~~~
---
title: "ST 537 Final Project"
author: "Shantel Ward, Rong Huang, John Clements"
date: "11/17/2020"
output:
  pdf_document: default
  word_document: default
---

```{r setup, include=FALSE}
knitr::opts_chunk$set(echo = TRUE)
```
# Methods: Data Analysis

```{r}
suppressWarnings(library(tidyverse))
##### DATA CLEANING #####
# Read in data set.
# Read in the Breast Cancer data set.
#setwd("C:/Users/17739/Documents/GitHub/BreastCancerDetection/Breast Cancer
Git Hub/")
bc.df <- read.table("Data/breast-cancer-wisconsin.data", sep=",")
# Add informative column names.
colnames(bc.df) = c("Id", "Clump Thickness", "Cell Size Uniformity",
                    "Cell Shape Uniformity", "Marginal Adhesion",
                    "Single Epithelial Cell Size", "Bare Nuclei",
                    "Bland Chromatin", "Normal Nucleoli", "Mitosis",
                    "Class")
# Calculate median Bare Nuclei by Class.
median.BareNuclei.2 <- bc.df %>%
  group_by(Class) %>%
  summarize(Median = median(as.numeric(`Bare Nuclei`),
                        na.rm=TRUE))%>%filter(Class == 2)
median.BareNuclei.4 <- bc.df %>%
  group_by(Class) %>%
  summarize(Median = median(as.numeric(`Bare Nuclei`),
                        na.rm=TRUE))%>%filter(Class == 4)
# Impute missing Bare Nuclei data with the median for each Class mean.
bc.df <- bc.df %>%
  mutate(`Bare Nuclei Revised` = ifelse(`Bare Nuclei` == "?" & Class == 2,
                                        round(median.BareNuclei.2$Median,2),
                                        ifelse(`Bare Nuclei` == "?" & Class ==
4,
                                               round(median.BareNuclei.
4$Median,2), `Bare Nuclei`)))
# Convert Bare Nuclei Revised data to the int data type.
bc.df$`Bare Nuclei Revised` <- as.integer(bc.df$`Bare Nuclei Revised`)
# Convert Class to a factor variable.
bc.df$Class = as.factor(bc.df$Class)
##### SUMMARY STATISTICS #####
bc.df.mean <- dplyr::select(bc.df, -c("Id","Bare Nuclei")) %>% group_by(Class)
%>% summarise_all("mean")
#transpose from wide to long
bc.df.mean<-as.data.frame(t(dplyr::select(bc.df.mean,-"Class")))
~~~

~~~
colnames(bc.df.mean) <- c("Benign", "Malignant")
bc.df.sd <- dplyr::select(bc.df, -c("Id","Bare Nuclei")) %>% group_by(Class)
%>% summarise_all("sd")
#transpose from wide to long
bc.df.sd<-as.data.frame(t(dplyr::select(bc.df.sd,-"Class")))
colnames(bc.df.sd) <- c("Benign", "Malignant")
##### DATA SPLITTING #####
# Set a random seed for reproducibility.
set.seed(17)
# Count the number of observations in the data set.
num.obs <- dim(bc.df)[1]
# Set the proportion of observations to be in the training data set.
prop.train <- 0.7
# Set the number of observations to be in the training data set.
num.train <- round(num.obs*prop.train)
# set the indicies of the dataset to be in the training data set.
train.indices <- sample(1:num.obs, num.train, replace=FALSE)
# Subset the data into training and testing data sets.
train.bc.df <- bc.df[train.indices,]
test.bc.df <- bc.df[-train.indices,]
# Save the vector of the indicies of the independent variable columns.
X_col_indicies <- c(2, 3, 4, 5, 6, 8, 9, 10, 12)
```

## Summary Statistics: Mean
```{r}
knitr::kable(bc.df.mean, align = "c",
             caption =  "Mean Summary Statistics", digits = 2)
```

## Summary Statistics: Standard Deviation
```{r}
knitr::kable(bc.df.sd, align = "c",
             caption =  "Standard Deviation Summary Statistics", digits = 3)
```

## Methods: Statistical Models

# Methods: Results

## Hotelling’s T2 Two Sample Test Results

