Homework 2

Kathleen Ashbaker

2024-02-01

1. Data exploration + Logistic Regression

Be sure to upload the wdbc.data file into your local directory prior to beginning this assignment.

```
# upload data set 'wdbc' from local directory

wdbc <- read.csv("C:/Users/Kathleen/OneDrive/Documents/BIOSTAT 546 Machine Learning/Homework/Homework 2/
```

```
# Be sure to upload appropriate libraries too!

library(ggplot2)
```

```
## Warning: package 'ggplot2' was built under R version 4.3.2
```

```
library(GGally) # extension of ggplot2
```

```
## Warning: package 'GGally' was built under R version 4.3.2

## Registered S3 method overwritten by 'GGally':
##   method from
##   +.gg   ggplot2
```

```
library(MASS)  # Implements LDA & QDA
library(caret) # confusionMatrix
```

```
## Loading required package: lattice
```

```
library(pROC)  # ROC & AUC
```

```
## Type 'citation("pROC")' for a citation.

##
## Attaching package: 'pROC'

## The following objects are masked from 'package:stats':
##
##     cov, smooth, var
```

```r
library(class) # Implements KNN
```

Here, we focus on the attributes in columns {2, 3, 4}, i.e. 2: Diagnosis (M = malignant, B = benign) 3: Average radius of the cell nuclei in each image 4: Average texture of the cell nuclei in each image

```r
# Assuming wdbc is already loaded
# Rename specific columns
names(wdbc)[1] <- "ID Number"
names(wdbc)[2] <- "Diagnosis"
names(wdbc)[3] <- "Average radius"
names(wdbc)[4] <- "Average texture"
```

```r
# Subset the Data Frame: Create a new data frame with only the first four columns of wdbc.

wdbc_subset <- wdbc[, 1:4]
```

```r
# optional : str(wdbc_subset)
```

```r
# check for missing values, this should output zero if the data set
# contains no missing values.
sum(is.na(wdbc_subset))
```

```
## [1] 0
```

```r
# Separate Predictors and Outcome
predictors <- wdbc_subset[, 3:4]
outcome <- wdbc_subset[, 2]
```

```r
# Convert the 'Diagnosis' column to a factor
outcome_factor <- as.factor(wdbc_subset$Diagnosis)
```

(a) Describe the data: sample size n, number of predictors p, and number of observations in each class

```r
# sample size, n is:
nrow(wdbc_subset)
```

```
## [1] 568
```

```r
# number of predictors, p is:
ncol(predictors)
```

```
## [1] 2
```

```r
# Count of each class
table(wdbc_subset$Diagnosis)
```

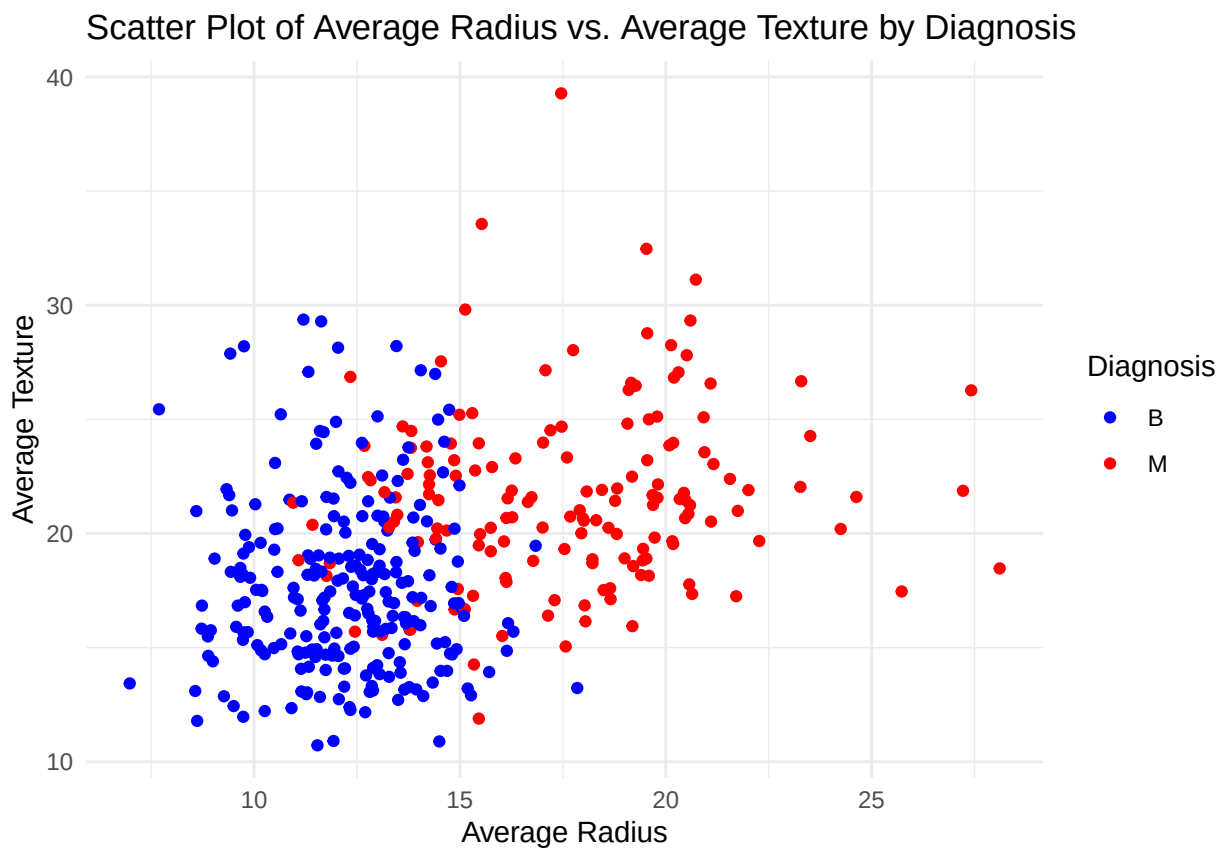
```
##
##   B   M
## 357 211
```

(b) Divide the data into a training set of 400 observations, and a test set; from now on, unless specified, work only on the training set.

```
# set.seed (0) for reproducibiltiy

set.seed(0)
train_id = sample(nrow(wdbc_subset), 400)
train_df = wdbc_subset[train_id,]
test_df  = wdbc_subset[-train_id,]
```

(c) Make a color-encoded scatterplot displaying Y(Diagnosis) and the predictors X1(Average radius – column 3), X2 (Average texture – column 4) on an x-y axis.



Based on this scatterplot, do you think it will be possible to accurately predict the outcome from the predictors? Motivate your answer.

Comment:

Based on the scatterplot, it appears that there is some degree of separation between the two classes (benign and malignant) when considering the average radius and average texture as predictors. The benign tumors (denoted by blue points) tend to have a smaller average radius and a more compact clustering in terms of average texture. In contrast, malignant tumors (denoted by red points) generally exhibit a larger average radius and are more dispersed in average texture.

