Assignment 2

Group 28: Mona Behrouzian, Traye Lin, Maya Ansu

October 22, 2023

Q1. DIMENSIONALITY REDUCTION

Q1.1. Perform PCA on the features of the dataset. How much of the variation in the data is associated with PC1?

ANSWER: 42.8% of the variation in the data is associated with PC1

```
# PCA code from tutorial 1. columns 3 to 32 only are features scales
# and centers the data, required for PCA Outputs the features as
# rows, and principle components as columns, or the axes
ovarian.pca <- prcomp(ovarian.dataset[, c(3:32)], center = TRUE, scale. = TRUE)

# Gives the Important of components \tShows us the standard deviation
# and proportion of variance
summary(ovarian.pca)
```

```
## Importance of components:
##                           PC1    PC2     PC3     PC4     PC5    PC6    PC7
## Standard deviation     3.5820 2.2873 1.62395 1.37410 1.24910 1.0844 0.8306
## Proportion of Variance 0.4277 0.1744 0.08791 0.06294 0.05201 0.0392 0.0230
## Cumulative Proportion  0.4277 0.6021 0.68997 0.75291 0.80492 0.8441 0.8671
##                           PC8     PC9    PC10    PC11   PC12   PC13   PC14
## Standard deviation     0.74686 0.67762 0.61684 0.60200 0.5771 0.5139 0.5021
## Proportion of Variance 0.01859 0.01531 0.01268 0.01208 0.0111 0.0088 0.0084
## Cumulative Proportion  0.88571 0.90101 0.91369 0.92578 0.9369 0.9457 0.9541
##                           PC15   PC16   PC17    PC18    PC19    PC20   PC21
## Standard deviation     0.45896 0.3989 0.3834 0.36254 0.32797 0.30949 0.3001
## Proportion of Variance 0.00702 0.0053 0.0049 0.00438 0.00359 0.00319 0.0030
## Cumulative Proportion  0.96110 0.9664 0.9713 0.97569 0.97928 0.98247 0.9855
##                           PC22    PC23    PC24   PC25    PC26    PC27    PC28
## Standard deviation     0.27191 0.26081 0.24722 0.2326 0.22154 0.20068 0.18042
## Proportion of Variance 0.00246 0.00227 0.00204 0.0018 0.00164 0.00134 0.00108
## Cumulative Proportion  0.98794 0.99020 0.99224 0.9940 0.99568 0.99702 0.99811
##                           PC29    PC30
## Standard deviation     0.17164 0.16532
## Proportion of Variance 0.00098 0.00091
## Cumulative Proportion  0.99909 1.00000
```

Q1.2. You want to represent 90% of the variance in the data by dimensionality reduction. How many PCs do you need to achieve this? In other words, what would be the dimensionality of the reduced feature space so that you preserve 90% of the variability in the data?

ANSWER: Need 9 PCs for 90% of the variance

```
# we are looking for a cumulative proportion of variance of 90%

# extract the standard deviation of the PCA
std.pca <- ovarian.pca$sdev

# variance is square of std
var.pca <- std.pca^2

# proportion of var is the variance divided by sum of all variances
prop.pca <- var.pca/sum(var.pca)

# cumulative proportion is the cumulative sum of proportion of
# variance
c.sum.pca <- cumsum(prop.pca)

# use 'which' function to find 1st instance when the cumsum is >=90%
which(c.sum.pca >= 0.9)[1]
```

```
## [1] 9
```

Q1.3. In a 2-D plot, can you plot the observations corresponding to the first two important PCs? Note, use two different colors to represent the two classes of cells.

ANSWER: See plot below.

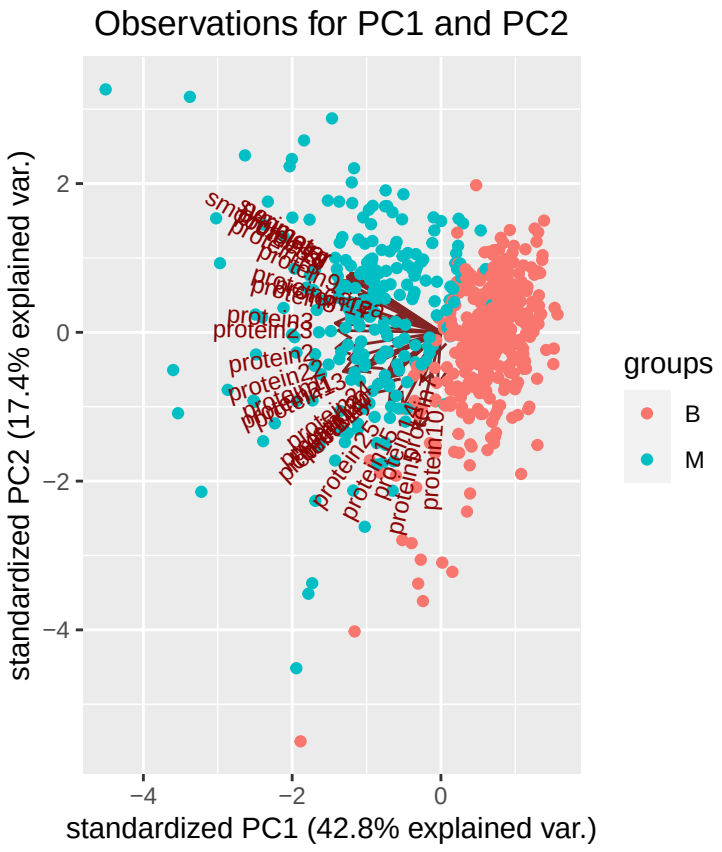
```
# install.packages('devtools') #in console
library(devtools)
```

```
## Loading required package: usethis
```

```
# install.packages('remotes') #in console
# remotes::install_github('vqv/ggbiplot') #in console
library(ggbiplot)
```

```
## Loading required package: ggplot2

## Loading required package: plyr

## Loading required package: scales

## Loading required package: grid
```