```{r}
# pair plot
pairs(bc.df[,-c(1,7,11)],
      col = c(4, 2)[as.numeric(bc.df$Class)],   # Change color by group
      pch = c(8, 1)[as.numeric(bc.df$Class)],   # Change points by group
      main = "Pair Plot of Benign and Malignant for All Variables",
oma=c(2,2,2,16))
par(xpd = TRUE)
legend("bottomright", col=c(4, 2),pch = c(8, 1),
       legend = c("Benign", "Malignant"))
```
~~~

~~~
```{r}
# boxplot
par(mfrow=c(3,3))
for(col_index in X_col_indicies){
  boxplot(bc.df[,col_index] ~ Class, data = bc.df, col=c(4,2),
          xlab="", ylab = "", main=names(bc.df)[col_index])
}
```

```{r}
suppressWarnings(library(ICSNP))
# Seperate the 2 classes into dataframes for mean vector comparison.
X = as.matrix(bc.df[bc.df$Class == "2", X_col_indicies]) # class 2
Y = as.matrix(bc.df[bc.df$Class == "4", X_col_indicies]) # class 4
#normality check
################## chisquare.plot ##############
chisquare.plot <- function(x, mark, title) {
  # x = A nxp data matrix, mark: number of extreme points to mark,
  # p = number of variables, n = sample size
  p <- ncol(x)
  n <- nrow(x)
  # xbar and s
  xbar <- colMeans(x)
  s <- cov(x)
  # Mahalanobis dist, sorted
  x.cen <- scale(x, center = T, scale = F)
  d2 <- diag(x.cen %*% solve(s) %*% t(x.cen))
  sortd <- sort(d2)
  # chi-sq quantiles
  qchi <- qchisq((1:n - 0.5)/n, df = p)
  # plot, mark points with heighest distance
  plot(qchi, sortd, pch = 19, xlab = "Chi-square quantiles",
       ylab = "Mahalanobis squared distances",
       main = title)
  points(qchi[(n - mark + 1):n], sortd[(n - mark + 1):n], cex = 3, col =
"#990000")
}
############################################
par(mfrow=c(1,2))
chisquare.plot(X,2, "Chi-square Q-Q Plot of Benign")
chisquare.plot(Y,2, "Chi-square Q-Q Plot of Malignant")
## Two-sample Hotelling's T2 test
HotellingsT2(X,Y)
## Bonferroni intervals
alpha = 0.05 # old significance level
p = 2 # number of intervals/variables
# mean vectors
xbar = colMeans(X)
ybar = colMeans(Y)
difference = xbar - ybar
# covariances of each group
S.x = cov(X)
S.y = cov(Y)
# bc.df statistics summary
~~~

~~~
stats = round(cbind(xbar, ybar, sqrt(diag(S.x)), sqrt(diag(S.y))),3)
colnames(stats) = c("Benign Mean", "Malignant Mean", "Benign Sd", "Malignant
Sd")
stats
# sample sizes
m = nrow(X)
n = nrow(Y)
# pooled covariance matrix
S.pool = ((m-1)*S.x + (n-1)*S.y) / (m + n - 2)
# critical value
crit = qt(alpha/(2*p), df = m+n-2, lower.tail = F)
```

```{r}
# Bonferroni intervals
half.width = crit*sqrt(diag(S.pool)*(1/m + 1/n))
lower = difference - half.width
upper = difference + half.width
int.bonf = round(cbind(lower, upper),3)
knitr::kable(int.bonf, align = "c", caption =  "Bonferroni Intervals", digits
= 2)
```

## Principal Components Analysis (PCA) Results
```{r}
##### PCA (Principle Components Analysis #####
# Scientific Question 2
# Can we use PCA to reduce the dimensionality of the data and
# to identify/summarize crucial variables?
suppressWarnings(library(knitr))
suppressWarnings(library(GGally))
# Split the training and testing data sets into independent
# variables and dependent variables.
train.bc.df.class <- train.bc.df$Class
train.bc.df.X <- select(train.bc.df, -c("Id","Bare Nuclei","Class"))
test.bc.df.class <- test.bc.df$Class
test.bc.df.X <- select(test.bc.df, -c("Id","Bare Nuclei","Class"))
# Plot a correlation matrix.
ggcorr(train.bc.df.X, label=TRUE, label_size=4, label_round=2, layout.exp = 5,
hjust = "right")
# Shows that Cell Shape Uniformity and Cell Size Uniformity are highly
correlated (0.90).
#cor(train.bc.df.X)
```

