilities
    outs.train <- nnOutput(best.net,X.train)
    outs.test <- nnOutput(best.net,X.test)
    # now classify the output
    classes.train <- classifyOuts(outs.train)
    classes.test <- classifyOuts(outs.test)

    # tally
    r.train <- r.train + classAccuracy2(classes.train, T.train, Nclass)
    trial.test <- classAccuracy2(classes.test, T.test, Nclass)
    r.test <- r.test + trial.test
  } # close trials with the best net
  r.train[,1] <- r.train[,1]/best.trials
  r.test[,1] <- r.test[,1]/best.trials
  r.train <- cbind(r.train, accuracy=0)
  r.test <- cbind(r.test, accuracy=0)

  # calc the mean accuracy
  for(j in 1:Nclass) {
    r.train[j,4] <- r.train[j,3]/r.train[j,2]
    r.test[j,4] <- r.test[j,3]/r.test[j,2]
  }

  results <- list(r.train=r.train,r.test=r.test,best.net=best.net)
  results
}
```

All the helper functions called by NNclassifer have been described earlier [6] except for `classAccuracy2` which is a modified version of the previously describe `classAccuracy` function. The function `classAccuracy2` is used to simply assist in the tallying of results.

6 Results

6.1 Comparing LDA with NNC on the Three Selected Dow Components

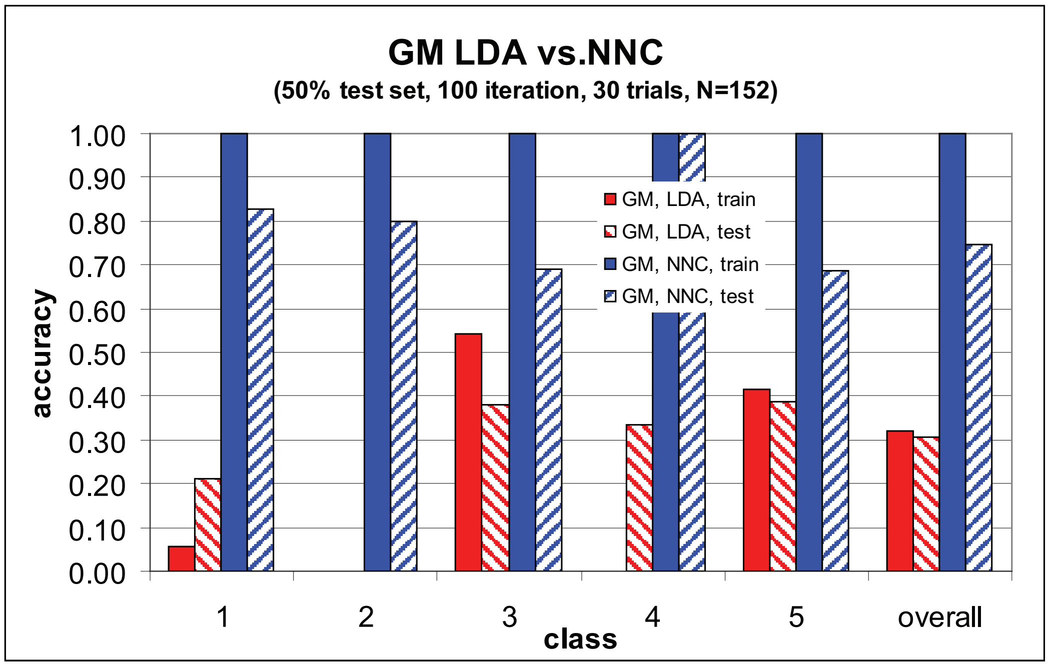

Figure 2: LDA versus NNC (10 hidden units) results for GM

Figure 2 shows the results for General Motors (GM). GM had the most gaps of 1% or greater over the specified time period (152). LDA did a rather poor job while the NNC did rather well. Reasons for this disparity are discussed in the section 7. Suprisingly, the NNC would fit itself perfectly to the training data using only 10 hidden units and did better than expected on the test data partition.

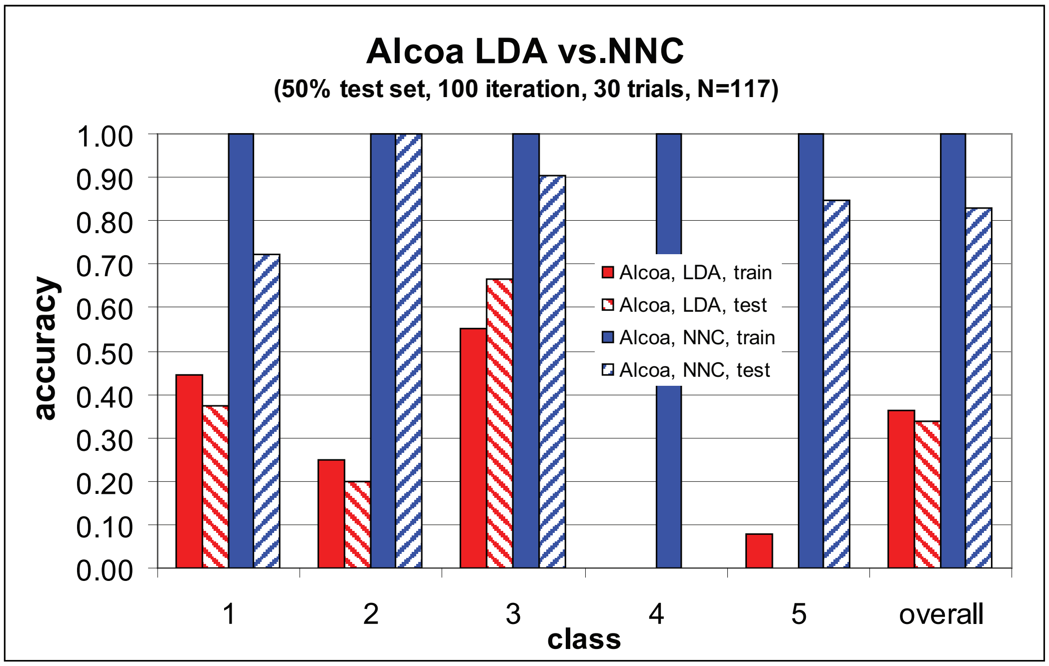

Figure 3: LDA versus NNC (10 hidden units) results for Alcoa

Figure 3 shows the results for Alcoa. Alcoa had the second highest number of gaps of 1% or greater over the specified time period (117). As with GM, LDA did a rather poor job while the NNC did rather well except for class 4. As seen with GM, the Alcoa NNC would fit itself perfectly to the training data using only 10 hidden units and also did well on the test data partition.

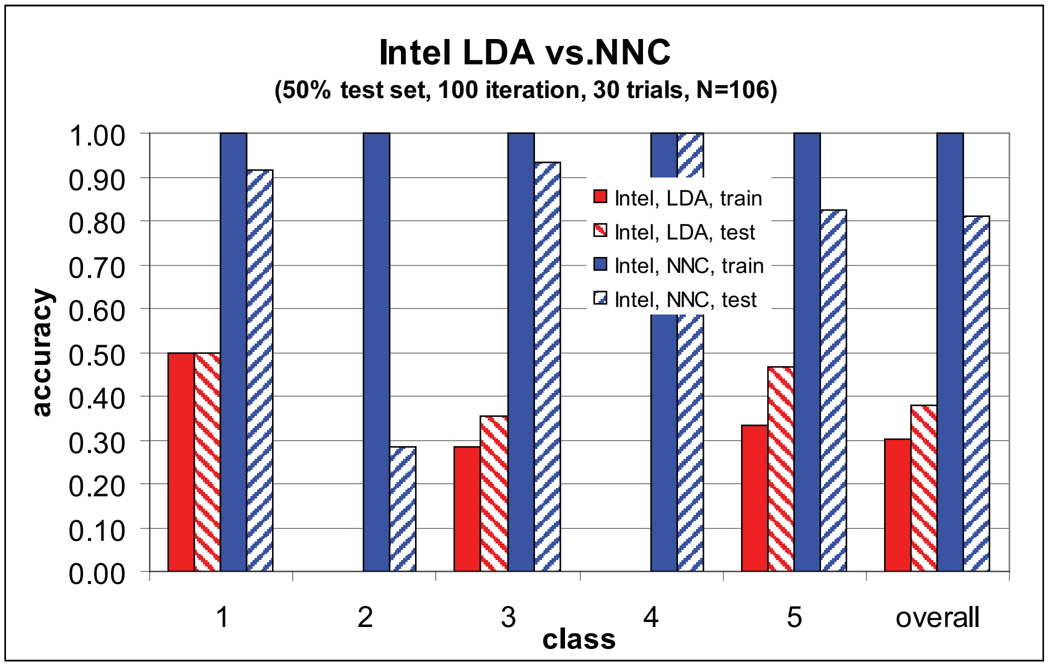

Figure 4: LDA versus NNC (10 hidden units) results for Intel