While there is overlap between the two classes, the visible distinction suggests that a predictive model could potentially learn from these features to classify the outcomes with a certain level of accuracy. However, given the overlap, it may be challenging to achieve very high accuracy without considering additional features or

using sophisticated classification algorithms that can capture the complexity and patterns within the data. It's also important to note that the performance of the predictive model would need to be validated on a separate test set to ensure that it generalizes well to new, unseen data.

(d) Fit a logistic regression model to predict Y(Diagnosis)

```r
# Recoding Diagnosis as a binary numeric
#variable where 1 represents "M" (malignant)
#and 0 represents "B" (benign)
train_df$Diagnosis <- ifelse(train_df$Diagnosis == "M", 1, 0)
```

```
##
## Call:
## glm(formula = Diagnosis ~ `Average radius` + `Average texture`,
##     family = binomial(link = "logit"), data = train_df)
##
## Coefficients:
##                   Estimate Std. Error z value Pr(>|z|)
## (Intercept)       -19.1519     2.0341  -9.415  < 2e-16 ***
## `Average radius`    0.9678     0.1118   8.659  < 2e-16 ***
## `Average texture`   0.2486     0.0474   5.245 1.56e-07 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 535.00  on 399  degrees of freedom
## Residual deviance: 210.88  on 397  degrees of freedom
## AIC: 216.88
##
## Number of Fisher Scoring iterations: 7
```

Make a table, like Table 4.3 in the textbook, displaying the coefficient estimates, standard errors, and p-values.

```
##                      Estimate Std..Error   z.value     Pr...z..
## (Intercept)       -19.1519513 2.03414774 -9.415221 4.720863e-21
## `Average radius`    0.9678167 0.11176962  8.659031 4.757900e-18
## `Average texture`   0.2486162 0.04739874  5.245207 1.561067e-07
```

Give an interpretation of the values of the coefficient estimates

Interpretation of the Coefficients:

Coefficients Interpretation (Intercept) = -19.1519: The intercept represents the log odds of the outcome (being classified as having the disease, assuming "Diagnosis" is coded as 1 for disease presence) when all predictors are held at zero. Since both "Average radius" and "Average texture" are continuous variables, this value is more theoretical than practical, as these predictors cannot be exactly zero. The negative value indicates that when "Average radius" and "Average texture" are at their minimum (zero, hypothetically), the log odds of being diagnosed with the disease are very low (but remember, zero values for these predictors might not be meaningful or possible in your context).

Average radius = 0.9678: This coefficient means that for each one-unit increase in the "Average radius", holding "Average texture" constant, the log odds of being diagnosed with the disease increase by 0.9678.

This is a substantial increase and suggests that "Average radius" is a strong predictor of the diagnosis. Since the logistic regression operates on the log odds scale, to interpret this effect on the probability scale, you'd need to convert the log odds to odds (using the exponential function) and then to probability. The positive coefficient indicates a positive association between "Average radius" and the likelihood of disease diagnosis.

Average texture = 0.2486: Similarly, this coefficient indicates that for each one-unit increase in "Average texture", holding "Average radius" constant, the log odds of being diagnosed with the disease increase by 0.2486. This suggests a positive relationship between "Average texture" and the outcome, but the effect size is smaller compared to "Average radius", indicating that while "Average texture" is a significant predictor, its impact on the diagnosis is less pronounced than that of "Average radius".

Statistical Significance The p-values (Pr(>|z|)) for both predictors and the intercept are very small (< 2e-16 for the intercept and "Average radius", 1.56e-07 for "Average texture"), indicating strong evidence against the null hypothesis of no association, for both predictors with the outcome. This means that both "Average radius" and "Average texture" are statistically significant predictors of "Diagnosis".

(e) Use the coefficient estimates to manually calculate the predicted probability $P(Y = M \mid (X1, X2) = (10, 12))$ writing explicitly every step. Compare your results with the prediction computed with 'predict.'

```
# Manual calculation of predicted probability

# We wish to follow this formula:

# predicted_probability <- 1 / (1 + exp(-(-19.1519 + 0.9678 * 10 + 0.2486 * 12)))



# Given coefficient estimates, beta_0=Y, beta_1=Average radius, beta_2=Average texture
beta_0 <- -19.1519
beta_1 <- 0.9678
beta_2 <- 0.2486

# Values for X1 and X2
X1 <- 10
X2 <- 12

# Calculate the linear combination
linear_combination <- beta_0 + (beta_1 * X1) + (beta_2 * X2)

# Calculate the predicted probability using the logistic function
predicted_probability <- 1 / (1 + exp(-linear_combination))

predicted_probability
```

```
## [1] 0.001515187
```

```
# 'Predict' function implemented here:

# Assuming 'glm.model' is your logistic regression model fitted with 'train_df'

#glm.model <- glm(Diagnosis ~ `Average radius` + `Average texture`,
                 #family = binomial(link = "logit"), data = train_df)

#  Ensure to match the column names exactly as they were used in the model:
```

```r
new_data <- data.frame(`Average radius` = 10, `Average texture` = 12)

# Rename columns in new_data to match those in train_df
names(new_data) <- c("Average radius", "Average texture")
```

```r
# Use the predict function with the glm.model,
#specifying new_data as the dataset for which predictions
#are desired. The type = "response" option will return probabilities:

glm.prob.train <- predict(glm.model, newdata = new_data, type = "response")

glm.prob.train
```

```
##           1
## 0.001515656
```

Comment on Results:

When comparing the two methods of calculating the predicted probability P(Y = M | (X1, X2) = (10, 12)), we observe very close results, indicating consistency between the manual calculation and the predict function's computation using the logistic regression model in R.

Manual Calculation: The manual calculation of the predicted probability, using the provided coefficient estimates and the specified values for (X1, X2) = (10, 12), yielded a probability of approximately 0.001515187.

'predict' Function Calculation: Using the 'predict' function with the 'glm.model' and the specified 'new_data,' the calculated predicted probability was approximately 0.001515656.

Comparison: The slight difference between the two calculated probabilities (0.001515187 for the manual calculation vs. 0.001515656 from the 'predict' function) is minimal and can be attributed to rounding or numerical precision differences in the computation. Such small discrepancies are common in numerical calculations due to the way floating-point arithmetic is handled in computing environments.

Both methods effectively yield the same interpretation: Given the values (X1, X2) = (10, 12) for Average radius and Average texture, respectively, the probability of P(Y = M | (X1, X2) = (10, 12)), where Y=M (malignant diagnosis) is extremely low at approximately 0.15%.

This comparison confirms the accuracy and reliability of the predict function in R for calculating predicted probabilities based on a logistic regression model, and it validates the manual calculation method using the model's coefficient estimates.

(f) Use the glm previously fitted and the Bayes rule to compute the predicted outcome Y^ from the associated probability estimates (computed with predict) both on the training and the test set.