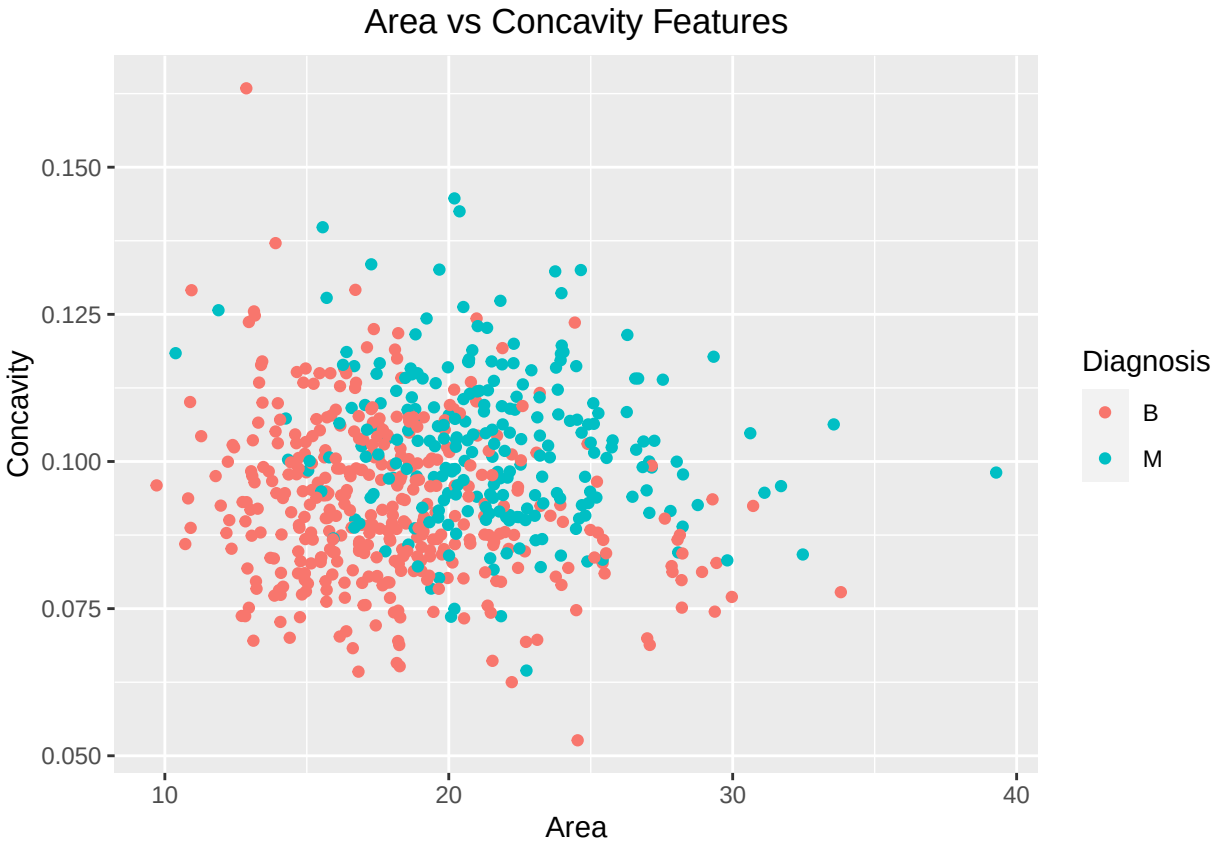
```
library(ggplot2)
library(plyr)
library(scales)
library(grid)

ggbiplot(ovarian.pca, choices = c(1, 2), groups = ovarian.dataset$diagnosis) +
    ggtitle("Observations for PC1 and PC2") + theme(plot.title = element_text(hjust = 0.5))
```



Q1.4. Can you plot the "area" and "concavity" features associated with the cells?

ANSWER: See plot below.

```
library(ggplot2)
Diagnosis <- ovarian.dataset$diagnosis
Area <- ovarian.dataset$area
Concavity <- ovarian.dataset$concavity

ggplot(ovarian.dataset) + geom_point(aes(x = Area, y = Concavity, color = Diagnosis)) +
    ggtitle("Area vs Concavity Features") + theme(plot.title = element_text(hjust = 0.5))
```

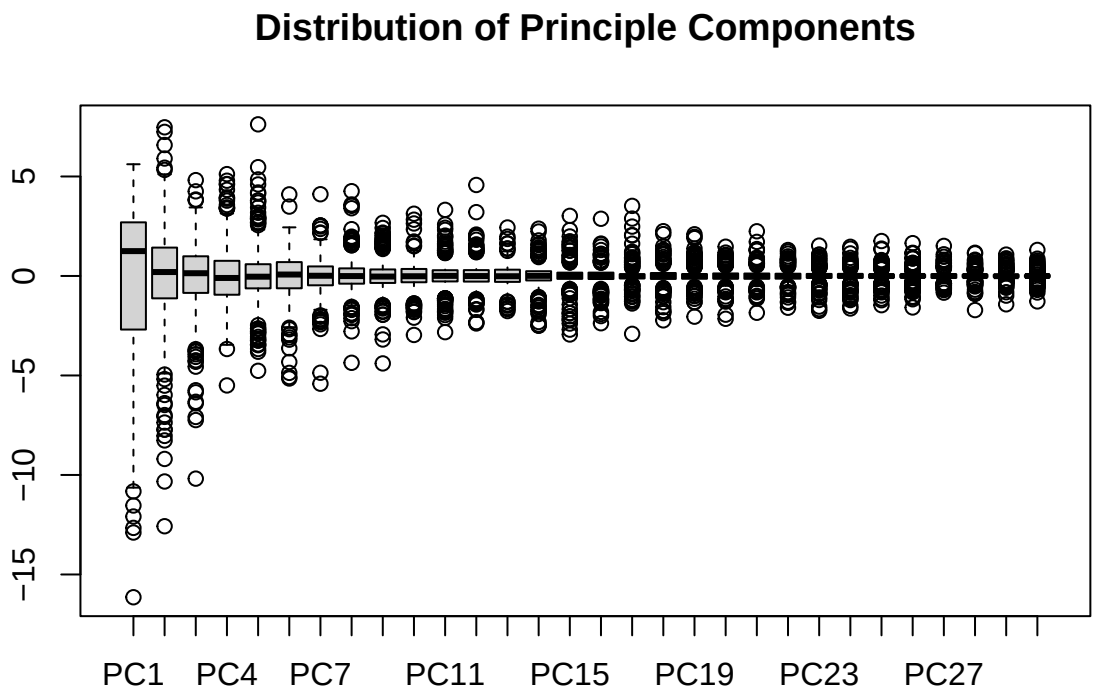


Q1.5. What is the difference between the two plots? Which one gives you better separation between the classes and why?

The clustered plot gives better separation compared to the second plot. This makes sense because the clustered plot used PCA to determine which two features define the most variance in the data set, whereas just plotting area vs concavity is simply selecting two features to plot at random. PCA determines new axes that gives the most variance in our data set so we are better able to cluster our observations.

BONUS: Q1.6 Plot the distribution of the PCs

```
boxplot(ovarian.pca$x)
title("Distribution of Principle Components")
```



Q2. CLUSTERING

Note: When comparing model predictions to true labels, obtain a confusion matrix and include this result in your submission. You can obtain this by using the table() function like so: table(predictions, labels)

Q2.1 Part 1: Apply kmeans clustering on the data and identify two clusters within your dataset.

