Stats503Hw3

David Li

February 21, 2018

Data Report for Stats503 Homework 3

– Introduction: Classification –

Classification is the act of sub-dividing in multiple categories, such that observations can be within these set categories. Usually, classification seeks to be as accurate as possible as when applied to datasets appropriately such that if we were to observe a new observation we can have a idea of which category it may belong in. That is, usually we may (for instance) split a dataset into a training set that "trains" the classifier or decision algorithm with known memberships so that we may apply the classifier algorithm upon other given values for inference and/or analysis (such as the remaining part of the dataset or hypothetical input values). Since Classification procedures have existing relationships to build the algorithm upon, they are examples of so-called supervised learning.

In practice of reality, Classification is extremely useful in many contexts. For example, Classification is a common challenge to many machine-learning related problems in real life such as DNA expression, image/voice recognition, quality assurance testing, and much more. With pre-existing relationships or data classifications, we can leverage these valuable statistics to predict and analyze on potential characteristics and/or behavior of newly given inputs even in other contexts or nature. This data report is focused upon the following classification techniques (which are only a subset of many possible classification methods): Linear Discriminant Analysis, Quadratic Discriminant Analysis, Logistic Regression Model, and k-nearest-neighbor classifier.

– Data & Procedure for Demonstration –

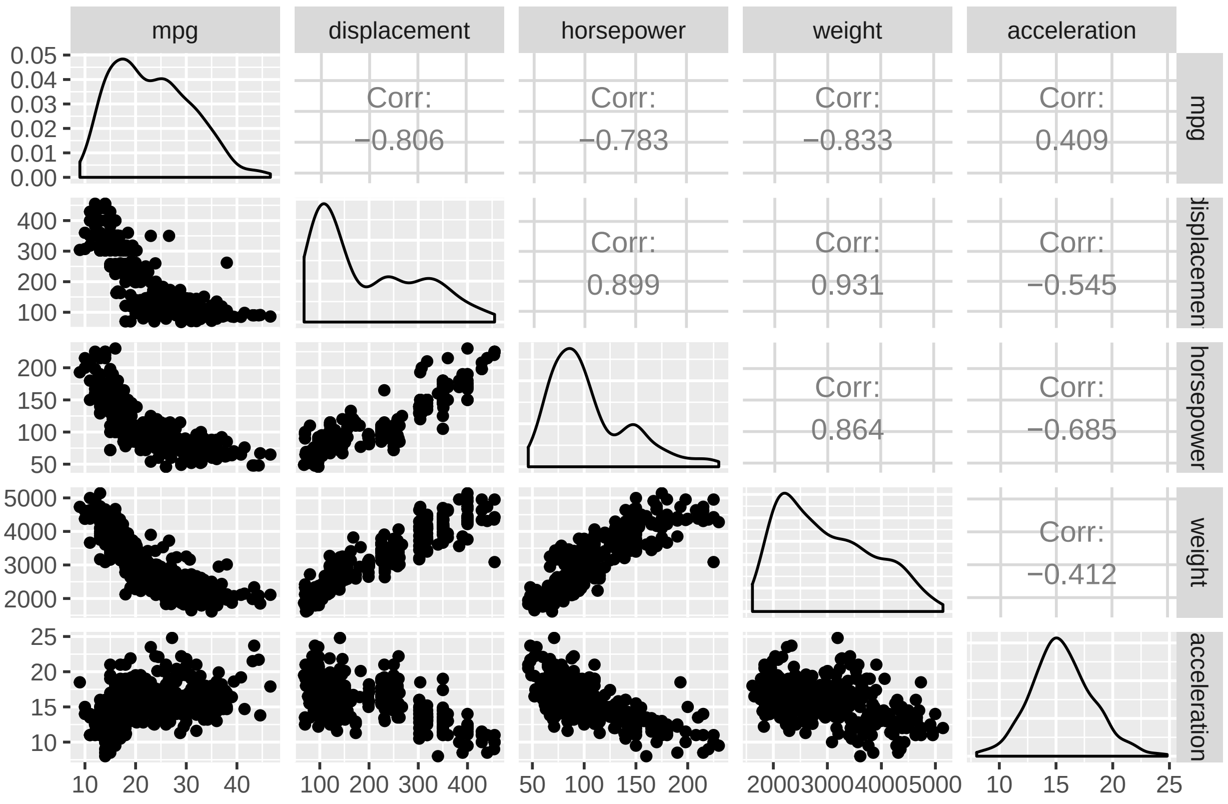
We will be doing a short analysis on the auto-mpg.data, with applications of the Classification techniques mentioned above. To compare in-depth patterns and behavior of the classification techniques, each technique will be applied to the original data as well as a standardized version of the dataset and a PCA-processed version (2 components) of the dataset. Based on the continuous predictors (MPG, Displacement, Horsepower, Weight, Acceleration), our definition of the class labeling will be Y = 1 for 5 cylinders or less, Y = 2 for 6 cylinders, and Y = 3 for all other cylinder numbers (that is 3, 4, 8). Recall that we should pay attention to the accuracy as well as the error rate of the classifier to get a sense of the strength and reliability of the classification method.