Figure 4 shows the results for Intel. Intel had the third highest number of gaps of 1% or greater over the specified time period (106)of the 30 Dow components. As with GM and Alcoa, LDA did a rather poor job while the NNC did rather well. Unlike GM or Alcoa however, the Intel NNC did poorly predicting class 2 trajectories. As seen with GM and Alcoa, the Intel NNC would also fit itself perfectly to the training data using only 10 hidden units and also did well overall especially predicting classes 1, 3, and 4.

6.2 A Personal Curiosity: Using One Stocks NNC to Make Predictions Based on Another Stocks Data

Just out of curiosity, the best GM NNC was tested on the Intel data set. These results are shown in Figure 5.

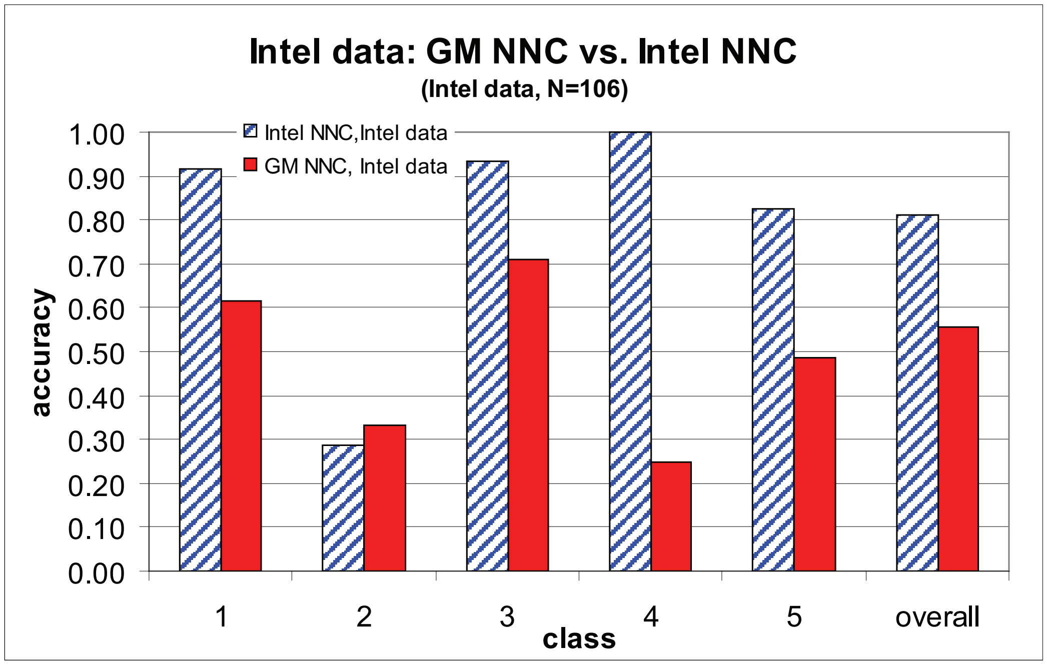

Figure 5: Comparing GM's NNC with Intel's NNC using 100% of the Intel data as input to GM NNC while using a 50% testing partition for the Intel NNC.

As expected, the Intel NNC did much better than the GM NNC on the Intel data which implies that even though the price data is normalized prior to classification, each stock appears to have its own 'personality'. At the same time, it is interesting that class 3 predictions were still above 70% which may imply that there are some underlying behavioral similarities that drive stocks in general, but this assertion is a large topic in and of itself and well beyond the scope of this work.

7 Discussion

This study was motivated by the need to discover a workable swing trading model. Swing trades are generally defined as trades that are exited after several days or weeks. This type of trading style is of practical interest to those that do not wish to or have the time to continuously monitor the markets throughout the day. Since markets show fractal characteristics [7], pricing patterns that are seen by day-traders tend to show up in longer time frames implying the same potential for profit or loss exists in any chosen time frame.

Given the potential use of this model, it is important to caution readers that a potentially profitable trading **concept** is not the same as a profitable trading **system**. Many books have been written and many courses have been created that focus on system creation. Turning a sound trading concept into a system involves many factors including: risk analysis, exit strategy, position sizing, money management, as well as factors such as psychological and emotional discipline and maturity [1, 3, 4]. None of these factors are addressed in this work which is why a note of caution is called for. That being said, attention can now be turned back to the model.