```{r}
# Standardize the variables.
train.bc.df.X.std <- scale(train.bc.df.X, center=TRUE, scale=TRUE)
# Perform PCA.
data.pca <- prcomp(train.bc.df.X.std)
# Extract the importance of each component.
# standard deviation
st.dev <- data.pca$sdev
# variance
~~~

~~~~
var <- st.dev^2
# total variance
TV <- sum(var)
#proportion of variance explained
prop <- var/TV
#cumulative proportion of variance explained
pve <- cumsum(var)/TV
#combine in a table
tab <- rbind(st.dev, prop, pve)
rownames(tab) <- c("Standard deviation", "Proportion of variance",
                   "Cumulative proportion")
colnames(tab) <- paste0("PC",1:9)
knitr::kable(tab, align = "c",
             caption = "PCA Importance of Components", digits = 3)
```

```{r}
# Make a Scree Plot to help determine how many PCs to retain.
screeplot(data.pca, type = "line", main = "Variance Explained by each PC")
```

```{r}
# The elbow in the graph is at PC2.
# PC 1 and 2 explain 74% of the variability
# Display the loadings for PC1 and PC2 for review.
#round(data.pca$rotation[, 1:2], 3)
knitr::kable(data.pca$rotation[, 1:2], align = "c",
             caption = "Loadings of the First Two PCs", digits = 3)
```

```{r}
# PC1 is roughly equally weighted.
# PC2 is essentially the effect of Mitosis.
PC1 <- data.pca$rotation[, 1]
PC2 <- data.pca$rotation[, 2]
PC.scores.1 <- train.bc.df.X.std %*% PC1
PC.scores.2 <- train.bc.df.X.std %*% PC2
# Plot PC score 1 vs PC score 2 colored by class.
plot(PC.scores.1,PC.scores.2, pch=19, col=train.bc.df.class, main = "PC1
Scores vs PC2 Scores", xlab = "PC1 Scores", ylab = "PC2 Scores")
legend("bottomleft", legend=c("Class 2: Benign", "Class 4: Malignant"),
       col=c("black", "red"), pch=19, cex=0.8)
```

## Classification
```{r}
# Create a dataframe only holding the X variables and the Class, starting with
the X variables.
train.df = train.bc.df.X
test.df = test.bc.df.X
# Re-attach the Class to the independent variables of the training and testing
dataframes.
train.df$Class <- train.bc.df.class
test.df$Class <- test.bc.df.class
~~~~

~~~~
# This is essential to get rpart to run on this data.
colnames(train.df) <- make.names(colnames(train.df))
colnames(test.df) <- make.names(colnames(train.df))
# Set the number of principal components to keep.
num.keep <- 2
# Re-fit prcomp() so it knows to center and scale input data.
pca.fit <- prcomp(train.bc.df.X, center=TRUE, scale=TRUE)
summary(pca.fit)
# Create a new dataframe to hold the prcomp() transformed training data.
train.pca <- as.data.frame(pca.fit$x[, 1:num.keep])
# Create a new dataframe to hold the prcomp() transformed testing data.
test.pca <- as.data.frame(predict(pca.fit, test.bc.df.X)[, 1:num.keep])
# Re-attach the Class to the PCA-trainsformed training and testing dataframes.
train.pca$Class <- train.bc.df.class
test.pca$Class <- test.bc.df.class
```