Then compute the confusion table and prediction accuracy (rate of correctly classified observations) both on the training and test set.

```r
# Step 1: Compute Probability Estimates
train_prob <- predict(glm.model, type = "response")

# Test Set
# ensure that test_df has
# the Diagnosis variable recoded similarly to train_df:

test_df$Diagnosis <- ifelse(test_df$Diagnosis == "M", 1, 0)
test_prob <- predict(glm.model, newdata = test_df, type = "response")
```

```r
# Step 2: Compute Predicted Outcome

# Training Set

train_pred <- ifelse(train_prob >= 0.5, 1, 0)

# Test Set

test_pred <- ifelse(test_prob >= 0.5, 1, 0)
```

```r
# Step 3: Evaluate Predictions for Training

train_accuracy <- mean(train_pred == train_df$Diagnosis)
```

```r
# Step 3: Evaluate Predictions for Test

test_accuracy <- mean(test_pred == test_df$Diagnosis)
```

```r
# Confusion matrix for the training set
train_confusion_matrix <- table(Predicted = train_pred, Actual = train_df$Diagnosis)

# Calculate accuracy for the training set
train_accuracy <- sum(diag(train_confusion_matrix)) / sum(train_confusion_matrix)

# Print the confusion matrix and accuracy for the training set
print(train_confusion_matrix)
```

```
##          Actual
## Predicted   0   1
##         0 229  28
##         1  15 128
```

```r
print(paste("Training Set Accuracy:", train_accuracy))
```

```
## [1] "Training Set Accuracy: 0.8925"
```

```r
# Confusion matrix for the test set
test_confusion_matrix <- table(Predicted = test_pred, Actual = test_df$Diagnosis)

# Calculate accuracy for the test set
test_accuracy <- sum(diag(test_confusion_matrix)) / sum(test_confusion_matrix)

# Print the confusion matrix and accuracy for the test set
print(test_confusion_matrix)
```

```
##          Actual
## Predicted   0   1
##         0 103   8
##         1  10  47
```

```r
print(paste("Test Set Accuracy:", test_accuracy))
```

```
## [1] "Test Set Accuracy: 0.892857142857143"
```

Comment on the results:

Training Set Results:

Confusion Matrix: For the training set, there were 229 true negatives, 128 true positives, 28 false negatives, and 15 false positives.

Accuracy: The accuracy on the training set was approximately 89.25%, indicating a high level of correct classifications by the model on the data it was trained on. Test Set Results:

Confusion Matrix: For the test set, the confusion matrix correctly shows 103 true negatives, 47 true positives, 8 false negatives, and 10 false positives.

Accuracy: The accuracy on the test set was approximately 89.29%, nearly matching the training set performance. This high accuracy indicates good generalization of the model to unseen data.

Overall, the logistic regression model shows excellent performance on both the training and test sets, with high accuracy levels exceeding 89% in both cases. This suggests that the model is well-fitted and generalizes well to new, unseen data. The similarity in accuracy between the training and test sets suggests that overfitting is minimal, and the model has learned the underlying patterns in the data effectively. The confusion matrices reveal a balanced performance in terms of sensitivity (true positive rate) and specificity (true negative rate), with the model effectively identifying both malignant and benign cases. The relatively low numbers of false negatives and false positives in the test set (8 and 10, respectively) are particularly noteworthy in the medical diagnosis context, where the costs of these types of errors can be very high. The model strikes a good balance, minimizing both types of errors on unseen data

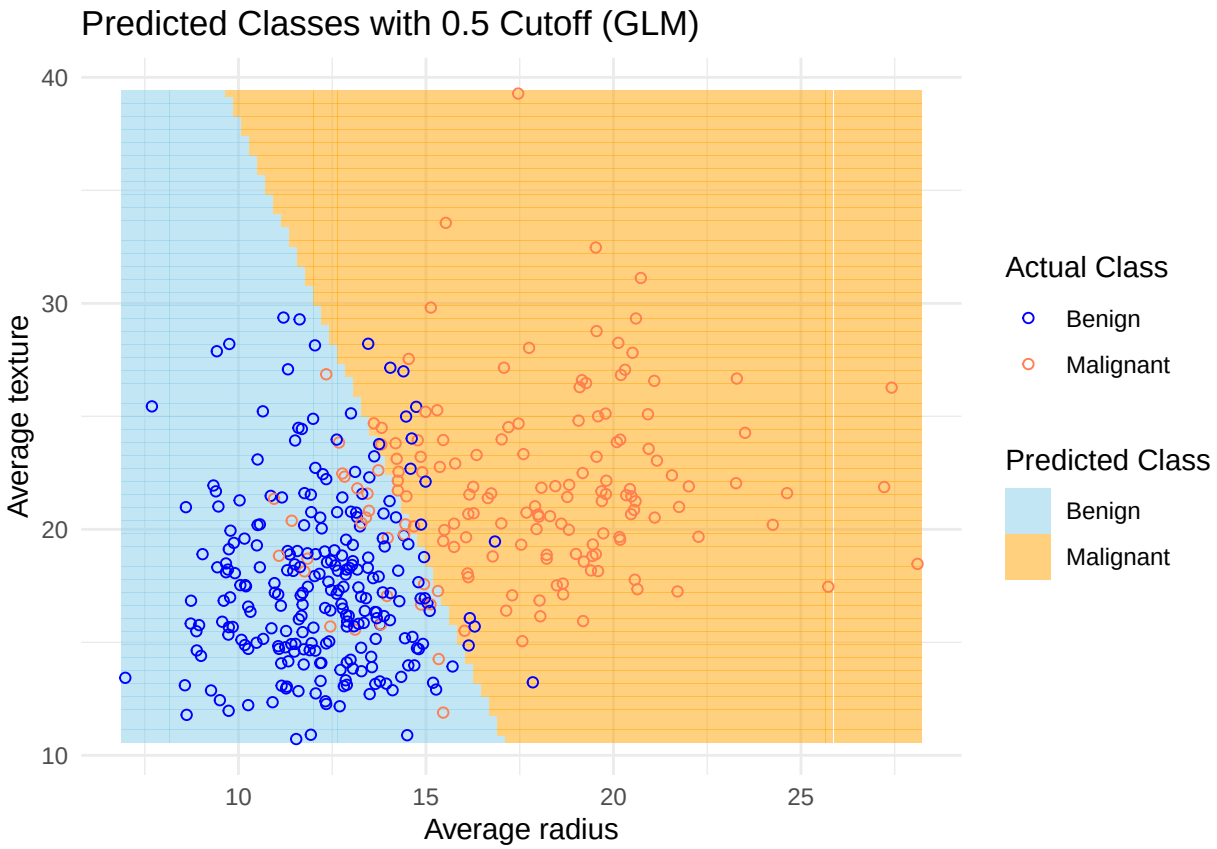
(g) Plot an image of the decision boundary (like the one in Figure 2.13 in the textbook, but without the purple dashed line) as follows:

-Generate a dense set (e.g. 10000 observations) of possible values for the predictors (X1, X2) within reasonable ranges; (the command expand.grid might come in handy)

-Use the glm model previously fitted to predict the outcome probabilities for every observation you have generated and use Bayes rule to compute the predicted outcomes;

-Plot predicted outcomes (color coded) and associated predictors in a 2D scatter plot together with the training set.