In addition to "test" our classifier, we will be splitting the dataset variations into their respective training sets and test sets with the training sets "training" the classifiers and the tests set used for experimental assessment of the classifiers. We take a quick look below:

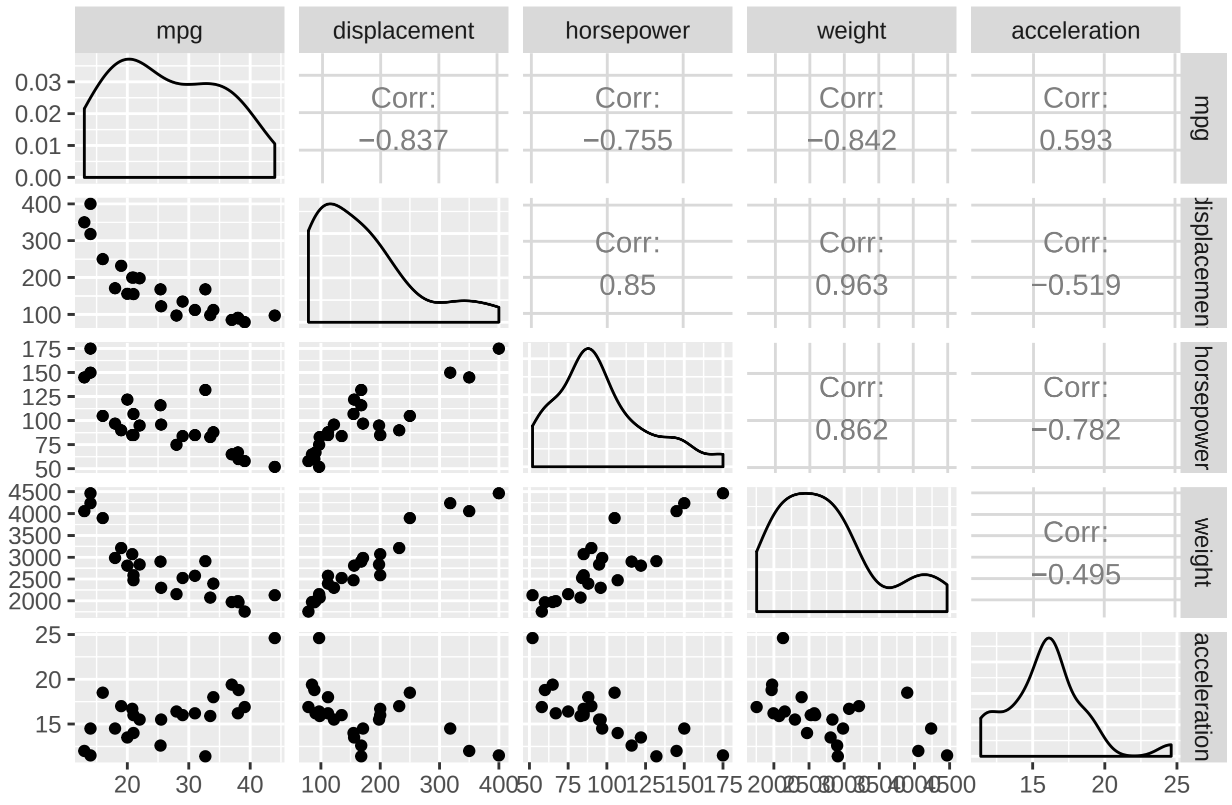
```
##       mpg         displacement     horsepower        weight
##  Min.   : 9.00   Min.   : 68.0   Min.   : 46.0   Min.   :1613
##  1st Qu.:17.00   1st Qu.:105.0   1st Qu.: 75.0   1st Qu.:2228
##  Median :22.45   Median :148.5   Median : 94.5   Median :2822
##  Mean   :23.25   Mean   :196.0   Mean   :105.0   Mean   :2992
##  3rd Qu.:29.00   3rd Qu.:302.0   3rd Qu.:129.2   3rd Qu.:3630
##  Max.   :46.60   Max.   :455.0   Max.   :230.0   Max.   :5140
##   acceleration
##  Min.   : 8.00
##  1st Qu.:13.70
##   Median :15.45
##   Mean   :15.52
##   3rd Qu.:17.12
##   Max.   :24.80
```

```
##        mpg         displacement      horsepower        weight    
##   Min.   :13.00   Min.   : 79.00   Min.   : 52.00   Min.   :1755  
##   1st Qu.:19.75   1st Qu.: 97.75   1st Qu.: 81.00   1st Qu.:2149  
##   Median :25.45   Median :155.50   Median : 89.00   Median :2581  
##   Mean   :26.42   Mean   :170.12   Mean   : 96.54   Mean   :2761  
##   3rd Qu.:33.62   3rd Qu.:200.00   3rd Qu.:109.25   3rd Qu.:3006  
##   Max.   :44.00   Max.   :400.00   Max.   :175.00   Max.   :4464  
##    acceleration 
##   Min.   :11.40  
##   1st Qu.:14.38  
##   Median :16.00  
##   Mean   :15.90  
##   3rd Qu.:16.93  
##   Max.   :24.60
```

Training set of Original Data



Test set of Original Data



These ggplot2-inspired plots generated are combined plots of various types in a plot matrix form; these were generated from the GGally package in R. It is evident that comparing the correlations and scatterplots in the training and test datasets that there are a lot of similarities. This is a good thing to see, because we want our training set to be representative and similar to our test set and vice-versa. If we saw significant differences in the scatterplots and correlations, it would be hard to make a convincing case that the strength of the classifier on the training dataset would be translatable for predicted results on the test dataset. We verify this by noticing that the shapes of corresponding scatterplots between the training set and test set all follow a similar shape, and that the correlations between the training set and test set happily do not differ by much. Supporting the GGally plots we could also look at the summary statistics for both the training and test datasets and observe that the distribution of each variable's statistics is also similar between the sets. We could create similar plots and summary statistics for the standardized and PCA-applied variations of the dataset, but we are confident in also not seeing any striking differences between the training and test datasets and thus we will omit these GGally plots and summaries for clarity.