The reader should note that most models that deal with trading treat the data in a time-series fashion. The analysis in this work however assumes that the pre-gap price information forms a pattern that can be used to predict the pricing trajectory following a gap. In this sense, the pre-gap pricing data is used in a way that is similar to the classification of digital digits with one key difference. With the digit image data, the images definitely have a particular meaning (each image represents an integer from 0 through 9). With the pre-gap pricing data, the pattern may **correlate** to a post-gap pricing pattern, but in and of itself can not be shown to have any meaning beyond the ability to draw a reasonable correlation.

For the sake of analysis, it was assumed that pre-gap pricing pattern holds predictive information about future price trajectories. This fundamental assumption may be false. This is why any past correlations found with this model may not hold at all for future prediction which is yet another reason one should be very careful in attempting to use this model as part of an actual trading system.

As shown eariler, the LDA classifier did a poor job of predicting the price trajectory following a gap. It is hypothesized that this is due to the complex shape of the multi-dimensional surface that is needed to separate the boundaries of the classes. If this hypothesis is correct, it would be difficult for a multi-dimensional hyperplane to separate the class boundaries. The relative success of the NNC might be described from this perspective as well. Since the NNC can create complex multi-dimensional surfaces, it has much more flexibility in defining classification boundaries.

A 1% gap was chosen to generate enough samples for a reasonable analysis. If a swing trader was attempting to profit from a gap-fill strategy, a larger than 1% change in price would be needed to make the trade feasibile. The collected data was then partitioned into 50% training and 50% testing. Originally, it was hoped that an 80% training partition could be used, but this was revised for two reasons. First, an 80% training partition often resulted in having no class 4 (which only made up about 5% of the sample population) samples in the remaining 20% testing set. Second, an 80% training partition left the testing partion rather sparse making comparisons difficult.