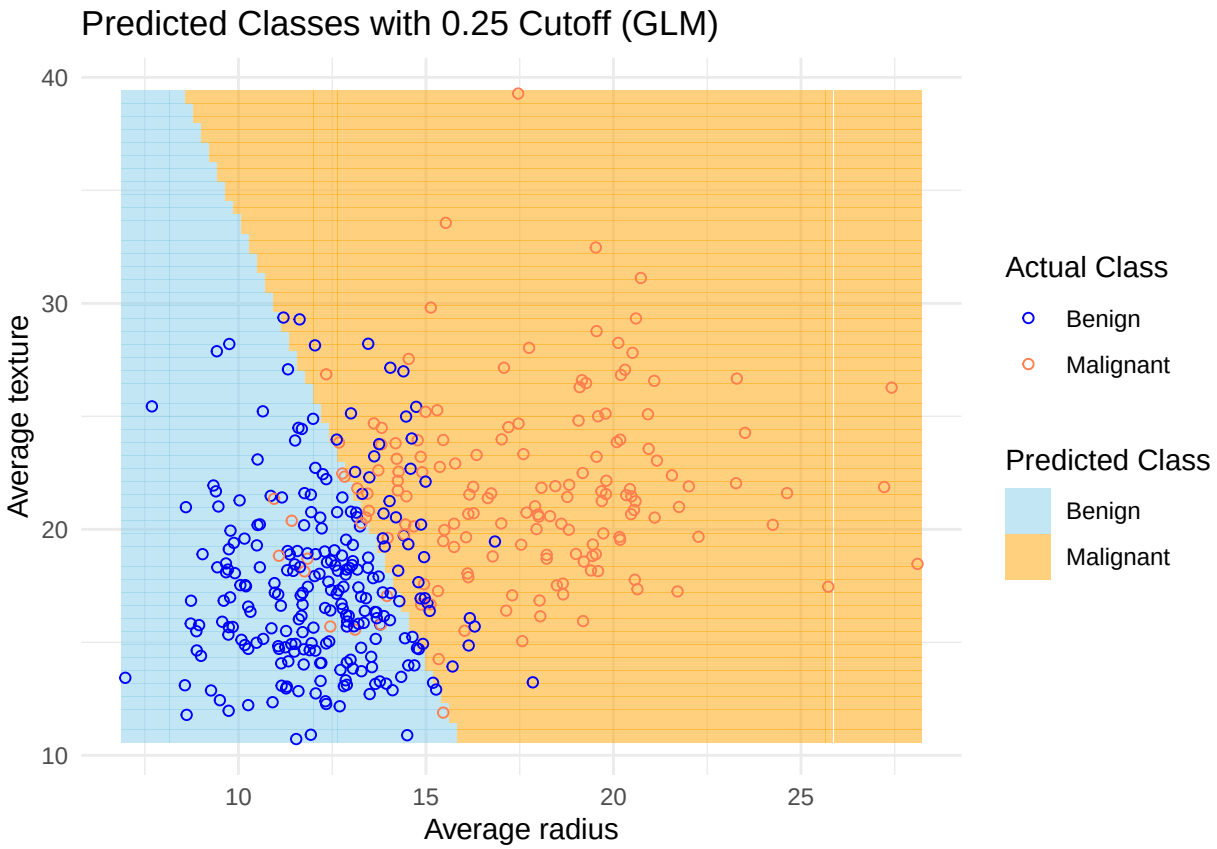
-Generate the same plot for probability cutoff values of 0.25 and 0.75.



Comment on the results:

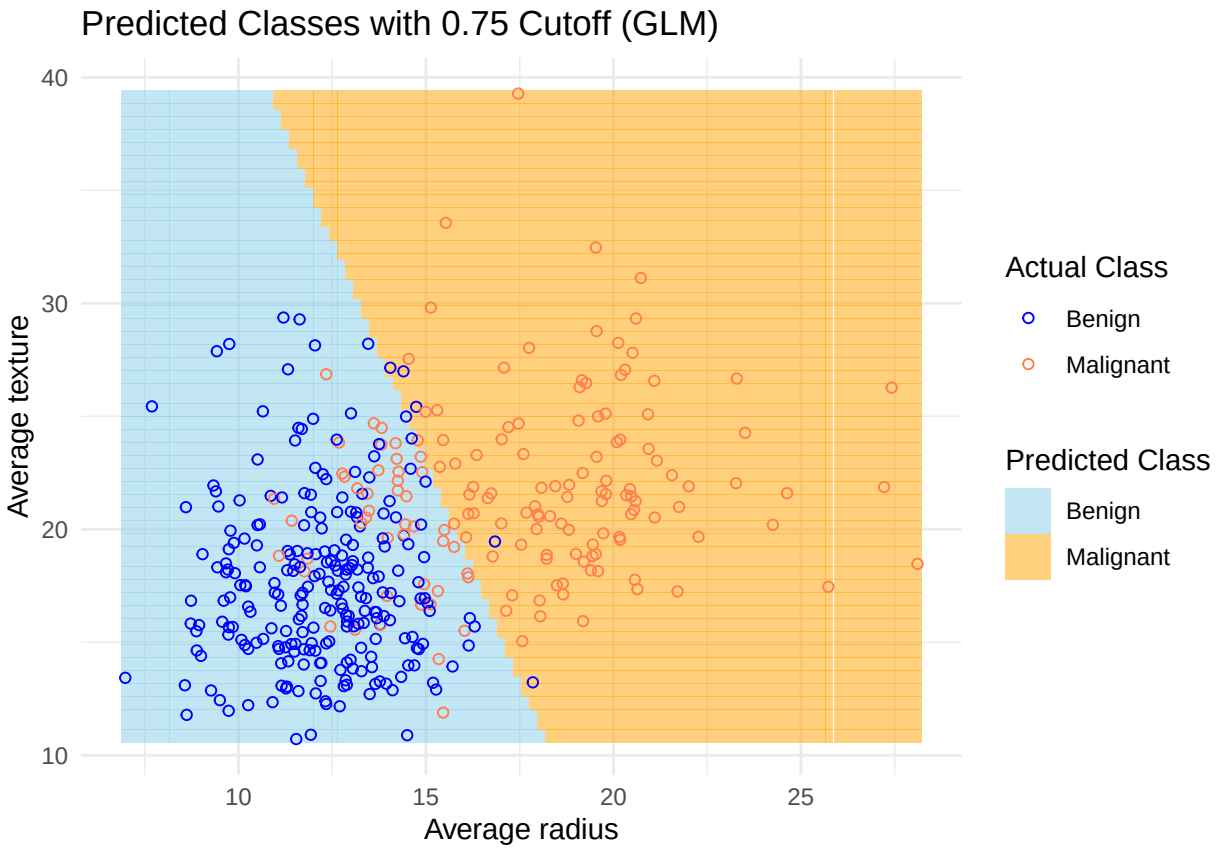
The plot shows a scatter plot of the training data points, with X1 (Average radius) on the x-axis and X2 (Average texture) on the y-axis. Points are color-coded based on the actual class labels from the train_df dataset, with one color representing benign (class 0) and another representing malignant (class 1) diagnoses. The background color indicates the predicted class labels generated by the logistic regression model (glm.model) and applied to the dense grid of potential predictor values created by expand.grid. The color represents the decision boundary based on a probability cutoff of 0.5, according to Bayes rule. The areas filled with different colors represent regions where new observations would be classified as either benign or malignant by the model. The overlay of actual data points allows for a visual assessment of how well the model's predicted classes align with the true classes. Ideally, most blue points (benign) should be within the blue region, and most orange points (malignant) should be within the orange region. The plot as a whole provides a visual interpretation of the model's decision boundary in the feature space defined by Average radius and Average texture. It can be used to intuitively understand the model's classification behavior and to identify areas where the model may be performing well or where it may be misclassifying observations. If the actual and predicted classes match well, the plot will show a clear and consistent separation between the two colors with minimal overlap. If there is significant overlap, it may indicate that the model has difficulty distinguishing between the two classes given the predictors used.