```{r}
# Calculate the no information rates.
no.info.rate.train <- sum(train.pca$Class == 2) / length(train.pca$Class)
no.info.rate.test <- sum(test.pca$Class == 2) / length(test.pca$Class)
print(paste("No Information Rate (Training Set): ",
            round(100*no.info.rate.train, 3), "%", sep=""))
print(paste("No Information Rate (Testing Set): ",
            round(100*no.info.rate.test, 3), "%", sep=""))
```


```{r}
suppressWarnings(library(caret))
suppressWarnings(library(kernlab))
# We will be using 5-fold cross-validation, repeating the process 20 times.
TrControl <- trainControl(method = "repeatedcv",
                          number = 5,
                          repeats = 20)
# Fit a KNN classifier by selecting the model with the best number of
neighbors
# by performing 5-fold cross-validation tuning the k parameter using the odd
numbers
# from 1 up to 9.
knn.model <- train(Class ~ ., data = train.df,
                   method = "knn",
                   trControl = TrControl,
                   tuneGrid = expand.grid(k = seq(1, 9, by=2)))
# Perform 5-fold cross-validation using a LDA classifier.
lda.model <- train(Class ~ ., data = , train.df,
                   method = "lda",
                   trControl = TrControl)
# Perform 5-fold cross-validation using a QDA classifier.
qda.model <- train(Class ~ ., data = , train.df,
                   method = "qda",
                   trControl = TrControl)
# Fit a tree classifier by selecting the model with the best complexity
parameter (cp)
~~~~

```
# by performing 5-fold cross-validation tuning the cp parameter using 0.001,
0.01,
# and 0.1
tree.model <- train(Class ~ ., data = train.df,
                    method = "rpart",
                    trControl = TrControl,
                    tuneGrid = expand.grid(cp = seq(0.05, 0.5, by=0.05)))
# Fit a SVM classifier with a radial kernel by selecting the model tuning the
C and
# sigma parameters by performing 5-fold cross-validation tuning
# the C parameter from 0.25 to 4, doubling each time, and the sigma parameter
from
# 0.125 to 8, doubling each time
svm.model <- train(Class ~ ., data = train.df,
                   method = "svmRadial",
                   trControl = TrControl,
                   tuneGrid = expand.grid(C = 2^c(-2:2),
                                          sigma = 2^(-3:3)))
# Fit a KNN classifier by selecting the model with the best number of
neighbors
# by performing 5-fold cross-validation tuning the k parameter using the odd
numbers
# from 1 up to 9.
knn.model.pca <- train(Class ~ ., data = train.pca,
                       method = "knn",
                       trControl = TrControl,
                       tuneGrid = expand.grid(k = seq(1, 9, by=2)))
# Perform 5-fold cross-validation using a LDA classifier.
lda.model.pca <- train(Class ~ ., data = , train.pca,
                       method = "lda",
                       trControl = TrControl)
# Perform 5-fold cross-validation using a QDA classifier.
qda.model.pca <- train(Class ~ ., data = , train.pca,
                       method = "qda",
                       trControl = TrControl)
# Fit a tree classifier by selecting the model with the best complexity
parameter (cp)
# by performing 5-fold cross-validation tuning the cp parameter using 0.001,
0.01,
# and 0.1
tree.model.pca <- train(Class ~ ., data = train.pca,
                        method = "rpart",
                        trControl = TrControl,
                        tuneGrid = expand.grid(cp = seq(0.05, 0.5, by=0.05)))
# Fit a SVM classifier with a radial kernel by selecting the model tuning the
C and
# sigma parameters by performing 5-fold cross-validation tuning
# the C parameter from 0.25 to 4, doubling each time, and the sigma parameter
from
# 0.125 to 8, doubling each time
svm.model.pca <- train(Class ~ ., data = train.pca,
                       method = "svmRadial",
                       trControl = TrControl,
                       tuneGrid = expand.grid(C = 2^c(-2:2),
```