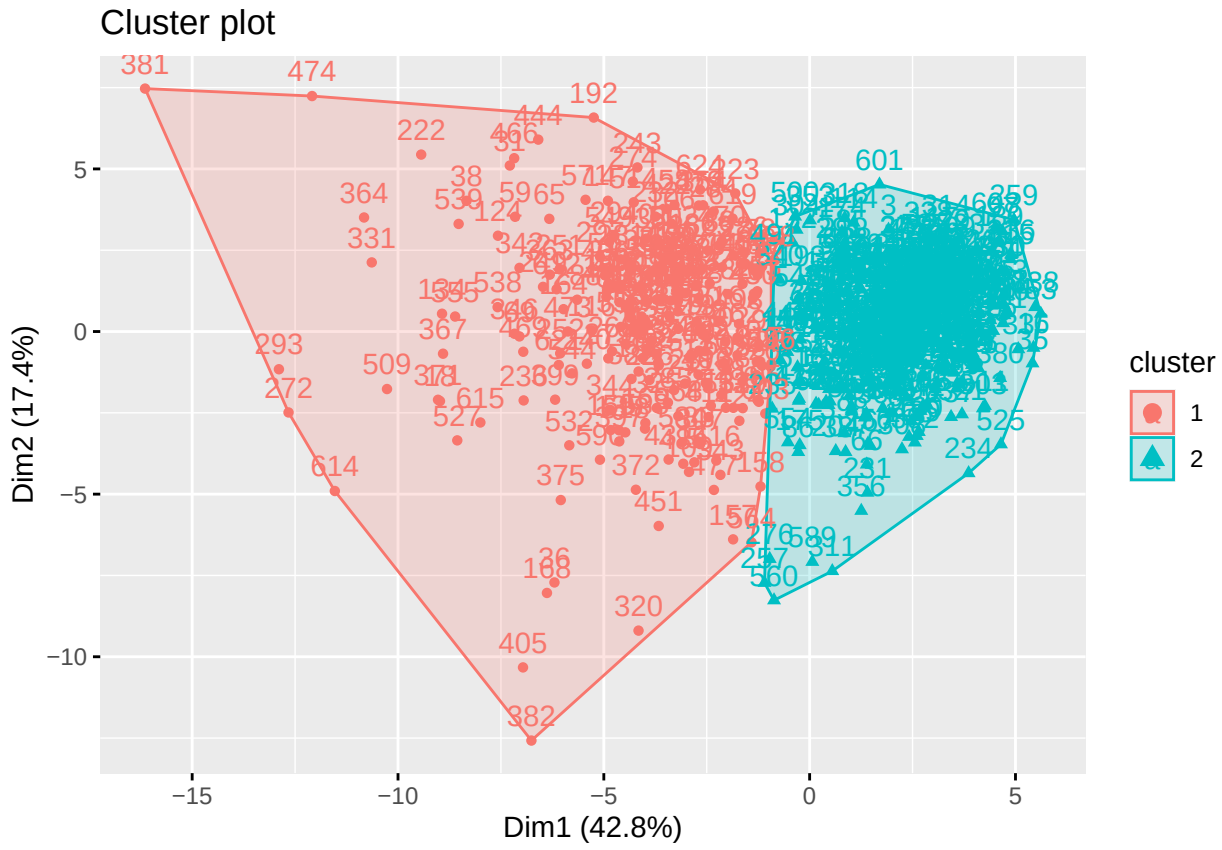
ANSWER: See clusters in plot below

```
# using code from the webpage linked in tutorial 2 by datanovia and
# factoextra

# install.packages('factoextra') # for beautiful graph of clusters
library(factoextra)
```

```
## Welcome! Want to learn more? See two factoextra-related books at https://goo.gl/ve3WBa
```

```
# must scale data first
ovarian.features <- scale(ovarian.dataset[, c(3:32)])
```

```
# Compute k-means with k = 2 set.seed(123)
kmeans2 <- kmeans(ovarian.features, 2, nstart = 25)

# plotting results
fviz_cluster(kmeans2, data = ovarian.features)
```




```
# Print the results print(km.res)
```

PART 2: What is the concordance between the clusters that you have identified and the true labels of the cells (Benign vs Malignant). *ANSWER: Accuracy, precision, and recall listed below! 92%, 91%, and 96%, respectively. This suggests we have a well-performing clustering model, i.e. it did a good job at separating and labelling our data.*

```
# First must convert the clusters into diagnosis labels
cluster <- kmeans2$cluster
predicted.diagnosis <- factor(ifelse(cluster == 1, "M", "B"))  #code from Hint
expected.diagnosis <- factor(ovarian.dataset$diagnosis)

# #Install required packages #install.packages('caret')
# library(caret) #library needed to create confusion matrix

# create confusion matrix using table() function
confusionMatrix <- table(predicted.diagnosis, expected.diagnosis)
confusionMatrix
```

```
##                    expected.diagnosis
## predicted.diagnosis   B   M
##                   B 371  35
##                   M  14 205
```

```
tb <- confusionMatrix[1, 1]  #true benign
tm <- confusionMatrix[2, 2]  #true malignant
fb <- confusionMatrix[1, 2]  #false benign
fm <- confusionMatrix[2, 1]  #false malignant

acc <- (tb + tm)/(tb + fb + fm + tm) * 100
sprintf("Accuracy is: %f percent", acc)
```

```
## [1] "Accuracy is: 92.160000 percent"
```

```
prec <- tb/(tb + fb) * 100
sprintf("Precision is: %f percent", prec)
```

```
## [1] "Precision is: 91.379310 percent"
```

```
recall <- tb/(tb + fm) * 100
sprintf("Recall is: %f percent", recall)
```

```
## [1] "Recall is: 96.363636 percent"
```

Q2.2. Repeat the kmeans analysis 10 times and report the mean accuracy across the 10 runs. Why are the results different in each run?

ANSWER: The results differ each time you run k-means, because there is a 50% chance that the model labels the B's as 0's, vice versa. To correct for this, we subtracted 100% from the accuracies that were less than 50%. This happens because k-means randomly picks initial cluster centres, and then iterates to minimize the within-cluster sum of squares error, leading to a different mean accuracy across the 10 runs every time you run k-means.

```
ten.runs <- c(1:10)  #initialize a variable

for (i in 1:10) {
    # 2 for 2 clusters, >=25 recommended for nStart
    kmeans2 <- kmeans(ovarian.features, 2, nstart = 25)
    cluster <- kmeans2$cluster
    predicted.diagnosis <- factor(ifelse(cluster == 1, "B", "M"))  #code from Hint.

    # create confusion matrix using table() function
    confusionMatrix <- table(predicted.diagnosis, expected.diagnosis)

    tb <- confusionMatrix[1, 1]  #true benign
    tm <- confusionMatrix[2, 2]  #true malignant
    fb <- confusionMatrix[1, 2]  #false benign
    fm <- confusionMatrix[2, 1]  #false malignant
```

```
    accuracy <- (tb + tm)/(tb + fb + fm + tm)

    # if statement to account the k-means giving the reverse of B or
    # M We do this because there is always a 50% chance the labels
    # will 0 for B or M, vice versa.
    if (accuracy < 0.5) {
        accuracy <- 1 - accuracy
    }

    ten.runs[i] <- accuracy
}
ten.runs * 100
```

```
##  [1] 92.16 92.16 92.16 92.16 92.16 92.16 92.16 92.16 92.16 92.16
```

```
sprintf("Mean Accuracy across the 10 runs is: %f percent", mean(ten.runs) *
    100)
```

```
## [1] "Mean Accuracy across the 10 runs is: 92.160000 percent"
```

Q2.3. Repeat the same analysis but with the top 5 PCs.

```
ten.runs.pca <- c(1:10)

for (i in 1:10) {
    data.kmean.pca <- kmeans(ovarian.pca$x[, 1:5], 2, nstart = 25)
    cluster <- data.kmean.pca$cluster
    predicted.diagnosis <- factor(ifelse(cluster == 1, "B", "M"))  #code from Hint

    # create confusion matrix using table() function
    confusionMatrix <- table(predicted.diagnosis, expected.diagnosis)

    tb <- confusionMatrix[1, 1]  #true benign
    tm <- confusionMatrix[2, 2]  #true malignant
    fb <- confusionMatrix[1, 2]  #false benign
    fm <- confusionMatrix[2, 1]  #false malignant

    accuracy <- (tb + tm)/(tb + fb + fm + tm)

    # if statement to account the k-means giving the reverse of B or
    # M We do this because there is always a 50% chance the labels
    # will 0 for B or M, vice versa.
    if (accuracy < 0.5) {
        accuracy <- 1 - accuracy
    }

    ten.runs.pca[i] <- accuracy
}
ten.runs.pca * 100
```

```
##  [1] 91.84 91.84 91.84 91.84 91.84 91.84 91.84 91.84 91.84 91.84
```

```
sprintf("Mean Accuracy across the 10 runs (but using top 5 PCs) is: %f percent",
    mean(ten.runs.pca) * 100)
```

```
## [1] "Mean Accuracy across the 10 runs (but using top 5 PCs) is: 91.840000 percent"
```

Q2.4. Compare the results between Q2.2. and Q2.3.