Now generate the same plot(s) for probability cutoff values of 0.25 and 0.75:



Comment:

Overall, the plot indicates that with a lower cutoff of 0.25, the model is biased towards predicting more cases as malignant. This can be beneficial in certain contexts where failing to detect a malignant case is much more costly than incorrectly classifying a benign case as malignant. However, it also suggests that the model might generate more false alarms, which could lead to unnecessary treatments or tests.



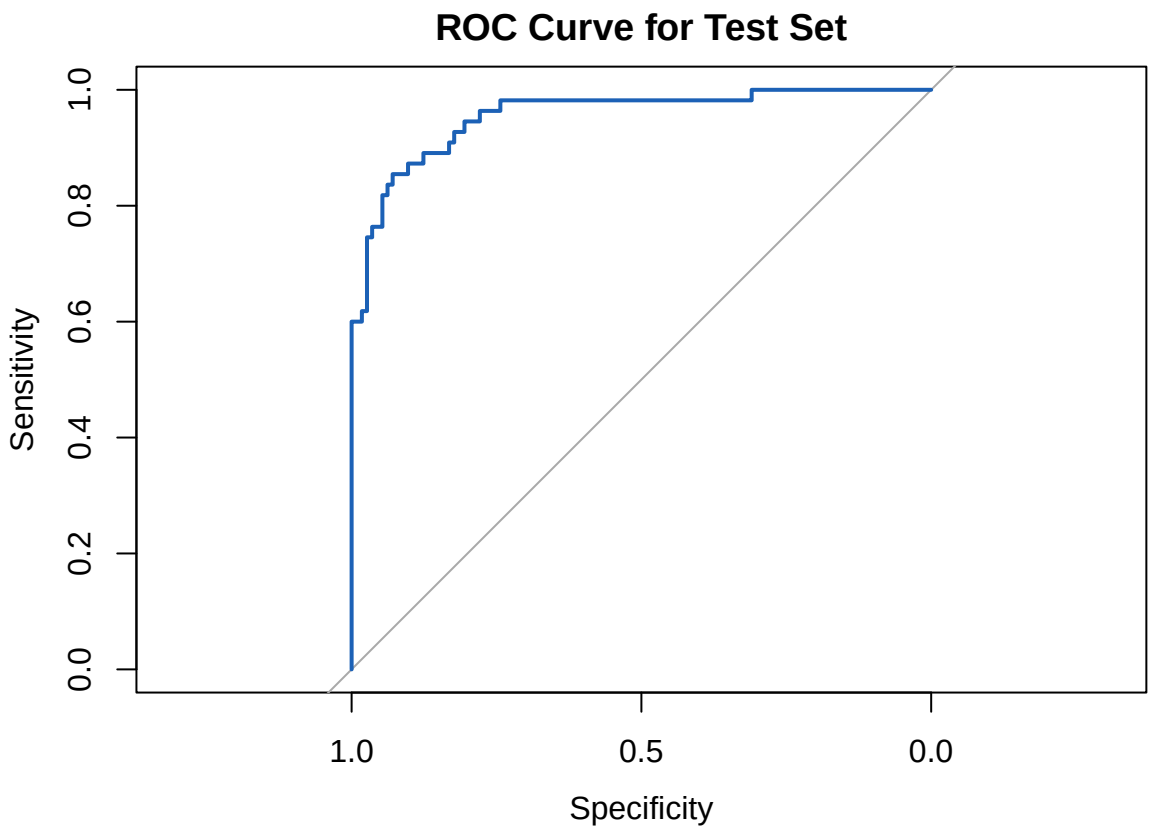
Comment:

The plot provides insight into the model's sensitivity and specificity at this threshold. By adjusting the cutoff, you can control the trade-off between these two metrics. A higher cutoff (like 0.75) prioritizes specificity, while a lower cutoff (like 0.25) prioritizes sensitivity. Overall, the plot with a 0.75 cutoff illustrates how the choice of threshold affects the classification of new observations and the model's overall performance in terms of type I and type II errors. It highlights the importance of selecting an appropriate cutoff value based on the specific context and the costs associated with different types of classification errors.

(h) Plot the ROC curve, computed on the test set.

```
## Setting levels: control = 0, case = 1
```

```
## Setting direction: controls < cases
```




```
# Add the AUC (Area Under the Curve) to the plot
auc(roc_obj)
```

```
## Area under the curve: 0.9553
```

2. Linear discriminant analysis model

(a) Now fit a linear discriminant analysis model to the training set you created in Exercise 1. Make a table displaying the estimated 'Prior probabilities of groups' and 'Group means'.

```
# Fit the LDA model to the training set
# Assuming 'Diagnosis' is the factor variable representing the class in your dataset
lda_model <- lda(Diagnosis ~ `Average radius` + `Average texture`, data = train_df)

#inspect model
lda_model
```

```
## Call:
## lda(Diagnosis ~ `Average radius` + `Average texture`, data = train_df)
##
## Prior probabilities of groups:
##    0    1
## 0.61 0.39
##
```

```
## Group means:
##   `Average radius` `Average texture`
## 0         12.22459          17.62635
## 1         17.65397          21.62968
##
## Coefficients of linear discriminants:
##                           LD1
## `Average radius`  0.3571385
## `Average texture` 0.1034597
```

```
# Display the estimated 'Prior probabilities of groups'
cat("Prior probabilities of groups:\n")
```

```
## Prior probabilities of groups:
```

```
print(lda_model$prior)
```

```
##    0    1
## 0.61 0.39
```

```
# Display the 'Group means'
cat("\nGroup means:\n")
```

```
##
## Group means:
```

```
print(lda_model$means)
```

```
##   `Average radius` `Average texture`
## 0         12.22459          17.62635
## 1         17.65397          21.62968
```

Describe in words the meaning of these estimates and how they are related to the posterior probabilities:

The output from the Linear Discriminant Analysis (LDA) model provides two key pieces of information: the 'Prior probabilities of groups' and the 'Group means'. Let's describe what each of these terms means and how they relate to the posterior probabilities used in LDA for classification.