~~~~
                                              sigma = 2^(-3:3)))
# Summarize the Accuracy and Cohen's Kappa for each model in the 5-fold CV.
resamp <- resamples(list(KNN = knn.model, LDA = lda.model, QDA = qda.model,
                         TREE = tree.model, SVM = svm.model,
                         KNN.PCA = knn.model.pca, LDA.PCA = lda.model.pca,
                         QDA.PCA = qda.model.pca, TREE.PCA = tree.model.pca,
                         SVM.PCA = svm.model.pca))
```

```{r}
quart.1st <- function(x){
  # This helper function finds the first quartile.
  return(quantile(x, probs=0.25))
}
quart.3rd <- function(x){
  # This helper function finds the third quartile.
  return(quantile(x, probs=0.75))
}
resample.summary <- function(resample, ordered.model.names, accuracy=TRUE){
  # This function takes in a resample object, a vector of the models tested
  # in order, and a boolean for accuracy. If accuracy is true, it returns
  # a dataframe summary of the resampling accuracy for each model. Else,
  # it returns a dataframe summary of the resampling Cohen's kappa for
  # each model.

  if(accuracy == TRUE){
    # Summarize the accuracies.
    X = resample$values[,seq(2, dim(resample$values)[2], 2)]
  }
  else{
    # Summarize the Cohen's Kappas.
    X = resample$values[,seq(3, dim(resample$values)[2], 2)]
  }
  my.min <- apply(X,MARGIN=2, FUN=min)
  my.1st <- apply(X, 2, quart.1st)
  my.median <- apply(X, MARGIN=2, FUN=median)
  my.mean <- apply(X, MARGIN=2, FUN=mean)
  my.3rd <- apply(X, 2, quart.3rd)
  my.max <- apply(X, MARGIN=2, FUN=max)

  my.summary <- matrix(c(my.min,
                         my.1st,
                         my.median,
                         my.mean,
                         my.3rd,
                         my.max),
                       nrow=(dim(resample$values)[2]-1)/2)
  my.summary <- as.data.frame(my.summary)
  colnames(my.summary) <- c("Min.", "1st Quartile", "Median",
                         "Mean", "3rd Quartile", "Max.")
  rownames(my.summary) <- ordered.model.names
  return(my.summary)
}
```
~~~~

~~~
```{r}
# Get summaries of the resampling perfomances for the models trained on the
raw
# data.
ordered.model.names <- c("KNN", "LDA", "QDA", "TREE", "SVM",
                         "KNN.PCA", "LDA.PCA", "QDA.PCA", "TREE.PCA",
"SVM.PCA")
acc <- resample.summary(resamp, ordered.model.names, accuracy=TRUE)
kappa <- resample.summary(resamp, ordered.model.names, accuracy=FALSE)
~~~

```{r}
knitr::kable(acc, align = "c",
             caption =  "Resampling Accuracy", digits = 7)
knitr::kable(kappa, align = "c",
             caption =  "Resampling Cohen's Kappa", digits = 7)
```

```{r}
# Make a boxplot of the model Accuracies and Cohen's Kappas.
bwplot(resamp)
```

```{r}
suppressWarnings(library(klaR))
# Get the best hyperparameter for the tree model.
best.cp = tree.model.pca$bestTune$cp
# Get predictions for the test set from the tree model.
tree.preds <- predict(tree.model.pca, test.pca)
# Make a confusion matrix for the tree model.
tree.emat <- errormatrix(test.pca$Class, tree.preds,
                         relative = TRUE)
# Convert the confusion matrix for the tree model to a dataframe.
tree.emat <- as.data.frame(round(tree.emat, 3))
# Add informative row and column names.
colnames(tree.emat) <- c('Predicted Benign', "Predicted Malignant", "Sum")
rownames(tree.emat) <- c('True Benign', "True Malignant", "Sum")
```