– Linear Discriminant Analysis (LDA) –

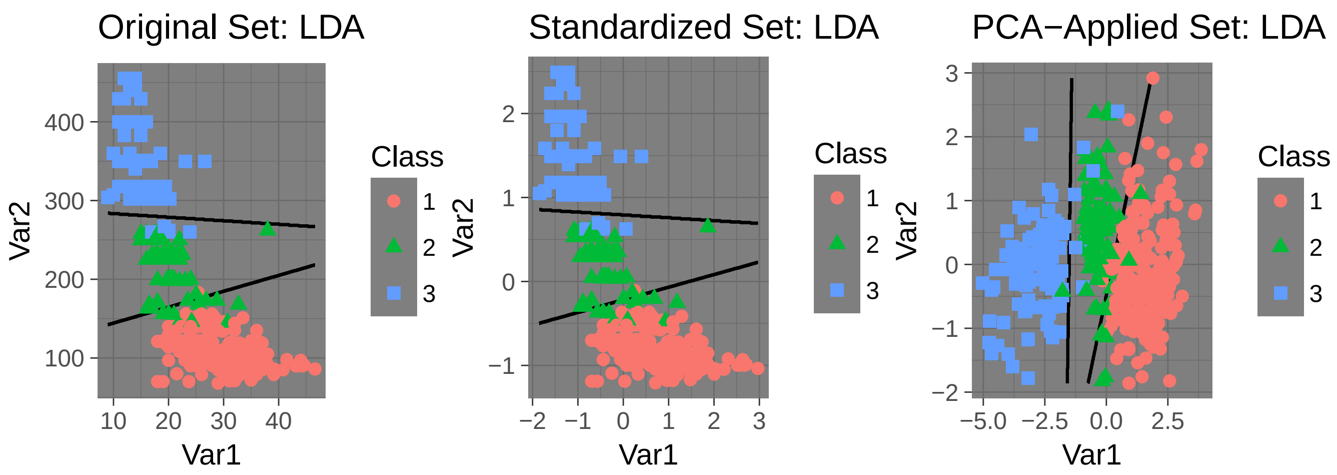
LDA is based upon the idea of computing "scores" that are simply linear combinations of the independent variables for each observation, these "scores" are used to determine what the potential response is among the response variable class. The "scores" are estimated upon various factors, such as average of all training observations from a particular class, a weighted average of sample variances for each classes, prior probabilities that an observation may be inside some class. Some assumptions of note are that the observations of each class are from a multivariate Gaussian dist. and that we can suppose that the covariance of the predictor variables are closely consistent in all levels of the response variable (in this case, which cylinder class is being used.) The advantages of LDA is that it is simpler in nature due to being a less flexible classifier though linear degree, so variance can be relatively low to other classification methods. However beware the bias-variance trade-off, as using a less flexible method can result in high bias if the uniform variance assumption is not accurate.

Table 1: LDA Classification Error Rates of Cylinder Class Prediction

	Training Set	Test Set
Original Dataset	0.0461957	0.1666667
Standardized Dataset	0.0489130	0.0833333
PCA Applied Dataset	0.0652174	0.1250000

We easily see that the error rates of the testing sets are always greater than the respective training set error rates, something usually expected since the testing dataset is much smaller than the training dataset and that the classifier was trained upon the training dataset (not the test dataset). However among the variations of the datasets, it is interesting to see that the standardized dataset has the lowest error rates while the range of error rates between training and test is greatest in the original version of the dataset.

Below is also a plot of the data projected onto the first two discriminant directions for each variation of the dataset (original, standardized, and PCA-Applied).



It's interesting to point out that the Boundary plots of each dataset version projected onto the first two discriminant cases all do a fairly great job of determining a convincing bound for the classification groups. More so, the boundary plots between the original versus the standardized dataset variations don't differ very much at all in shape; the scale of the discriminant directions is different, but this is natural from knowing that the standardized data is also a re-scaling of the original dataset. The PCA boundary plot looks like a rotation of the other two boundary plots, and still presents the same "strength" of the accuracy of the classification as the other two boundary plots.

– Quadratic Discriminant Analysis (QDA) –

QDA is similar to LDA in classification since it shares the same assumptions with LDA, with the one unlike exception that the covariance matrix is NOT the same among all of the possible classes. Therefore it turns out that we have to compute or take multiple co-variance matrixes into consideration when computing the discriminant scores. Since more predictors have to be accounted for and we no longer require a linear decision boundary, QDA is more computationally expensive than LDA. However with more predictors, QDA will tend to have lower prediction error rates compared to LDA since more predictors can lead to higher model-fitting and prediction accuracy due to non-linear classification decision boundaries. We can recommend QDA when the training set is large, where we are not as concerned about the fragility of the classifier's variance. In general, a co-variance matrix will need to estimate (p(p+1))/2 parameters for p predictors.

Table 2: QDA Classification Error Rates of Cylinder Class Prediction

	Training Set	Test Set
Original Dataset	0.0244565	0.0000000
Standardized Dataset	0.0244565	0.0000000
PCA Applied Dataset	0.0489130	0.0833333

In general, the error rates are lower compared to LDA across the board; this was an anticipated byproduct by the QDA's method of approach described earlier above due to higher accuracy power from the non-linear structure of QDA. It is interesting that the test set had a lower error rate than the training set, which was probably due to the small size of the test set. Another possible explanation is that since qda is more flexible in establishing the decision bounds, it could be overfitting based on the test data so it looks like the fit is very high; if the test dataset was larger then we would probably not encounter this coincidence as much. It is also by coincidence that the training and test error rates were identical in the original and standardized versions of the dataset, as setting a different seed for the computation also presents different results.

– Logistic Regression Model –

The Logistic Regression Model is somewhat similar to the LDA classification in that they both are based upon linear decision boundaries. When the decision boundaries are expected to be linear then both Logistic Regression and LDA can perform fairly well; in fact usually with limited differences. The catch is that LDA is more reliant and restricted upon the idea that observations are from a Gaussian distribution with a common covariance matrix across the classes, such that when this assumption is strongly valid then LDA will do a better job of classification over Logistic regression. In turn, logistic regression is more resistant when this assumption fails to hold and will show better results in those cases over LDA instead. In this sense, the method of computing the discriminant scores differs in this way and will cause LDA to be better in some situations while Logistic regression to be better in other instances.