ANSWER: The mean accuracy across the 10 runs was 92.16% for using k-means alone for clustering the original data set, but it was 91.84% for clustering the top 5 principle components. This suggests to us that when using only the top 5 PCs in this case, we ended up losing some of the important information/features that help describe the variance of our observations, leading to a reducing in the performance of our model.

To improve this clustering model, we could chose to either only use k-means (no PCA), or we could try to use PCA but with more of the PCs, e.g. we know that the top 9 PCs captures 90% of the variance, so we could try top 9 PCs instead of top 5. Conversely, the top 5 PCs only captured 80% of the variance.

Q3. CLASSIFICATION

Set up:

```
# Divide your data into training and test sets using the following
# command:
temp1 <- ovarian.dataset[sample(nrow(ovarian.dataset))[1:(nrow(ovarian.dataset)/2)],
    ]
ovarian.dataset.train <- temp1

temp2 <- ovarian.dataset[sample(nrow(ovarian.dataset))[(nrow(ovarian.dataset)/2):(nrow(ovarian.dataset)
    ]
ovarian.dataset.test <- temp2
#'sample' randomly takes half of the data as training data, other half for testing

# install.packages('ISLR') #in console require(ISLR)

# Need to see how our output is dependent on our input. To do that,
# we can use correlation, ie how much the variation in our output is
# realted to our input.  Or, if i change the input, how much that
# affects the output.

# # install this for correlation install.packages('corrplot')
# library(corrplot) correlations <- cor(ovarian.dataset.train[,
# 3:32]) #correlations corrplot(correlations, method = 'circle')

# Now, we can see the correlation plot above. We used circles. Rows
# and cols are features.  We are measuring correlation between allllll
# the pairs of features.  Larger the circle, higher the correlation.
# Dark blue means huge correlation with the data
```

Q3.1. Design a logistic regression classifier to identify (differentiate) benign and malignant cells.

Part 1: Report the performance of the classification technique on the training and test sets. You can report accuracy, precision and recall.

```
# We need to first convert all the char variables to be numbers (help
# from TA!) note we don't know if M becomes 1 or 0, vice versa for B.
# We just trust the function.
ovarian.dataset.train$diagnosis <- as.factor(ovarian.dataset.train$diagnosis)

expected <- ovarian.dataset.train$diagnosis

# trying to scale the data first, but gives errors!
# ovarian.dataset.train <-
# as.data.frame(scale(ovarian.dataset.train[3:32]))
# ovarian.dataset.test <-
# as.data.frame(scale(ovarian.dataset.test[3:32]))

# Creating the logistic regression model. using the binomial argument
# to make it clear that the intent is to git a logistic regression
# model the '.' means we are taking all the features, but not
# including cell_id nor diagnosis
log.reg.model <- glm(expected ~ . - cell_id - diagnosis, data = ovarian.dataset.train,
    family = binomial)
```

```
## Warning: glm.fit: algorithm did not converge
```

```
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
```

```
summary(log.reg.model)
```

```
##
## Call:
## glm(formula = expected ~ . - cell_id - diagnosis, family = binomial,
##     data = ovarian.dataset.train)
##
## Coefficients:
##               Estimate Std. Error z value Pr(>|z|)
## (Intercept) -1.568e+03  1.602e+06  -0.001    0.999
## perimeter    3.779e+01  2.093e+05   0.000    1.000
## area        -7.579e+00  1.915e+04   0.000    1.000
## smoothness  -1.457e+00  9.054e+03   0.000    1.000
## symmetry    -8.443e-01  2.726e+03   0.000    1.000
## concavity    1.651e+03  3.882e+06   0.000    1.000
## protein1    -4.246e+03  3.045e+06  -0.001    0.999
## protein2     1.061e+03  1.928e+06   0.001    1.000
## protein3     5.077e+03  6.470e+06   0.001    0.999
## protein4     2.295e+02  2.135e+06   0.000    1.000
## protein5     1.096e+04  2.285e+07   0.000    1.000
## protein6     1.960e+02  6.478e+05   0.000    1.000
## protein7    -4.574e+01  1.581e+05   0.000    1.000
```

```
## protein8     6.025e+01  4.660e+04   0.001    0.999
## protein9    -1.212e+00  4.674e+03   0.000    1.000
## protein10   -1.472e+04  3.433e+07   0.000    1.000
## protein11    3.320e+03  5.520e+06   0.001    1.000
## protein12   -1.359e+03  3.238e+06   0.000    1.000
## protein13   -5.057e+03  1.632e+07   0.000    1.000
## protein14   -5.014e+03  1.090e+07   0.000    1.000
## protein15    1.279e+03  3.694e+07   0.000    1.000
## protein16    9.929e+00  1.024e+05   0.000    1.000
## protein17    1.446e+01  2.009e+04   0.001    0.999
## protein18    1.896e+00  6.520e+03   0.000    1.000
## protein19    2.953e-01  8.592e+02   0.000    1.000
## protein20    1.479e+03  3.961e+06   0.000    1.000
## protein21   -7.354e+01  5.954e+05   0.000    1.000
## protein22    1.966e+02  4.296e+05   0.000    1.000
## protein23   -7.840e+01  5.502e+06   0.000    1.000
## protein24    1.000e+03  1.241e+06   0.001    0.999
## protein25   -3.778e+03  3.472e+06  -0.001    0.999
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 4.1282e+02  on 311  degrees of freedom
## Residual deviance: 1.6063e-07  on 281  degrees of freedom
## AIC: 62
##
## Number of Fisher Scoring iterations: 25
```

```
# note that this gives the error: 'glm.fit: algorithm did not
# converge glm.fit: fitted probabilities numerically 0 or 1 occurred'
# but, that just means that the function was able to perfectly
# separate the variable into 0's and 1's. this is what we wanted!
```

Note that ALL the p-values are so HIGH! Thus, none of the coefficients are significant here. To improve the model, you would drop the features based on which ones have high/bad p value.

We now have the model trained, but need to get the prediction from it. Need to extract the probabilities from the model, using the prediction function!

```
# input our model, then features, then the type. If you don't use the
# type, it will NOT GIVE YOU THE PROBABILITY!!!!
probabilities <- predict(log.reg.model, ovarian.dataset.train, type = "response")
# the function knows which features to use in the dataset since we
# set it up in glm

probabilities[1:5]
```

```
##          214           26          474          172          354
## 2.220446e-16 2.220446e-16 1.000000e+00 2.220446e-16 1.000000e+00
```

These probabilities are very far from 0.5, ie they are extremely close to 0 or 1, which is good! Our model is classifying strongly.