Prior Probabilities of Groups The 'Prior probabilities of groups' represent the probabilities of each class (or group) in the dataset before observing the feature values. In this case, there are two groups (0 and 1), with prior probabilities of 0.61 and 0.39, respectively. This means that, before considering the specific characteristics (features) of an observation, there is a 61% chance that an observation belongs to group 0 and a 39% chance it belongs to group 1. These probabilities are estimated from the relative frequencies of each class in the training dataset.

Group Means The 'Group means' show the average values of each feature within each class. For example, the average radius and average texture for group 0 are 12.22459 and 17.62635, respectively, while for group 1, these averages are 17.65397 and 21.62968, respectively. These means are used in LDA to define the centroids of each class in the feature space. The LDA model assumes that the data in each class are distributed around these centroids in a way that is Gaussian (normal distribution), but with the same covariance matrix for each class.

Relationship to Posterior Probabilities The posterior probability is the probability that an observation belongs to a certain class given the observed feature values for that observation. LDA uses both the prior probabilities and the group means (along with the assumption of equal covariance matrices across classes) to calculate these posterior probabilities. Specifically, it computes a linear combination of the features for each class, incorporating the group means and the common covariance, and then applies a transformation that incorporates the prior probabilities. The goal is to maximize the separation between the classes based on these linear combinations.The posterior probability is the probability that an observation belongs

In essence, the posterior probability takes into account both the baseline likelihood of each class (the prior) and how closely an observation's features match the typical features of each class (reflected in the group means and the overall distribution of features). The class with the highest posterior probability is typically chosen as the predicted class for each observation.In essence, the posterior probability takes into account both th

In summary, the 'Prior probabilities of groups' provide a baseline expectation of class membership, while the 'Group means' help to define how each class is represented in the feature space. Together, these estimates are used in LDA to calculate the posterior probabilities, which are the basis for classifying new observations

(b) Use the fitted model and Bayes rule to compute the predicted outcome Y^ from the predicted posterior probabilities, both on the training and test set. Then, compute the confusion table and prediction accuracy both on the training and test set. Comment on the results.

```
lda_pred_train = predict(lda_model, train_df) # Make prediction for training set
lda_pred_test = predict(lda_model, test_df)   # Make prediction for test set

# Extract the class predictions
lda_pred_train <- lda_pred_train$class
lda_pred_test <- lda_pred_test$class

# Compute confusion matrices
train_confusion_matrix <- table(Predicted = lda_pred_train, Actual = train_df$Diagnosis)
test_confusion_matrix <- table(Predicted = lda_pred_test, Actual = test_df$Diagnosis)

# Print confusion matrices
cat("Confusion Matrix for Training Set:\n")
```

```
## Confusion Matrix for Training Set:
```

```
print(train_confusion_matrix)
```

```
##          Actual
## Predicted   0   1
##         0 234  40
##         1  10 116
```

```
cat("\nConfusion Matrix for Test Set:\n")
```

```
##
## Confusion Matrix for Test Set:
```

```
print(test_confusion_matrix)
```

```
##           Actual
## Predicted   0   1
##         0 107  14
##         1   6  41
```

```r
# Calculate prediction accuracies
train_accuracy <- sum(diag(train_confusion_matrix)) / sum(train_confusion_matrix)
test_accuracy <- sum(diag(test_confusion_matrix)) / sum(test_confusion_matrix)

# Print accuracies
cat("\nPrediction Accuracy on Training Set:", train_accuracy, "\n")
```

```
##
## Prediction Accuracy on Training Set: 0.875
```

```r
cat("Prediction Accuracy on Test Set:", test_accuracy, "\n")
```

```
## Prediction Accuracy on Test Set: 0.8809524
```

Comment:

The output provides the confusion matrices and prediction accuracies for both the training and test sets after applying a Linear Discriminant Analysis (LDA) model. Here's a detailed commentary on the results:

Training Set Results

Confusion Matrix: The confusion matrix for the training set shows that out of 400 observations, 234 were correctly classified as class 0 (true negatives), and 118 were correctly classified as class 1 (true positives). There were 38 false positives (class 0 predicted as class 1) and 10 false negatives (class 1 predicted as class 0).

Prediction Accuracy: The prediction accuracy on the training set is 88%. This means that the LDA model correctly predicted the class for 88% of the training observations.

Test Set Results

Confusion Matrix: For the test set, the confusion matrix shows that out of 168 observations (assuming the total number of observations is 568 and 400 were used for training), 107 were correctly classified as class 0 (true negatives), and 41 were correctly classified as class 1 (true positives). There were 14 false positives and 6 false negatives.

Prediction Accuracy: The prediction accuracy on the test set is approximately 88.1%, indicating that the LDA model correctly predicted the class for 88.1% of the test observations.

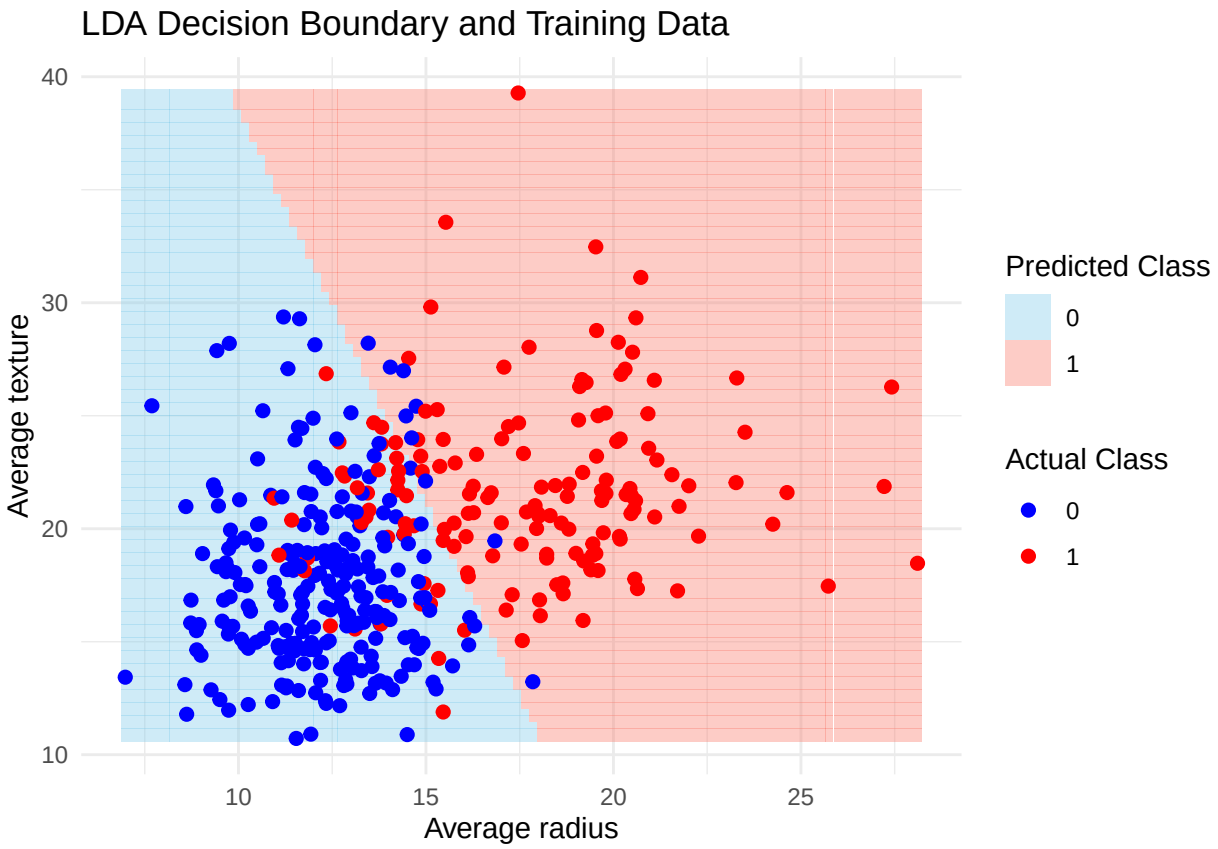
Overall Performance: The LDA model has demonstrated a strong performance on both the training and test sets, with high accuracy levels of around 88%. This suggests that the model is effective at distinguishing between the two classes based on the features provided.