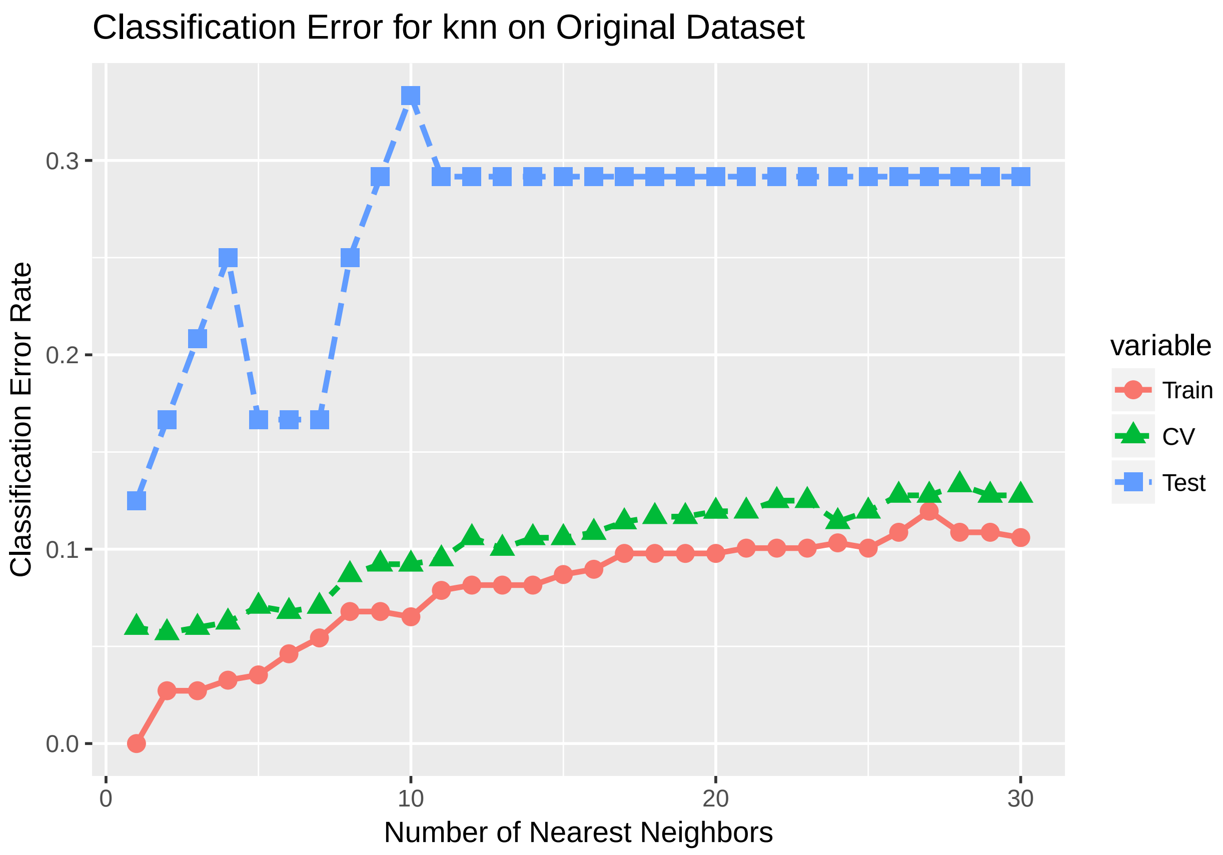
Table 3: Logistic Regression Classification Error Rates of Cylinder Class Prediction

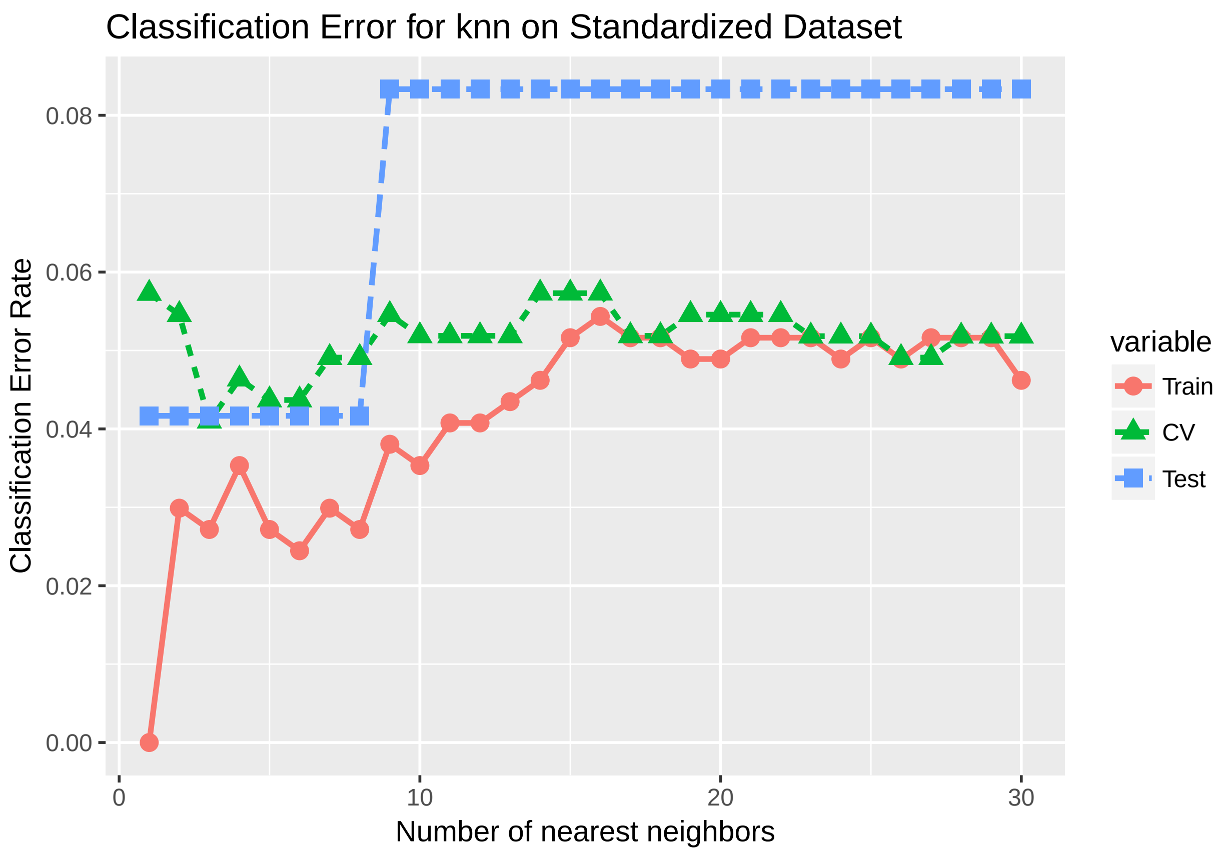
	Training Set	Test Set
Original Dataset	0.0135870	0.0416667
Standardized Dataset	0.0108696	0.0416667
PCA Applied Dataset	0.0407609	0.0833333