So we have the probabilities, but how to convert them to labels? We simply need to use ifelse to do this. It will convert each of the probabilities to the B or M label.

```
prediction <- ifelse(probabilities > 0.5, "M", "B")

expected <- ovarian.dataset.train$diagnosis

confusionMatrix <- table(prediction, expected)
confusionMatrix
```

```
##           expected
## prediction   B   M
##          B 195   0
##          M   0 117
```

There are no false positives/malignant or false negatives/benign. This makes sense because we are testing against our training set, so of course our model seems to classify "perfectly".

Using this table, we can manually calculate many statistics such as accuracy, precision, and recall of the prediction.

```
tb <- confusionMatrix[1, 1]  #true benign
tm <- confusionMatrix[2, 2]  #true malignant
fb <- confusionMatrix[1, 2]  #false benign
fm <- confusionMatrix[2, 1]  #false malignant

acc <- (tb + tm)/(tb + fb + fm + tm) * 100
sprintf("Accuracy is: %f", acc)
```

```
## [1] "Accuracy is: 100.000000"
```

```
prec <- tb/(tb + fb) * 100
sprintf("Precision is: %f", prec)
```

```
## [1] "Precision is: 100.000000"
```

```
recall <- tb/(tb + fm) * 100
sprintf("Recall is: %f", recall)
```

```
## [1] "Recall is: 100.000000"
```

ANSWER: Thus, the accuracy, precision, and recall of our model all seems to be 100% - which makes sense because we are testing our model with the training set we used to build the model in the first place!

PART 2: Compare the performance of the classifier on the training and test set and provide a reason as to why one is better than the other.

```
# repeating all the steps as above, except putting our *test* set in
# instead of training

ovarian.dataset.test$diagnosis <- as.factor(ovarian.dataset.test$diagnosis)
expected.test <- ovarian.dataset.test$diagnosis

# log.reg.model.test <- glm(expected.test ~. - cell_id - diagnosis,
```

```
# data = ovarian.dataset.test, family = binomial)
# summary(log.reg.model.test)

probabilities.test <- predict(log.reg.model, ovarian.dataset.test, type = "response")

# probabilities.test[1:5]

prediction.test <- ifelse(probabilities.test > 0.5, "M", "B")

confusionMatrix.test <- table(prediction.test, expected.test)
confusionMatrix.test
```

```
##                expected.test
## prediction.test   B   M
##               B 184   6
##               M   7 116
```

```
tb <- confusionMatrix.test[1, 1]  #true benign
tm <- confusionMatrix.test[2, 2]  #true malignant
fb <- confusionMatrix.test[1, 2]  #false benign
fm <- confusionMatrix.test[2, 1]  #false malignant

acc <- (tb + tm)/(tb + fb + fm + tm) * 100
sprintf("(3.1) Accuracy is: %f", acc)
```

```
## [1] "(3.1) Accuracy is: 95.846645"
```

```
prec <- tb/(tb + fb) * 100
sprintf("(3.1) Precision is: %f", prec)
```

```
## [1] "(3.1) Precision is: 96.842105"
```

```
recall <- tb/(tb + fm) * 100
sprintf("(3.1) Recall is: %f", recall)
```

```
## [1] "(3.1) Recall is: 96.335079"
```

ANSWER: The accuracy, precision, and recall from running our test set through our model are very high - but not all 100% like we saw before. This makes sense, because it's the first time the model is seeing this data set (test), so it's expected to make some mistakes in classifying.

Q3.2. Repeat the same task as Q3.1. with the top 5 PCs.

```
# Apply PCA on raw training set
train.pca <- prcomp(ovarian.dataset.train[, c(3:32)], center = TRUE, scale. = TRUE)
summary(train.pca)
```

```
## Importance of components:
##                            PC1    PC2     PC3     PC4     PC5     PC6     PC7
## Standard deviation     3.5205 2.3863 1.68673 1.40414 1.23756 1.02973 0.79659
## Proportion of Variance 0.4131 0.1898 0.09483 0.06572 0.05105 0.03534 0.02115
## Cumulative Proportion  0.4131 0.6030 0.69780 0.76352 0.81457 0.84991 0.87107
##                            PC8    PC9   PC10    PC11    PC12    PC13    PC14
## Standard deviation     0.72632 0.6752 0.6125 0.59815 0.57258 0.51313 0.48805
## Proportion of Variance 0.01758 0.0152 0.0125 0.01193 0.01093 0.00878 0.00794
## Cumulative Proportion  0.88865 0.9039 0.9163 0.92828 0.93921 0.94798 0.95592
##                           PC15    PC16   PC17    PC18    PC19    PC20    PC21
## Standard deviation     0.44975 0.42193 0.3874 0.36888 0.32586 0.31285 0.29654
## Proportion of Variance 0.00674 0.00593 0.0050 0.00454 0.00354 0.00326 0.00293
## Cumulative Proportion  0.96266 0.96860 0.9736 0.97814 0.98168 0.98494 0.98787
##                           PC22    PC23   PC24    PC25    PC26    PC27    PC28
## Standard deviation     0.26096 0.24946 0.2325 0.20686 0.19844 0.18824 0.17015
## Proportion of Variance 0.00227 0.00207 0.0018 0.00143 0.00131 0.00118 0.00096
## Cumulative Proportion  0.99014 0.99222 0.9940 0.99544 0.99676 0.99794 0.99890
##                           PC29    PC30
## Standard deviation     0.13936 0.11626
## Proportion of Variance 0.00065 0.00045
## Cumulative Proportion  0.99955 1.00000
```

```
top5PCs <- train.pca$x[, 1:5]  #pull out the top 5 principle components

# Converting test set to compare with.
test.pca <- predict(train.pca, ovarian.dataset.test[, c(3:32)], type = "response")
```

```
## Warning: In predict.prcomp(train.pca, ovarian.dataset.test[, c(3:32)], type = "response") :
##  extra argument 'type' will be disregarded
```

```
ovarian.dataset.train$diagnosis <- as.factor(ovarian.dataset.train$diagnosis)

top5PCs_frame <- data.frame(top5PCs, diagnosis = ovarian.dataset.train$diagnosis)
test.pca_frame <- data.frame(test.pca, diagnosis = ovarian.dataset.test$diagnosis)

log.reg.model.pca <- glm(diagnosis ~ ., data = top5PCs_frame, family = binomial)
```

```
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
```

```
summary(log.reg.model.pca)
```

```
##
## Call:
## glm(formula = diagnosis ~ ., family = binomial, data = top5PCs_frame)
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept)  -0.6954     0.4546  -1.530 0.126055
## PC1           2.7551     0.5271   5.227 1.72e-07 ***
## PC2          -1.5263     0.4169  -3.661 0.000251 ***
## PC3          -0.6658     0.3293  -2.022 0.043166 *
## PC4          -0.5664     0.2513  -2.253 0.024240 *
```

```
## PC5           1.7611     0.6391   2.756 0.005857 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 412.815  on 311  degrees of freedom
## Residual deviance:  49.832  on 306  degrees of freedom
## AIC: 61.832
##
## Number of Fisher Scoring iterations: 9
```

```
probabilities.pca <- predict(log.reg.model.pca, test.pca_frame, type = "response")
probabilities.pca[1:5]
```

```
##            4           41          549           36          217
## 4.908184e-01 2.036569e-04 9.999999e-01 9.988389e-01 3.876346e-05
```

```
prediction <- ifelse(probabilities.pca > 0.5, "M", "B")

confusionMatrix <- table(prediction, ovarian.dataset.test$diagnosis)
confusionMatrix
```

```
##
## prediction   B   M
##          B 188   5
##          M   3 117
```

```
tb <- confusionMatrix[1, 1]  #true benign
tm <- confusionMatrix[2, 2]  #true malignant
fb <- confusionMatrix[1, 2]  #false benign
fm <- confusionMatrix[2, 1]  #false malignant

acc <- (tb + tm)/(tb + fb + fm + tm) * 100
sprintf("Accuracy using PCA is: %f", acc)
```

```
## [1] "Accuracy using PCA is: 97.444089"
```

```
prec <- tb/(tb + fb) * 100
sprintf("Precision using PCA is: %f", prec)
```

```
## [1] "Precision using PCA is: 97.409326"
```

```
recall <- tb/(tb + fm) * 100
sprintf("Recall using PCA is: %f", recall)
```

```
## [1] "Recall using PCA is: 98.429319"
```

Q3.3. Compare the results between Q3.1. and Q3.2. Do the results get better or worse? Why?