There were two fundamental challenges with this project. The first was preparing the data. The way this analysis was designed required that the original downloaded data be restructured so that 10 days of pricing and volume data be put in one row to create a single sample (5 days pre-gap and 5 days post gap). This required a custom parser to get the data into a form that the classifiers could use.

The second challenge was a subtle limitation in the original LDA classification code that required that all classes have at least one sample in the training set. This was a tricky situation to deal with because the code was functioning correctly. When an LDA training set was selected with one or more of the classes not being represented in the sample, problems arose in attempting to calculate the class means vector and the average covariance matrix. To address this, the function `containsAllClasses` was written which took in the data set and the number of classes and returned TRUE if the set contained at least 1 sample in each class, FALSE if it did not. This function was then put inside a while loop which forced the classifier function to reselect its training data set until all classes were represented in the sample.

Future work will likely focus on some portion of the following five areas:

- Apply the NNC to smaller, more volitile stocks.
- Fine tune the NNC for better prediction performance.
- Examine principal components to determine which components have the largest impact on the prediction of pricing patterns.

- Back test and paper trade the models using various stop loss patterns to assess profit potential.
- Explore whether an NNC can reasonably predict if/when a gap will occur.

8 Additional R code used in this work

Any code that was used in this work, but was not listed in this paper was listed in earlier works [5, 6]. The NNC code is not listed below because it is exactly the same as before [6]. The LDA code is listed because some small modifications were made to some of the related functions. All of the code used for pre-analysis data manipulation are included in this section.