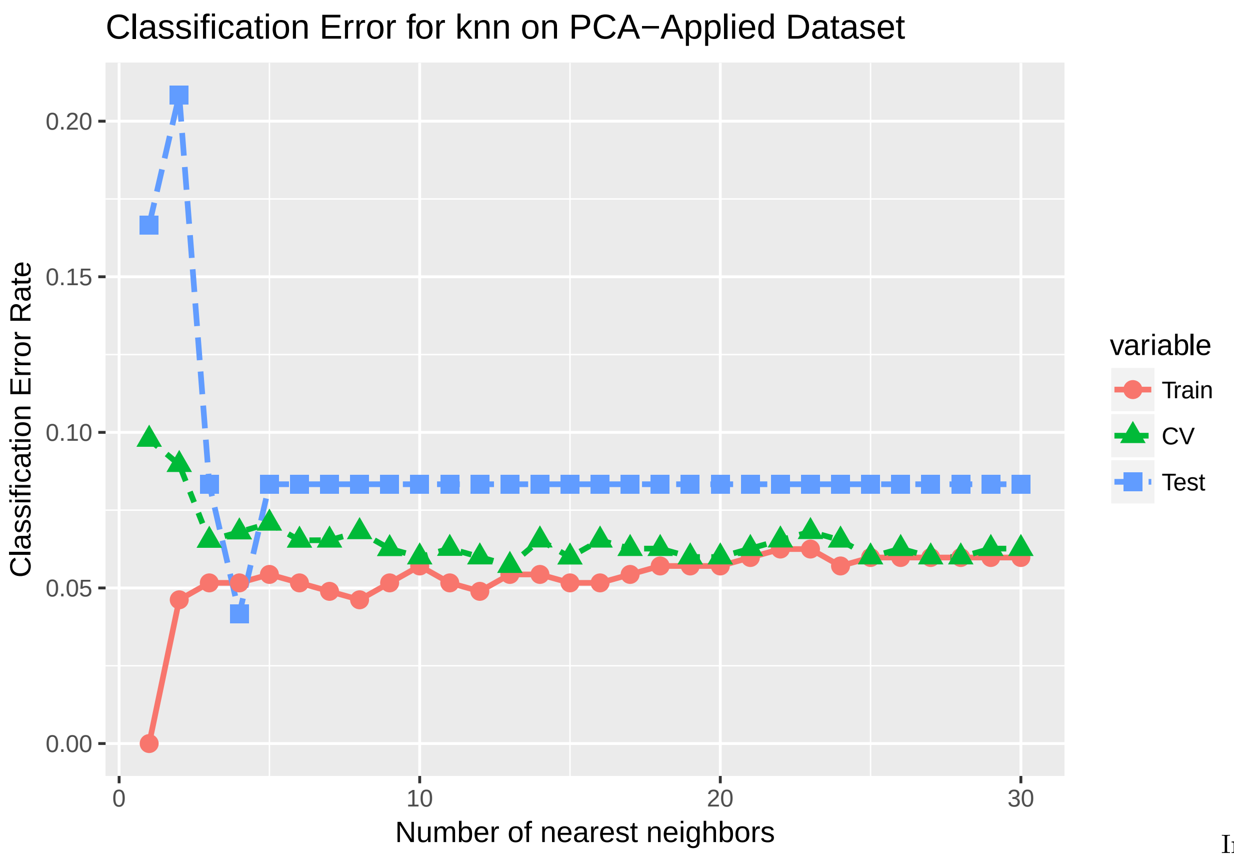
Though Logistic Regression and LDA are supposedly similar in using linear boundary decisions, comparing the error rates shows that the Logistic Regression has overall lower error rates compared to LDA error rates. The pattern is still evident that the test set shows higher error rates compared to the training set error rates, which we probably would have expected from using linear boundary decisions.

– Nearest Neighbor Classifier (knn) –

The final classification method we may introduce in this report is k-nearest neighbors. Unlike the previously mentioned classification techniques, knn is different in that it is a non-parametric classification technique. In other words, there is no established expectation of what the shape of the decision boundaries should be. As a result, there is a very large amount of flexibility in creating the decision boundaries thus causing knn to excel in situations where the decision boundaries are nowhere close to linear. However knn does come with its own set of limitations. For instance knn is known as a local method - relying on the fact that there are not large sudden disturbances in a local neighborhood of the classifications. Knn is also sensitive to skewed distributions and can suffer in dealing with very high dimensional data due to less intuitive understanding of what neighbors are close and which are far. Not only that, but the entire approach of knn is quite different from QDA and LDA; we determine the nearest neighbors of x among the xi and assign x to a class dependent upon the popularity of vote by what class x should be belong in. One final limitation is that knn fails to provide information about what predictors have stronger influence upon the classification compared to others. In the previous parametric methods, we were able to obtain coefficients of the linear discriminants which provide some very important information in the classification decision; this advantage is lost in knn. Otherwise, knn is still a very widely-known classification method that is generally simple to understand and easily implemented.