ANSWER: Without PCA we got 97.1% accuracy, but with using top 5 PCs, we got 97.7% accuracy. The accuracy got marginally better when using the top 5 PCs, i.e. the accuracy increased. This makes sense, because PCA allowed us to pick the features which describe the most variance in the data set, thus making our classifier's performance better. Note that the accuracy only got marginally better. This could suggest that a lot of the features in the original set were already important to consider for variance. Note that the training set and test set were randomly picked/divided - this may affect how well our model performed with and without PCA. Regardless, the difference in accuracy was marginal.

Q3.4. Compare the results of the clustering (Q2) and classification (Q3) methods. Which one gives you better result?

ANSWER: For clustering, without PCA we got 92.16% accuracy, but with using top 5 PCs, we got 91.84% accuracy. For classification, without PCA we got 97.1% accuracy, but with using top 5 PCs, we got 97.7% accuracy.

CLUSTERING vs CLASSIFICATION (not using PCA) *In terms of comparing clustering and classification methods, classification showed much higher accuracy (regardless of using PCs). This makes sense, because classification is a form of supervised learning, meaning we tell the model what the patterns/labels should be. Whereas clustering is unsupervised learning, meaning we have observations without any labels but the model needs to determine the labels based on the features.*

For clustering, we used k-means to minimize the within-cluster sum of squares error, and iterate multiple times. We recalculate the cluster centres again and again. It involves starting with a random assignment, so each time you run k-means you get a different clustering and accuracy. Clustering performs well if the intra-class similarity is high and the inter-class similarity is low. This helps explain why our results were not very accurate. Our cluster plot in Q2.1 showed that the 2 clusters were right beside each other, meaning that the inter-class similarity is high, which is not ideal. Additionally, the intra-class similarity is low, meaning the observations withing the clusters are quite spread out, which is again not ideal.

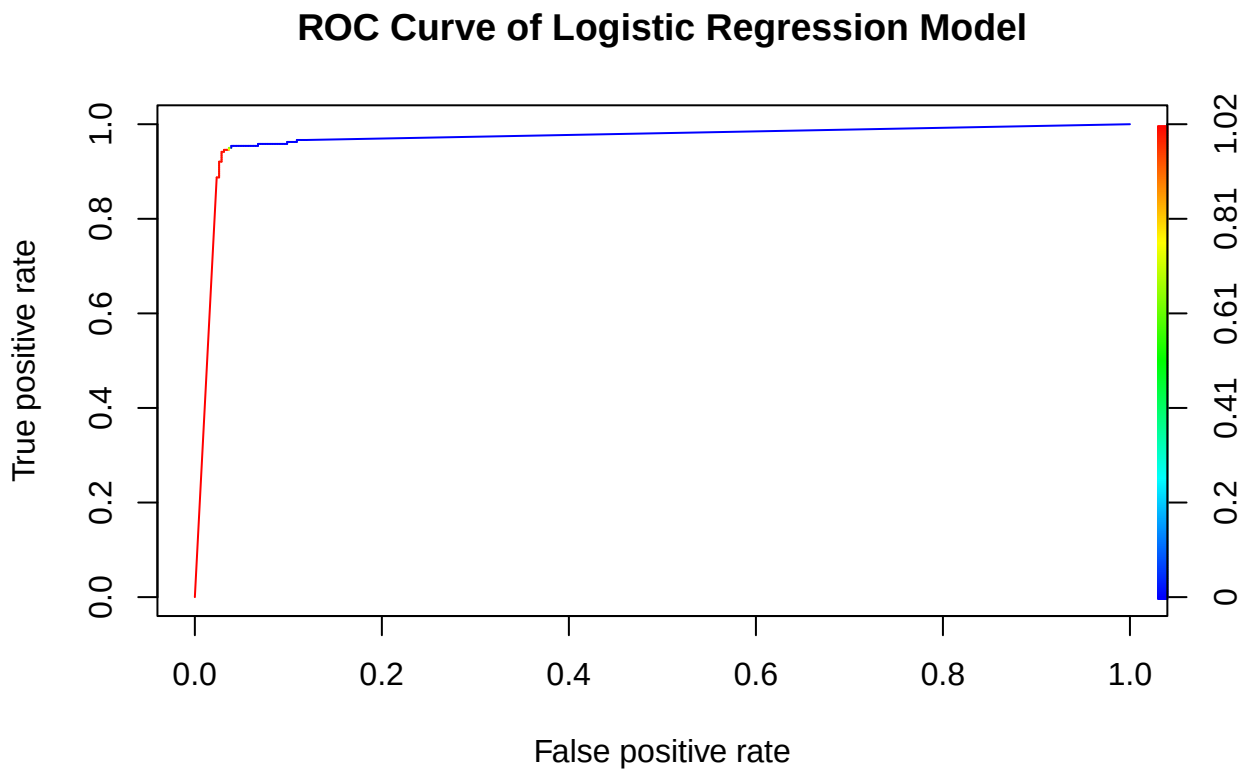
For logistic regression, we want to fit the data with a sigmoidal curve and then outputs an approximation of the probability of each point being a 0 or 1. It forces us to fit our data to the sigmoidal curve.

USING PCA FOR CLUSTERING vs USING PCA FOR CLASSIFIATION *In terms of comparing how each model did when using the top 5 PCs, we saw that the accuracy for the clustering model slightly decreased, whereas the accuracy for the classification model slightly increased. Note that this is not always true; it can depend on our data set (i.e. how the algorithm split the data into training and test) and our application.*

The decrease of accuracy for clustering using the top 5 PCs could be due to the fact that the top 5 PCs only captured 80% of the variance in our data set. More PCs would be needed to improve the accuracy of this model, for example we know that the top 9 PCs capture 90% of the variance, and the top 14 PCs give 95% of the variance, etc. K-means performs better when there is more variance in the data set, so excluding features (i.e. only using the top 5 PCs) reduced the performance/accuracy. Note that we randomly split the data set into training and test for classification, which may contribute to the differences in accuracy as well. The increase of accuracy for classification using the top 5 PCs is due to the fact that PCA allowed us to pick the features which describe the most variance in the data set, thus making our classifier's performance better.