```
##############################################################################
# Reads in the cvs data files the were previously downloaded from:
# http://finance.yahoo.com/
# Creates a list of 30 dataframes: one df per stock
##############################################################################
read.dow30 <- function(gap.trigger=0.01) {
  dow30 <- c()
  # loop through the csv data files that were previousl downloaded
  # file names were of the form: downnn.csv where nn=01 for Alcoa, etc.
  # (the current dow components as of 5/1/08 in alphabetical order)
  for(i in 1:30) {

    if(i < 10)
      fid <- paste("0",i,sep="")
    else
      fid <- paste(i,sep="")

    file.name <- paste("dow",fid,".csv",sep="")
    dow.file <- read.csv(file=file.name)
    # index the dataframe so it can be reordered properly
    index <- nrow(dow.file):1
    dow.file <- cbind(i=index, dow.file)
    # data is read with most recent at top, so we need to reorder
    # so that most recent is at the bottom
    dow.file <- dow.file[order(dow.file$i, decreasing=FALSE),]
    # remove the adjusted close column because we aren't using it
    dow.file <- dow.file[,-8]
    # add gap columns
    dow.file <- cbind(dow.file, gap.size=0)   # col 8
    dow.file <- cbind(dow.file, gap.frac=0)   # col 9
    dow.file <- cbind(dow.file, is.gap=FALSE) # col 10
    # calc gap, percent gap, and whether the gap is big enough to
    # be considered for analysis
    for(j in 6:nrow(dow.file)) {
      dow.file[j,8] <- (dow.file[j,3]-dow.file[j-1,6])
      dow.file[j,9] <- dow.file[j,8]/dow.file[j-1,6]
      dow.file[j,10] <- (abs(dow.file[j,9]) >= gap.trigger)
    }
    # add the particular dow components dataframe to a list
    dow30 <- c(dow30, list(dow.file))
  }

  save(dow30,file="dow30gap.data")
  return(dow30)
}

##########################################################################
# Creates the input matrices X and T to use for for classification
# X is normalized, in rows, and is the first 29 columns of return matrix
# T is the class (1-5) which is in column 30
##########################################################################
makeInputsToClassify <- function(table=dow30[[16]]) {
  table.rows <- nrow(table)  # number of rows in the table
  n.gaps <- countGaps(table)
  inputXT <- NULL
  input.index <- 1

  # load the data matrix
  for(i in 6:(table.rows-4)) {
    if(table[i,10]) {  # if there is a gap, look back 5 trading days and
                       # start filling the table with info on the gap
      postpmin <- min(table[i:(i+4),5])    # min price following gap
      postpmax <- max(table[i:(i+4),4])    # max price following gap
      ppclose  <- table[(i+4),6]           # close price last day of period
      p1 <- table[(i-1),6]                 # pre-gap close
      p2 <- table[i,3]                     # post-gap open
      # assign the gap classification to the data
      class <- classifyGap(p1,p2,postpmin,postpmax,ppclose)

      # stack to prices OHLC for the 5 days followed by the volumes
      # for the 5 days
      inputXT <- rbind(inputXT, c(class=class,
                       table[(i-5),(3:6)],table[(i-4),(3:6)],
                       table[(i-3),(3:6)],table[(i-2),(3:6)],
                       table[(i-1),(3:6)],
                       pmin=postpmin,pmax=postpmax,finish=ppclose,
                       table[(i-5),7],table[(i-4),7],table[(i-3),7],
                       table[(i-2),7],table[(i-1),7],
                       gap=table[i,8])
                       )
      input.index <- input.index + 1
    }
  }