Classification Error for knn on Original Dataset
Classification Error Rate
0.0
0.1
0.2
0.3
0
10
20
30
Number of Nearest Neighbors
variable
Train
CV
Test


Classification Error for knn on Standardized Dataset
Classification Error Rate
0.00
0.02
0.04
0.06
0.08
0
10
20
30
Number of nearest neighbors
variable
Train
CV
Test



In the implementation of knn onto the dataset variations of original, standardized, and PCA-applied, our choice of the distance metric was simply Euchlidean just for simplicity sake of picking the most convenient distance metric. Further analysis could be drawn upon if we varied the distances, which in turn is interpreted as how we want to preceive the distance between points and neighbors. After doing Cross-Validation on the training dataset upon the different variations of the data, we recieved the above graphs that depict the number of nearest neighbors versus the classification error in each run. Respectively for original, standardized, and PCA-applied the apparent most optimal k to choose at the time of this run was 3, 5, and 8. These numbers can vary greatly especially with the variation in running the cross-validation as well as how the dataset was split into training and test datasets. In general in the plots, the training classification error always tended to be lower than the test classification error with the cross-validation error always bounded between these two error values.

– Performance Comparison / Final Remarks –

Overall, all 3 classification methods were fairly successful to an extent as we never saw error rates above 20%. Even most error rates were capped at about 10%, which is fairly decent considering the size of the dataset as well as the size of the training and test datasets. Because the error rates among the methods are fairly close to each other, I would not make any large convincing argument that any of these classification methods would be largely inappropriate or unsuccessful in achieving the goal of correctly doing classification. We would need to do further analysis upon the structure of the numerical data to more surgically determine to what extent that the assumptions were met or violated for each classification method being applied; this can provide some insight into which method is technically superior to the others. Most advantages and disadvantages of each data processing classification method have been discussed above throughout the report as reference, concluding this report.