Q3.5. Install the library "ROCR" and load it. Then, run the following lines using your trained logistic regression model:

```r
# install.packages('ROCR') #in console
library(ROCR)
pred.prob <- predict(log.reg.model, ovarian.dataset, type = "response")
predict <- prediction(pred.prob, ovarian.dataset$diagnosis, label.ordering = c("B",
    "M"))
perform <- performance(predict, "tpr", "fpr")
plot(perform, colorize = TRUE)
title("ROC Curve of Logistic Regression Model")
```




```r
# This will generate an ROC curve of the performance of the
# classifier for a range of thresholds.  Take a look at
# https://towardsdatascience.com/understanding-auc-roc-curve-68b2303cc9c5
# for a summary of ROC curves.
```

Given our ROC curve, what would you say it tells us about the overlap of the two classes? What can we say about the model's separability? How does an ROC curve tell us more about the model's performance than a single sensitivity/specificity measure?

ANSWER: Based on the shape of the ROC curve, there is very little overlap, suggesting that the model has high separability. This is because the AUC (area under ROC curve) is approximately 1 as seen from the plot, which indicates that the curves for the positive class (Malignant cell in our case) and negative class (Benign cell) don't overlap very much. This means that the model can strongly distinguish between B and M cells.

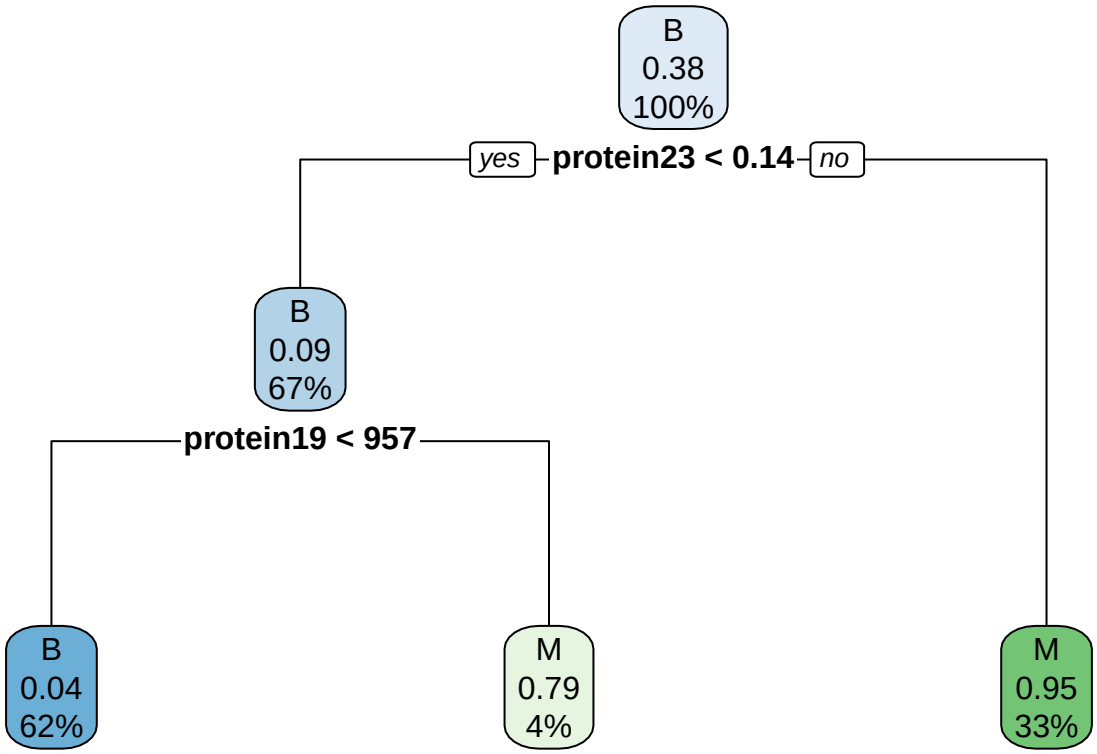
Comparing just accuracy is not always sufficient, but an ROC curve can tell us much more about our model; it can tell us how robust our model is for different applications. This is essentially a performance plot over a

range of different thresholds. The ROC also gives us sensitivity, which is another performance metric other than accuracy/precision/recall. Note that a single sensitivity measure is just one point on the ROC curve, but the ROC curve gives all the measures across possible thresholds for our model.

In other words, ROC curves help us determine what threshold to select based on our application, i.e. what threshold to pick if we want to have a very low false positive rate, or a very low true positive rate. This is especially helpful for imbalanced problems, which we are dealing with here.

Q3.6. Design another classifier (using a different classification method) and repeat Q3.1-3.

```
# 3.6.1 Designing a new classifier

# install.packages('rpart') #in console
# install.packages('rpart.plot')
library(rpart)
library(rpart.plot)

# classification method decision tree type = 'class' for binary model
tree.model <- rpart(diagnosis ~ ., data = ovarian.dataset.train, method = "class")
rpart.plot(tree.model, extra = 106)
```




```
# extra = display extra information at the nodes here '106' was
# chosen for our binary model but can just choose 'auto' to
# automatically select too
```

```r
# try tuning hyper-parameters to get higher accuracy
tree.control <- rpart.control(minsplit = 6, minbucket = round(5/3), cp = 0,
    maxdepth = 3)
# this method did not enhance the model's accuracy so will comment
# out perhaps the model could not be refined any more

# making predictions NOTE: output of predict() directly provides
# class labels in decision tree classifiers. Ie, no need to apply a
# threshold like we did for our logistic regression model before.
tree.prediction.test <- predict(tree.model, newdata = ovarian.dataset.test,
    type = "class", control = tree.control)
# tree.model is the object stored after model estimation newdata is
# the data that's used to make the prediction type = 'class' for the
# classification type of prediction

t.tree <- table(ovarian.dataset.test$diagnosis, tree.prediction.test)
# generating the confusion matrix for the test set based on the
# predictions
t.tree
```

```
##    tree.prediction.test
##       B   M
##   B 181  10
##   M   6 116
```

```r
accuracy.tree = (t.tree[2, 2] + t.tree[1, 1])/(t.tree[2, 2] + t.tree[1,
    2] + t.tree[1, 1] + t.tree[2, 1])
precision.tree = (t.tree[2, 2]/(t.tree[2, 2] + t.tree[1, 2]))
recall.tree = (t.tree[1, 1]/(t.tree[1, 1] + t.tree[2, 1]))

paste("Accuracy: ", round(accuracy.tree, 6))
```

```
## [1] "Accuracy:  0.948882"
```

```r
paste("Precision: ", round(precision.tree, 6))
```

```
## [1] "Precision:  0.920635"
```

```r
paste("Recall: ", round(recall.tree, 6))
```