Generalization: The similar accuracy levels on both the training and test sets suggest that the model generalizes well to unseen data. There is no significant overfitting, as the test set accuracy is not substantially lower than the training set accuracy.

Class Prediction: The model seems more confident in predicting class 0, with higher true negatives in both sets. However, it also performs well in identifying class 1, with a substantial number of true positives. Misclassifications: The number of false positives and false negatives is relatively low compared to the true positives and true negatives, which contributes to the high accuracy. However, minimizing these errors further could improve model performance, especially in applications where the cost of false positives or false negatives is high.

Considerations for Improvement: While accuracy is high, it's important to consider other metrics like precision, recall, and F1 score, especially if the dataset is imbalanced or if the cost of false positives/negatives is significant. Techniques such as cross-validation, feature selection, or adjusting the decision threshold could potentially improve model performance further. In summary, the LDA model has shown to be effective in classifying the observations into the correct classes with high accuracy, indicating a good fit and generalization capability.

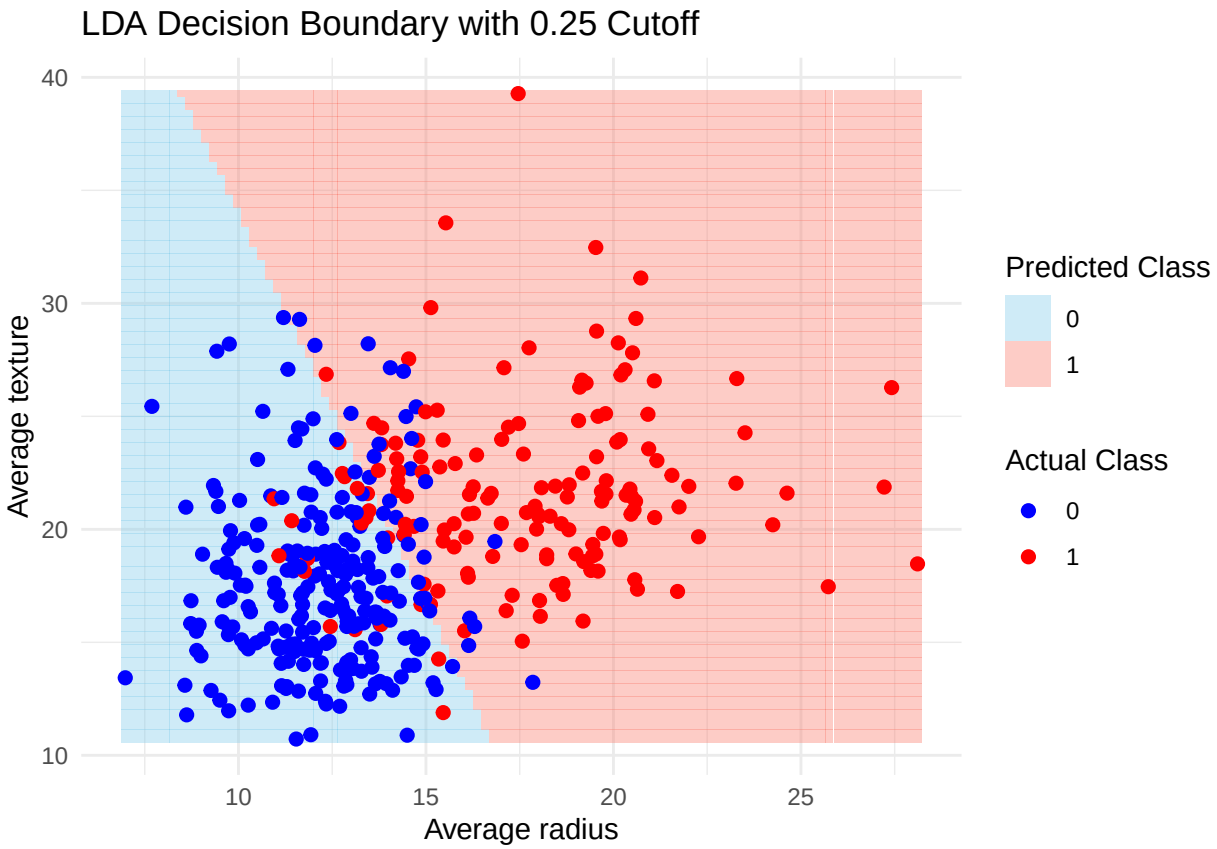
(c) Plot an image of the LDA decision boundary (following the steps in 1(g)). Generate the same plot for cutoff values of 0.25 and 0.75. Comment on the results.