```{r}
# Get the best hyperparameter for the knn model.
best.k = knn.model.pca$bestTune$k
# Get predictions for the test set from the knn model.
knn.preds <- predict(knn.model.pca, test.pca)
# Make a confusion matrix for the knn model.
knn.emat <- errormatrix(test.pca$Class, knn.preds,
                        relative = TRUE)
# Convert the confusion matrix for the tree model to a dataframe.
knn.emat <- as.data.frame(round(knn.emat, 3))
# Add informative row and column names.
colnames(knn.emat) <- c('Predicted Benign', "Predicted Malignant", "Sum")
rownames(knn.emat) <- c('True Benign', "True Malignant", "Sum")
```
```
```

~~~
```{r}
# Set the proportion of tree predictions equal to knn predictions.
proportion.equal <- sum(tree.preds == knn.preds) / length(tree.preds)
print(paste("Percent of Tree Model Predictions equal to K-NN Model
Predictions: ",
            round(100*proportion.equal, 5), "%", sep=""))
```

```{r}
knitr::kable(knn.emat, align = "c",
             caption =  paste("Tree (cp = ", best.cp, ") and ", "KNN (k = ",
best.k, ")", sep=""))
```

```{r}
# Calculate the accuracy for the tree model.
tree.accuracy <- sum(tree.preds == test.pca$Class) / length(tree.preds)
# Calculate the APER for the tree model.
tree.aper <- sum(tree.preds != test.pca$Class) / length(tree.preds)
# Calculate the sensistivity for the tree model.
tree.true.pos <- test.pca[test.pca$Class == 4, "Class"]
tree.true.pos.preds <- tree.preds[test.pca$Class == 4]
tree.sensitivity <- sum(tree.true.pos == tree.true.pos.preds) /
length(tree.true.pos)
# Calculate the specificity for the tree model.
tree.true.neg <- test.pca[test.pca$Class == 2, "Class"]
tree.true.neg.preds <- tree.preds[test.pca$Class == 2]
tree.specificity <- sum(tree.true.neg == tree.true.neg.preds) /
length(tree.true.neg)
```

```{r}
# Calculate the accuracy for the knn model.
knn.accuracy <- sum(knn.preds == test.pca$Class) / length(knn.preds)
# Calculate the APER for the knn model.
knn.aper <- sum(knn.preds != test.pca$Class) / length(knn.preds)
# Calculate the sensistivity for the knn model.
knn.true.pos <- test.pca[test.pca$Class == 4, "Class"]
knn.true.pos.preds <- knn.preds[test.pca$Class == 4]
knn.sensitivity <- sum(knn.true.pos == knn.true.pos.preds) /
length(knn.true.pos)
# Calculate the specificity for the knn model.
knn.true.neg <- test.pca[test.pca$Class == 2, "Class"]
knn.true.neg.preds <- knn.preds[test.pca$Class == 2]
knn.specificity <- sum(knn.true.neg == knn.true.neg.preds) /
length(knn.true.neg)
```

```{r}
test.performance <- matrix(c(tree.accuracy, tree.aper, tree.sensitivity,
tree.specificity),
                          nrow=1, byrow=TRUE)
~~~

~~~~
rownames(test.performance) <- c(paste("Tree (cp = ", best.cp, ") and ", "KNN
(k = ", best.k, ")", sep=""))
colnames(test.performance) <- c("Accuracy", "APER", "Sensitivity",
"Specificity")
test.performance <- as.data.frame(test.performance)
knitr::kable(test.performance, align = "c",
             caption =  "Test Set Peformance", digits = 5)
```

# Conclusions
```{r}
par(mfrow=c(3,3))
for(col_index in X_col_indicies){
 hist(bc.df[,col_index],
      main = paste("Histogram of", colnames(bc.df[col_index])),
      xlab = "Values")
}
```

```{r}
par(mfrow=c(3,3))
for(col_index in X_col_indicies){
 hist(bc.df[bc.df$Class == 2,col_index],
      main = paste("Histogram of", colnames(bc.df[col_index])),
      xlab = "Values")
}
```

```{r}
par(mfrow=c(3,3))
for(col_index in X_col_indicies){
 hist(bc.df[bc.df$Class == 4,col_index],
      main = paste("Histogram of", colnames(bc.df[col_index])),
      xlab = "Values")
}
```
~~~~