Appendix: R Code

```
# Packages for Use
library("dplyr", lib.loc="~/R/x86_64-pc-linux-gnu-library/3.4")
library("GGally", lib.loc="~/R/x86_64-pc-linux-gnu-library/3.4")
library("gridBase", lib.loc="~/R/x86_64-pc-linux-gnu-library/3.4")
library("gridExtra", lib.loc="~/R/x86_64-pc-linux-gnu-library/3.4")
library("MASS", lib.loc="~/R/x86_64-pc-linux-gnu-library/3.4")
library("knitr", lib.loc="~/R/x86_64-pc-linux-gnu-library/3.4")
library("ggplot2", lib.loc="~/R/x86_64-pc-linux-gnu-library/3.4")
library("nnet", lib.loc="~/R/x86_64-pc-linux-gnu-library/3.4")
library("class", lib.loc="~/R/x86_64-pc-linux-gnu-library/3.4")
library("sparsediscrim", lib.loc="~/R/x86_64-pc-linux-gnu-library/3.4")
library("reshape2", lib.loc="~/R/x86_64-pc-linux-gnu-library/3.4")
library("e1071", lib.loc="~/R/x86_64-pc-linux-gnu-library/3.4")

# Import the dataset first
auto = read.table("~/Desktop/School/Stats503/Datasets/auto-mpg.data", stringsAsFactors = FALSE)
colnames(auto) = c("mpg", "cylinders", "displacement", "horsepower", "weight", "acceleration", "modelyea
auto$horsepower <- as.numeric(as.character(auto$horsepower))

# Original Dataset "origset", with classes added
origset = auto
origclass = ifelse(auto$cylinders <= 5, 1,
        ifelse(auto$cylinders == 6, 2,
               3))
origset = cbind(origset, origclass)
origset = origset %>%
  transmute(mpg, displacement, horsepower, weight, acceleration, origclass) %>%
  filter(!is.na(horsepower))

# Standardized Dataset "standset", with classes added
standset = origset
standset[,1:5] = scale(standset[,1:5])

# PCA-processed Dataset "PCAset", with classes added
matform = na.omit(as.matrix(origset[,1:5]))
PCAcorr = princomp(matform, cor = T)$scores[,1:2]
PCAset = cbind(PCAcorr, origset$origclass)

set.seed(123) # Make results reproducible
# Training Datasets
trainsize = floor(0.75 * nrow(mtcars))
# Original Dataset
origtraintargets = sample(seq_len(nrow(origset)), size = trainsize)
origtrain =  origset[-origtraintargets, ]
# Standardized Dataset
standtraintargets = sample(seq_len(nrow(standset)), size = trainsize)
standtrain =  standset[-standtraintargets, ]
# PCA-processed Dataset
PCAtraintargets = sample(seq_len(nrow(PCAset)), size = trainsize)
PCAtrain = as.data.frame(PCAset[-PCAtraintargets,])
# Test Datasets
```

```r
# Original Dataset
origtest = origset[origtraintargets, ]
# Standardized Dataset
standtest = standset[standtraintargets, ]
# PCA-processed Dataset
PCAtest = as.data.frame(PCAset[PCAtraintargets,])

plot1 = ggpairs(data=as.data.frame(origtrain),
        columns=1:5,
        title="Training set of Original Data") # Training set of Original Data

plot2 = ggpairs(data=as.data.frame(origtest),
        columns=1:5,
        title="Test set of Original Data") # Test set of Original Data

summary(origtrain[,1:5])
summary(origtest[,1:5])
plot1
plot2

# Original
origlda = lda(origclass ~ mpg + displacement + weight + horsepower + acceleration, data = origtrain)
origldatrainpred = predict(origlda, origtrain)
origldatestpred = predict(origlda, origtest)
a = 1 - mean(origldatrainpred$class == origtrain$origclass)
b = 1 - mean(origldatestpred$class == origtest$origclass)

# Standardized
standlda = lda(origclass ~ mpg + displacement + weight + horsepower + acceleration, data = standtrain)
standldatrainpred = predict(standlda, standtrain)
standldatestpred = predict(standlda, standtest)
c = 1 - mean(standldatrainpred$class == standtrain$origclass)
d = 1 - mean(standldatestpred$class == standtest$origclass)

# PCA-ed
pcalda = lda(V3 ~ Comp.1 + Comp.2, data = PCAtrain)
pcaldatrainpred = predict(pcalda, PCAtrain)
pcaldatestpred = predict(pcalda, PCAtest)
e = 1 - mean(pcaldatrainpred$class == PCAtrain$V3)
f = 1 - mean(pcaldatestpred$class == PCAtest$V3)

# Combined Table
ldatable = matrix(c(a,b,c,d,e,f), nrow = 3, ncol = 2, byrow = T)
ldarownames = c('Original Dataset', 'Standardized Dataset', 'PCA Applied Dataset')
ldacolnames = c("Training Set", "Test Set")
rownames(ldatable) = ldarownames
colnames(ldatable) = ldacolnames
kable(ldatable, caption = "LDA Classification Error Rates of Cylinder Class Prediction")

boundary_plot <- function(df, classifier, predict_function, resolution = 500) {
  colnames(df) = c("Var1", "Var2", "Class")
  class_train = classifier(x = df[,1:2], y = df[,3])
  v1 = seq(min(df[,1]), max(df[,1]), length=resolution)
  v2 = seq(min(df[,2]), max(df[,2]), length=resolution)
  Grid = expand.grid(Var1 = v1, Var2 = v2)
  Grid$class = predict_function(class_train, Grid)
  ggplot(data=df, aes(x=Var1, y=Var2, color=Class)) +
    geom_contour(data=Grid, aes(z=as.numeric(class)),
                 color="black",size=0.5)+
    geom_point(size=2,aes(color=Class, shape=Class))
}

temporigset = as.data.frame(origset)
temporigset$origclass = as.factor(temporigset$origclass)

lda_wrapper = function(x, y) lda(x = x, grouping = y)
predict_wrapper = function(classifier, data) predict(classifier, data)$class
bp_origlda = boundary_plot(temporigset[,c(1,2,6)], lda_wrapper, predict_wrapper) +
  ggtitle('Original Set: LDA') + theme_dark()
bp_origlda


tempstandset = as.data.frame(standset)
tempstandset$origclass = as.factor(tempstandset$origclass)

bp_standlda = boundary_plot(tempstandset[,c(1,2,6)], lda_wrapper, predict_wrapper) +
  ggtitle('Standardized Set: LDA') + theme_dark()
bp_standlda

temppcaset = as.data.frame(PCAset)
temppcaset$V3 = as.factor(temppcaset$V3)

bp_pcalda = boundary_plot(temppcaset[,c(1,2,3)], lda_wrapper, predict_wrapper) +
  ggtitle('PCA-Applied Set: LDA') + theme_dark()
bp_pcalda

# Original
origqda = qda(origclass ~ mpg + displacement + weight + horsepower + acceleration, data = origtrain)
origqdatrainpred = predict(origqda, origtrain)
origqdatestpred = predict(origqda, origtest)
g = 1 - mean(origqdatrainpred$class == origtrain$origclass)
h = 1 - mean(origqdatestpred$class == origtest$origclass)

# Standardized
standqda = qda(origclass ~ mpg + displacement + weight + horsepower + acceleration, data = standtrain)
standqdatrainpred = predict(standqda, standtrain)
standqdatestpred = predict(standqda, standtest)
i = 1 - mean(standqdatrainpred$class == standtrain$origclass)
j = 1 - mean(standqdatestpred$class == standtest$origclass)

# PCA-ed
pcaqda = qda(V3 ~ Comp.1 + Comp.2, data = PCAtrain)
pcaqdatrainpred = predict(pcaqda, PCAtrain)
pcaqdatestpred = predict(pcaqda, PCAtest)
k = 1 - mean(pcaqdatrainpred$class == PCAtrain$V3)
l = 1 - mean(pcaqdatestpred$class == PCAtest$V3)