  # now we need to normalize the prices, volume, and gap data
  # mean.matrix will be 1 x (20 pre-gap prices + 3 post gap prices +
  # 5 pre-gap volumes + 1 gap) or 1 x 29
  temp.matrix <- matrix(as.numeric(inputXT[,-1]),nrow(inputXT),
                                                         (ncol(inputXT)-1))
  inputNorms <- standardize(temp.matrix,returnParms=TRUE)
  meanNorms <- inputNorms$means
  stdevNorms <- inputNorms$stdevs
  inputNorms <- inputNorms$data

  # build the final version of the input data table
  class.matrix <- matrix(as.numeric(inputXT[,1],nrow(inputXT),1))
  temp <- cbind(class.matrix, inputNorms)

  colnames(temp) <- c("class",
                      "O5","H5","L5","C5","O4","H4","L4","C4",
                      "O3","H3","L3","C3","O2","H2","L2","C2",
                      "O1","H1","L1","C1",
                      "pmin","pmax","pfinish",
                      "V5","V4","V3","V2","V1","gap")

  as.matrix(temp)
}
##############################################################################
# Returns TRUE if the gap was filled during the specified period following
# a gap.  Returns FALSE if the gaps was not filled in the specified period
# following a gap.
# p1 - pre-gap closing price
# p2 - post-gap opening price
# postpmin - minimum price during the specified period following the gap
# postpmax - maximum price during the specified period following the gap
#
# PRECONDITION: p1 != p2 and postpmax >= postpmin
##############################################################################
gapFill <- function(p1,p2,postpmin,postpmax) {
  if((p2-p1) > 0)  # gap up
    retval <- (postpmin <= p1)
  else             # gap down
    retval <- (postpmax >= p1)

  retval
}
```

```
###############################################################################
# Returns TRUE if ppclose (price at the close of a period) is ABOVE the
# the gap zone.  Returns FALSE if ppclose is below or in the gap zone.
# p1 - pre-gap closing price
# p2 - post-gap opening price
# ppclose - closing price on the last day of the specified period
#
# PRECONDITION: p1 != p2
###############################################################################
finishAbove <- function(p1,p2,ppclose) {
  retval <- (ppclose > max(p1,p2))
  retval
}

###############################################################################
# Returns TRUE if ppclose (price at the close of a period) is BELOW the
# the gap zone.  Returns FALSE if ppclose is above or in the gap zone.
# p1 - pre-gap closing price
# p2 - post-gap opening price
# ppclose - closing price on the last day of the specified period
#
# PRECONDITION: p1 != p2
###############################################################################
finishBelow <- function(p1,p2,ppclose) {
  retval <- (ppclose < min(p1,p2))
  retval
}

###############################################################################
# Returns TRUE if ppclose (price at the close of a period) is IN the
# the gap zone.  Returns FALSE if ppclose is above or below the gap zone.
# p1 - pre-gap closing price
# p2 - post-gap opening price
# ppclose - closing price on the last day of the specified period
#
# PRECONDITION: p1 != p2
###############################################################################
finishIn <- function(p1,p2,ppclose) {
  retval <- !(finishAbove(p1,p2,ppclose))
  retval <- retval && (!finishBelow(p1,p2,ppclose))
  retval
}

################################################################################
# Returns TRUE if there is at least one sample from every class in the data
# set. Returns FALSE if at least one class does not have at least 1 sample
# associated with it.
```

```
#
# data.set - N x (D+1) data matrix that has the class in the first column
# nclass - the number of classes in a data set
#
# PRECONDITION: classes are enumerated by integers starting from 1 to nclass
####################################################################################
containsAllClasses <- function(data.set,nclass)  {
  has.class <- TRUE
  for(k in 1:nclass) {
      classrows <- which(data.set[,1]==k) # find rows for particular class
      has.class <- has.class && (length(classrows) >= 1)
  }
  has.class
}