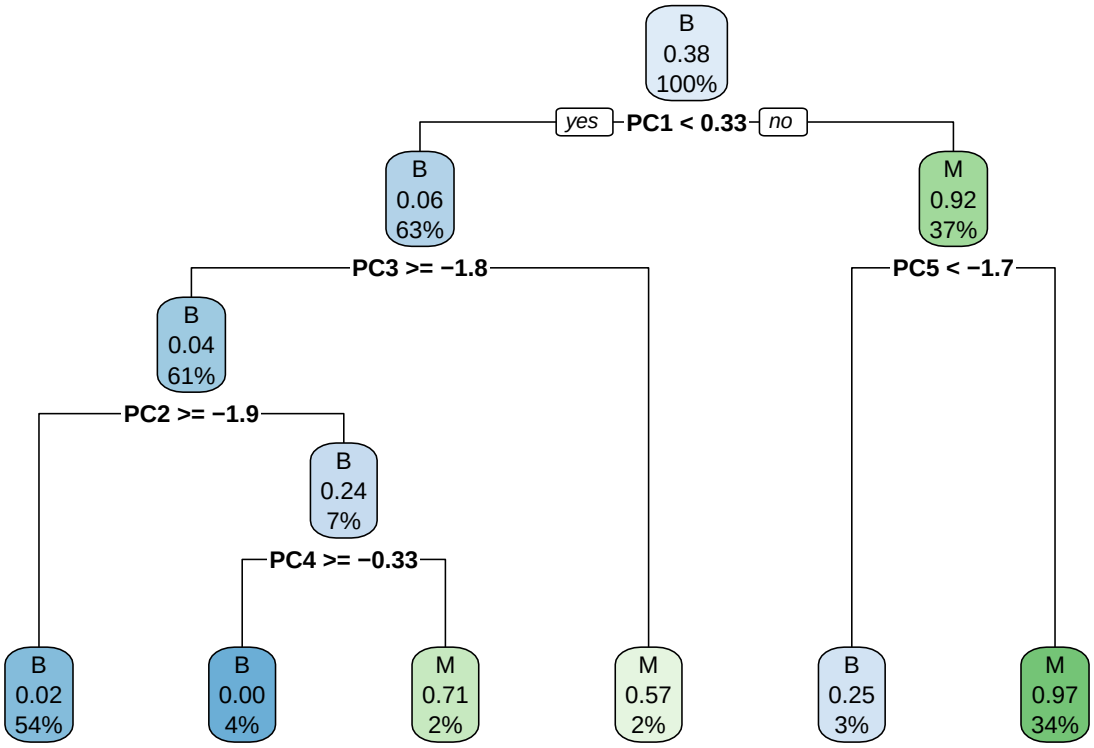
```
## [1] "Recall:  0.967914"
```

```r
# 3.6.2 Repeate 3.6.1 but with top 5 PCs same thing from Q3.2 here
data.pca2 <- prcomp(ovarian.dataset.train[, features], center = TRUE, scale. = TRUE)
summary(data.pca2)
```

```
## Importance of components:
##                           PC1    PC2     PC3     PC4     PC5     PC6     PC7
## Standard deviation     3.5205 2.3863 1.68673 1.40414 1.23756 1.02973 0.79659
## Proportion of Variance 0.4131 0.1898 0.09483 0.06572 0.05105 0.03534 0.02115
```

```
## Cumulative Proportion  0.4131 0.6030 0.69780 0.76352 0.81457 0.84991 0.87107
##                            PC8    PC9   PC10    PC11    PC12    PC13    PC14
## Standard deviation     0.72632 0.6752 0.6125 0.59815 0.57258 0.51313 0.48805
## Proportion of Variance 0.01758 0.0152 0.0125 0.01193 0.01093 0.00878 0.00794
## Cumulative Proportion  0.88865 0.9039 0.9163 0.92828 0.93921 0.94798 0.95592
##                           PC15    PC16   PC17    PC18    PC19    PC20    PC21
## Standard deviation     0.44975 0.42193 0.3874 0.36888 0.32586 0.31285 0.29654
## Proportion of Variance 0.00674 0.00593 0.0050 0.00454 0.00354 0.00326 0.00293
## Cumulative Proportion  0.96266 0.96860 0.9736 0.97814 0.98168 0.98494 0.98787
##                           PC22    PC23   PC24    PC25    PC26    PC27    PC28
## Standard deviation     0.26096 0.24946 0.2325 0.20686 0.19844 0.18824 0.17015
## Proportion of Variance 0.00227 0.00207 0.0018 0.00143 0.00131 0.00118 0.00096
## Cumulative Proportion  0.99014 0.99222 0.9940 0.99544 0.99676 0.99794 0.99890
##                           PC29    PC30
## Standard deviation     0.13936 0.11626
## Proportion of Variance 0.00065 0.00045
## Cumulative Proportion  0.99955 1.00000
```

```r
top_PCs <- data.pca2$x[, 1:5]

test.predict <- predict(data.pca2, ovarian.dataset.test)
ovarian.dataset.train$diagnosis <- as.factor(ovarian.dataset.train$diagnosis)

# next, create a dataframe with the PC and the target variable
df_pca <- data.frame(top_PCs, diagnosis = ovarian.dataset.train$diagnosis)


# now fit the decision tree model with df_pca here diagnosis =
# ovarian.dataset.train$diagnosis which is the target variable
tree.model.pca <- rpart(diagnosis ~ ., data = df_pca, method = "class")
# making a decision tree plot
rpart.plot(tree.model.pca, extra = 106)
```




```
# make predictions on the training set based on the decision tree
# model developed by the top PCAs
tree.prediction.test.pca <- predict(tree.model.pca, newdata = as.data.frame(test.predict),
    type = "class")

t.tree.pca <- table(ovarian.dataset.test$diagnosis, tree.prediction.test.pca)
t.tree.pca
```

```
##    tree.prediction.test.pca
##       B   M
##   B 181  10
##   M   8 114
```

```
accuracy.tree.pca = (t.tree.pca[2, 2] + t.tree.pca[1, 1])/(t.tree.pca[2,
    2] + t.tree.pca[1, 2] + t.tree.pca[1, 1] + t.tree.pca[2, 1])
precision.tree.pca = (t.tree.pca[2, 2]/(t.tree.pca[2, 2] + t.tree.pca[1,
    2]))
recall.tree.pca = (t.tree[1, 1]/(t.tree[1, 1] + t.tree[2, 1]))

paste("Accuracy: ", round(accuracy.tree.pca, 6))
```

```
## [1] "Accuracy:  0.942492"
```

```
paste("Precision: ", round(precision.tree.pca, 6))
```

```
## [1] "Precision:  0.919355"
```

```
paste("Recall: ", round(recall.tree.pca, 6))
```

```
## [1] "Recall:  0.967914"
```

3.6.3 Compare results of clustering & classification

ANSWER: Comparing the results from 3.6.1 and 3.6.2, we noticed that the non-PCA model actually performed better with higher accuracy and precision. The accuracy of using decision tree was 93.9%, but with using the top 5 PCs it was 95.2%.

This means that perhaps using the decision tree classifier without PCA analysis is good enough for this data set. Again, the use of PCA can lead to loss of information and worsen the performance of classifier, especially when we only chose the top 5 PCs in our analysis. Some of the variance in the original training data set could be important for classification and was lost during the process.

It is also worth noting that the overall accuracy and precision of the decision tree classifier is LOWER than that of the logistic regression model, and this is not uncommon or surprising A main reason for this is because decision tree models usually perform poorly on imbalanced data sets (our data set is imbalanced), since the tree may favour the majority class a lot. Other factors such as linearity, overfitting, and dimensionality also play a factor in the performance of decision tree models. For this particular data set, we would not consider using the decision tree model over the logistic regression model we made earlier, but it was still a good practice to experiment different models and compare their performance to determine the best fit.

Note that we did not do pruning, i.e. parameter control, in our decision tree. It didn't improve our model, however more iterations could be done in the future to increase the accuracy.