```

```
# Combined Table
qdatable = matrix(c(g,h,i,j,k,l), nrow = 3, ncol = 2, byrow = T)
qdarownames = c('Original Dataset', 'Standardized Dataset', 'PCA Applied Dataset')
qdacolnames = c("Training Set", "Test Set")
rownames(qdatable) = qdarownames
colnames(qdatable) = qdacolnames
kable(qdatable, caption = "QDA Classification Error Rates of Cylinder Class Prediction")

# Original
origlog = multinom(origclass ~ mpg + displacement + weight + horsepower + acceleration, data = origtrai
origlogtrainpred = predict(origlog, origtrain, type = 'class')
origlogtestpred = predict(origlog, origtest, type = 'class')
m = 1 - mean(origlogtrainpred == origtrain$origclass)
n = 1 - mean(origlogtestpred == origtest$origclass)


# Standardized
standlog = multinom(origclass ~ mpg + displacement + weight + horsepower + acceleration, data = standtra
standlogtrainpred = predict(standlog, standtrain, type = 'class')
standlogtestpred = predict(standlog, standtest, type = 'class')
o = 1 - mean(standlogtrainpred == standtrain$origclass)
p = 1 - mean(standlogtestpred == standtest$origclass)

# PCA-ed
pcalog = multinom(V3 ~ Comp.1 + Comp.2, data = PCAtrain, trace = FALSE)
pcalogtrainpred = predict(pcalog, PCAtrain, type = 'class')
pcalogtestpred = predict(pcalog, PCAtest, type = 'class')
q = 1 - mean(pcalogtrainpred == PCAtrain$V3)
r = 1 - mean(pcalogtestpred == PCAtest$V3)

# Combined Table
logtable = matrix(c(m,n,o,p,q,r), nrow = 3, ncol = 2, byrow = T)
logrownames = c('Original Dataset', 'Standardized Dataset', 'PCA Applied Dataset')
logcolnames = c("Training Set", "Test Set")
rownames(logtable) = logrownames
colnames(logtable) = logcolnames
kable(logtable, caption = "Logistic Regression Classification Error Rates of Cylinder Class Prediction"

# For original data set
origfolds = cv_partition(origtrain$origclass, num_folds = 5)

train_cv_error = function(K) {
  # Training error
  origknn = knn(train = origtrain[,1:5], test = origtrain[,1:5],
                cl = origtrain$origclass, k = K)
  trainerror = sum(origknn != origtrain$origclass) / nrow(origtrain)
  # Cross Validation error
  auto.cverr = sapply(origfolds, function(fold) {
    sum(origtrain$origclass[fold$test] != knn(train = origtrain[fold$training,1:5], cl = origtrain[fold
  })
  cverror = mean(auto.cverr) # Mean error rate from the Cross validation
  # Testing error
  origknn.test = knn(train = origtrain[,1:5], test = origtest[,1:5],
```

```r
                  cl = origtrain$origclass, k = K)
  testerror = sum(origknn.test != origtest$origclass) / nrow(origtest)
  return(c(trainerror, cverror, testerror))
}

orig_k_errors = sapply(1:30, function(k) train_cv_error(k)) # for 1 to 30 nearest neighbors
origerrs = data.frame(t(orig_k_errors), 1:30)
colnames(origerrs) = c('Train', 'CV', 'Test', 'K')

longform = melt(origerrs, id="K")
ggplot(longform, aes_string(x="K", y="value", colour="variable",
group="variable", linetype="variable", shape="variable")) +
geom_line(size=1) + labs(x = "Number of Nearest Neighbors",
y = "Classification Error Rate", title = "Classification Error for knn on Original Dataset") +
geom_point(size=3)

# Determining the optimal k to choose, it turns out it is k = 3 after a few runs
knntuning = tune.knn(x= origtrain[,1:5], y = as.factor(origtrain$origclass), k = 1:30)
# summary(knntuning) uncomment this to obtain the optimal k, suppressed for markdown output

# For standardized dataset
standfolds = cv_partition(standtrain$origclass, num_folds = 5)

train_cv_error = function(K) {
  #Training error
  standknn = knn(train = standtrain[,1:5], test = standtrain[,1:5],
                  cl = standtrain$origclass, k = K)
  trainerror = sum(standknn != standtrain$origclass) / nrow(standtrain)
  # Cross Validation error
  auto.cverr = sapply(standfolds, function(fold) {
    sum(standtrain$origclass[fold$test] != knn(train = standtrain[fold$training,1:5], cl = standtrain[f
  })
  cverror = mean(auto.cverr)
  # Testing error
  standknn.test = knn(train = standtrain[,1:5], test = standtest[,1:5],
                  cl = standtrain$origclass, k = K)
  testerror = sum(standknn.test != standtest$origclass) / nrow(standtest)
  return(c(trainerror, cverror, testerror))
}

stand_k_errors = sapply(1:30, function(k) train_cv_error(k))
standerrs = data.frame(t(stand_k_errors), 1:30)
colnames(standerrs) = c('Train', 'CV', 'Test', 'K')

longform = melt(standerrs, id="K")
ggplot(longform, aes_string(x="K", y="value", colour="variable",
group="variable", linetype="variable", shape="variable")) +
geom_line(size=1) + labs(x = "Number of nearest neighbors",
y = "Classification Error Rate", title = "Classification Error for knn on Standardized Dataset") +
geom_point(size=3)

# Determining the optimal k to choose, it turns out it is k = 5 after a few runs
knntuning = tune.knn(x= standtrain[,1:5], y = as.factor(standtrain$origclass), k = 1:30)
```

```r
# summary(knntuning) uncomment this to obtain the optimal k, suppressed for markdown output

# For pca-applied dataset
PCAfolds = cv_partition(PCAtrain$V3, num_folds = 5)

train_cv_error = function(K) {
  #Training error
  PCAknn = knn(train = PCAtrain[,1:2], test = PCAtrain[,1:2],
                cl = PCAtrain$V3, k = K)
  trainerror = sum(PCAknn != PCAtrain$V3) / nrow(PCAtrain)
  #CV error
  auto.cverr = sapply(PCAfolds, function(fold) {
    sum(PCAtrain$V3[fold$test] != knn(train = PCAtrain[fold$training,1:2], cl = PCAtrain[fold$training,
  })
  cverror = mean(auto.cverr)
  #Testing error
  PCAknn.test = knn(train = PCAtrain[,1:2], test = PCAtest[,1:2],
                cl = PCAtrain$V3, k = K)
  testerror = sum(PCAknn.test != PCAtest$V3) / nrow(PCAtest)
  return(c(trainerror, cverror, testerror))
}

PCA_k_errors = sapply(1:30, function(k) train_cv_error(k))
PCAerrs = data.frame(t(PCA_k_errors), 1:30)
colnames(PCAerrs) = c('Train', 'CV', 'Test', 'K')

longform = melt(PCAerrs, id="K")
ggplot(longform, aes_string(x="K", y="value", colour="variable",
group="variable", linetype="variable", shape="variable")) +
geom_line(size=1) + labs(x = "Number of nearest neighbors",
y = "Classification Error Rate", title = "Classification Error for knn on PCA-Applied Dataset") +
geom_point(size=3)

# Determining the optimal k to choose, it turns out it is k = 8 after a few runs
knntuning = tune.knn(x= PCAtrain[,1:2], y = as.factor(PCAtrain$V3), k = 1:30)
# summary(knntuning) uncomment this for the optimal k, suppressed for markdown output
```