##########################################################################
# Calculates P(C) - xtrain must be a |n x d+1| matrix where n is the number
# of training data samples and d is the number of dimesions.  The first
# column of xtrain must be the class designation which must be an integer
# from 1 to k, where k = number of classes. Columns 2 through d+1 define
# each dimensional coordinate of a sample vector of length d.
#
# The function returns vector/list of size k corresponding to P(C). The
# first value will be P(C=1) followed by P(C=2), etc. up to P(C=K)
pofc <- function(xtrain) {
   nclass <- length(unique(xtrain[,1]))  # number of classes in training set
   pc <- c()
   for(class in 1:nclass)
      pc <- c(pc, length(which(xtrain[,1]==class))/nrow(xtrain))

   pc <- as.matrix(pc)
}

####################################################################################
# Calulates the mean of each class from the training data xtrn. This function
# returns a k(# of classes) x d(dimensions) matrix.  The first row is the
# mean vector for class 1, the second row for class 2 etc...
# The first column of xtrn must be the class designation which must be an
# integer from 1 to K, where K = number of classes. Columns 2 through d+1
# define each dimensional coordinate of the mean vector of length d for
# each class
#
# PRECONDITION: THERE MUST BE AT LEAST ONE SAMPLE FROM EACH CLASS!!!
####################################################################################
class_means <- function(xtrn) {
   ndim <- ncol(xtrn) - 1  # number of dimensions in each data sample
   nclass <- length(unique(xtrn[,1])) # calc the number of classes
   muk <- matrix(0,nclass,ndim)  # initialize matrix of means

   for(k in 1:nclass) {
      classrows <- which(xtrn[,1]==k) # find rows for particular class
#cat("within class_means: k=",k," length(classrows)=",length(classrows),"\n")
#browser()
      # start at col 2 because the class is in the first column
      xclass <- matrix(xtrn[classrows,2:ncol(xtrn)],length(classrows),
                                                   ,byrow=TRUE)

      for(idim in 1:ndim)
         muk[k,idim] <- mean(xclass[,idim])
   }
   muk <- as.matrix(muk)
}

##########################################################################
# Returns the single average covariance matrix for LDA analysis.  If
# data is 1-D, the function returns average variance
# ((standard deviation)^2).
#
# The first column of xtrn must be the class designation which must be an
# integer from 1 to k, where k = number of classes. Columns 2 through d+1
# define each dimensional coordinate of a sample vector of length d.
# muk is a k(# of classes) x d(dimensions) matrix of class means - each
# row being a vector containing the mean of the class = row number
#
# PRECONDITION: THERE MUST BE AT LEAST ONE SAMPLE FROM EACH CLASS!!!
##########################################################################
AVGvar <- function(Ntest, Kclass, xtrn, muk) {
   d <- ncol(xtrn) - 1
   sqdiff <- diag(0,d)
   for(k in 1:Kclass) {
      classrows <- which(xtrn[,1]==k)  # rows in class k

      # start at col 2 because the class is in the first column
      xclass <- matrix(xtrn[classrows,2:ncol(xtrn)],length(classrows),
                                                   ,byrow=TRUE)

      Nk <- length(classrows)  # number of samples in the class

      for(s in 1:Nk) {
         x <- matrix(xclass[s,],1,d,byrow=TRUE)
         mu <- matrix(muk[k,],1,d,byrow=TRUE)

         sqdiff <- sqdiff + (t(x-mu)%*%(x-mu))
      }
   }
   avgvar <- sqdiff/(Ntest-Kclass)
   avgvar
}

########################################################################
# makes the classification indicator variable for Y data layed out in
# columns.  Y is a 1 x N matrix of the classes. Returns a K x N matrix
# of indicators where earch column contains all 0's but a single 1.
# the row number of the 1 indicates the class.
########################################################################
makeIndicatorVars <- function(Y,classes=sort(unique(Y, MARGIN=2))) {
  ## specify classes if some might be missing from Y
  N <- ncol(Y)
  K <- length(classes)
  leftside <- matrix(Y,K,N,byrow=TRUE)
  rightside <- matrix(classes,K,N)

  return((leftside == rightside) * 1)
}

########################################################################
# Calculates the discriminant for LDA:
# Function returns a N(number of samples) x k(number of classes) matrix,
# where the first column is the delta(x) for the first class and so on
# to the last column which is delta(x) for class k.
#
# x is a N(number of samples) x d(dimensions) matrix defining the sample
# data.  nsamp is the number of samples in the test set being analyzed.
# nclass is the number of classes the function is to discriminate against.
# pc is a list of length k(number of classes) that holds the probablity of
# each class.
# muk is k(number of classes) x d(dimensions) matrix holding the means
# for each class generated from the training data.
# var is the average covariance matrix derived from the covarience matrices
# for each class generated from the training data.
#
deltLD <- function(x,nsamp,nclass,pc,muk,var) {
   x <- as.matrix(x)
   #print(dim(var))
   d <- ncol(x)
   nsamp <- nrow(x)
   deltx <- matrix(0,nsamp,nclass)  # initialize output
   for(k in 1:nclass) {
      invSigma <- solve(var)
      for(i in 1:nsamp) {
            xs <- matrix(x[i,1:d],1,d)  # an x sample
            mu <- matrix(muk[k,],1,d,byrow=TRUE)

            p1 <- xs %*% invSigma %*% t(mu)
            p2 <- 0.5*(mu%*%invSigma%*%t(mu))

            deltx[i,k] <- p1 - p2 + log(pc[k])
        }
    }
#print(dim(deltx))
    deltx
}

######################################################################
### Function to standardize inputs.  Assumes data samples are in rows,
### so the function calc's means & stdevs per column
######################################################################

standardize <- function(X,means=apply(X,2,mean),stdevs=apply(X,2,sd),
                        returnParms=FALSE) {
  stdevs[stdevs==0] <- 1
  D <- ncol(X)
  N <- nrow(X)
  X <- (X - matrix(rep(means,N),N,D,byrow=TRUE))/
          matrix(rep(stdevs,N),N,D,byrow=TRUE)
  if (returnParms)
    list(data=X,means=means,stdevs=stdevs)
  else
    X
}

###########################################################################
# Returns a randomly selected subset of rows from the original input
# dataframe.
# orig - original (complete) dataset (dataframe), assumed that samples
#        are stored in rows of a 2-D table
# fraction - the fraction of the original dataset used for training
#            (e.g. 0.8 is 80%)
trainSetRows <- function(orig, fraction) {
  randorder <- sample(nrow(orig))  # create set of randomly selected rows
  nTrain <- round(nrow(orig)*fraction) # calc # of rows in training set
  trainRows <- randorder[1:nTrain]  # 1st nTrain rows in rand order
  trainRows
}

###########################################################################
# returns a vector of the row numbers of test set
```

```
testSetRows <- function(allRows, trainRows) {
  testRows <- setdiff(allRows, trainRows)
  testRows
}
```

References

[1] Tharp, V.K., *Trade Your Way To Financial Freedom*, McGraw-Hill, 1999.

[2] *Yahoo! Finance*, `http://finance.yahoo.com/`

[3] Tharp, V.K., Barton, D.R, Sjuggerud, S. *Safe Strategies for Financial Freedom*, McGraw-Hill, 2004.

[4] Tharp, V.K., *Peak Performance For Traders and Invesetors*, `http://www.iitm.com/products/course/peak_performance_crse.htm`

[5] Szczepaniak, M.D. *Linear, Quadratic, and Logistical Discriminant Analysis: An Overview with Examples*, March 2008

[6] Szczepaniak, M.D. *Neural Network Classification of Hand-Drawn Digits*, April 2008

[7] *Fractal Financial Markets* `http://www.economymodels.com/factalmarkets.asp`