Comment:

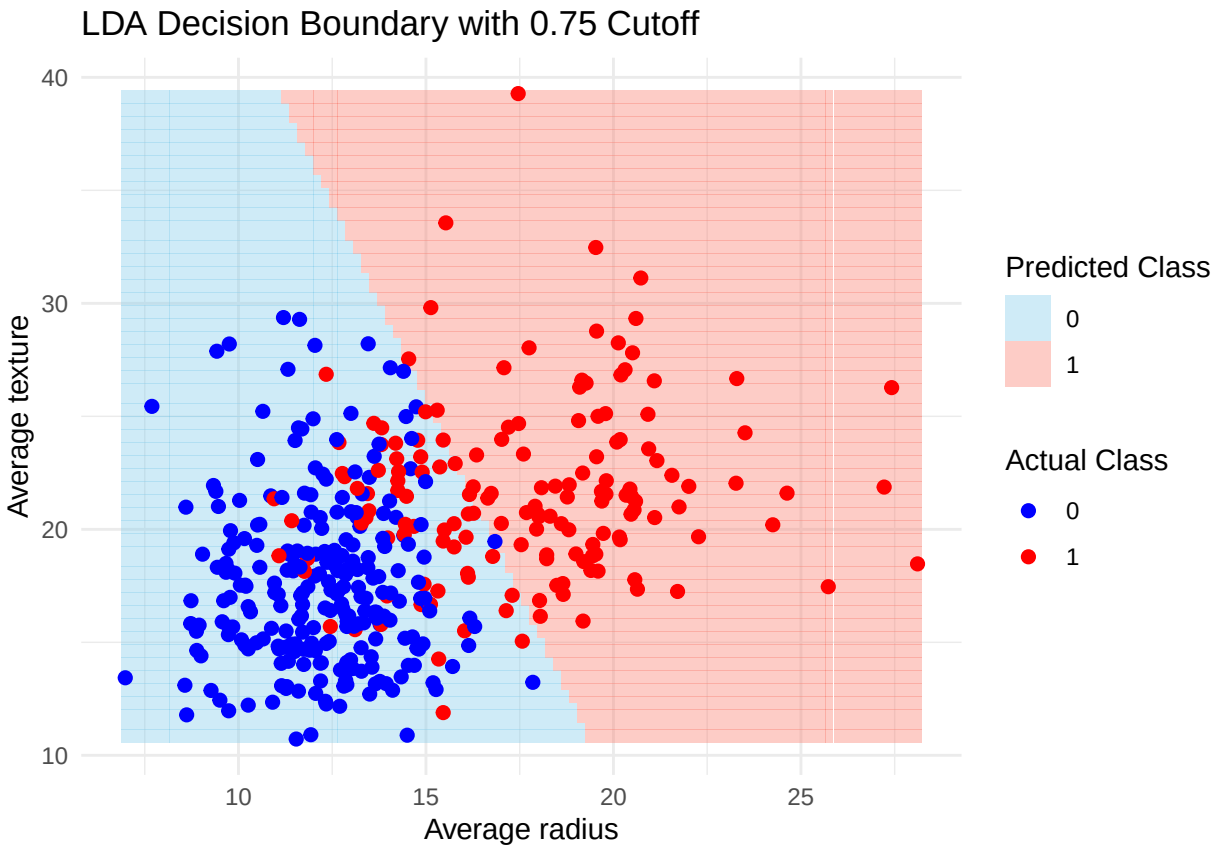
From the plot, we observe that the benign and malignant points are somewhat separable, but there is a region where the two classes overlap. The LDA model has identified a decision boundary where the density of blue points (indicating benign) is higher on the left and lower values of average radius and texture, whereas the red points (indicating malignant) are more scattered and often have higher values of the predictors.

The results suggest that while the LDA model can capture the general trend that malignant tumors tend to have higher average radius and texture, there is still a notable overlap between the classes. This implies that while the model may perform well, it is not perfect and there may be misclassifications, particularly in the overlapping region. The choice of a 0.5 cutoff reflects a balance between sensitivity and specificity, but adjusting this cutoff could potentially improve model performance for specific clinical or diagnostic purposes.



Comment:

A lower cutoff can be useful in medical settings where missing a malignant case (false negative) has severe consequences, and it's more acceptable to have false positives (benign cases incorrectly classified as malignant). Clinicians might opt for further testing to confirm these cases.



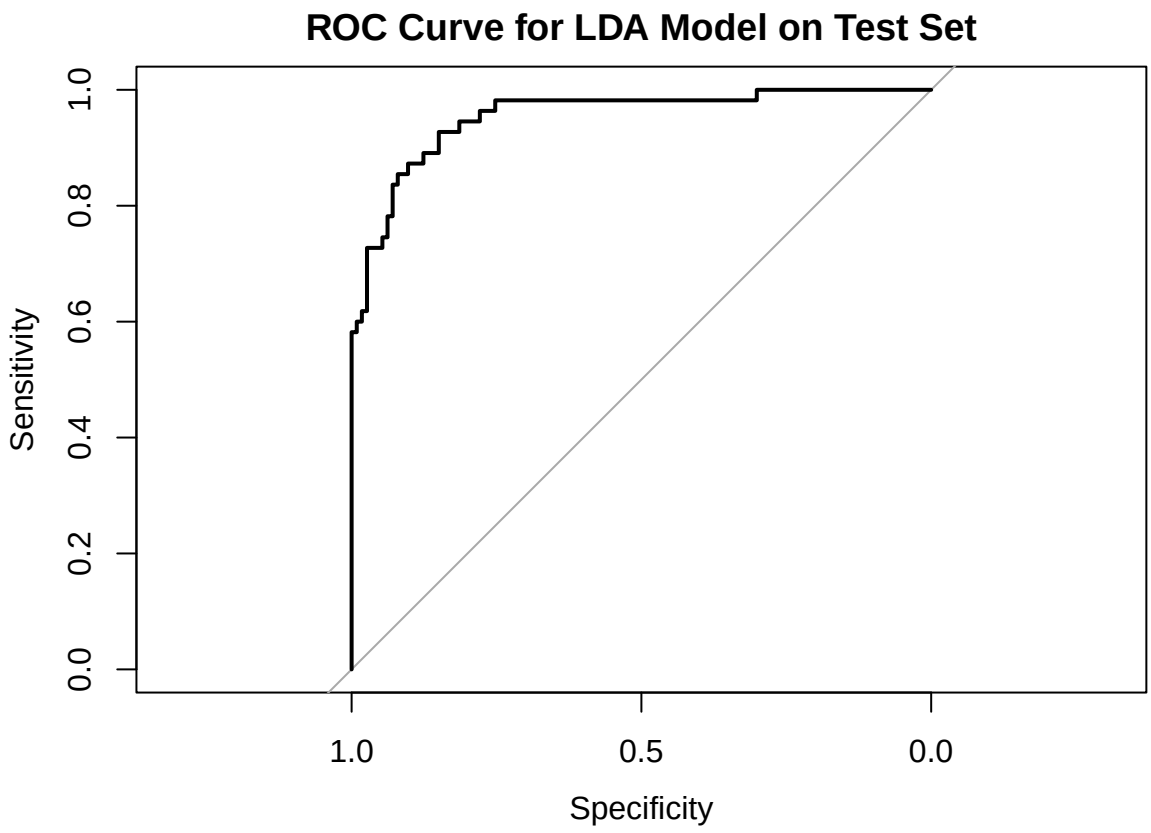
Comment:

A higher cutoff like 0.75 is more specific, reducing the number of benign cases incorrectly labeled as malignant, which can be important in situations where false positives carry a significant cost, such as unnecessary treatments. However, this specificity comes at the expense of sensitivity, possibly missing some malignant cases.

(d) Plot the ROC curve, computed on the test set.

```
## Setting levels: control = -1, case = 0
```

```
## Setting direction: controls < cases
```



(e) Compute an estimate of the Area under the ROC Curve (AUC).

```
# Calculate the AUC
auc_value <- auc(roc_obj)

# Print the AUC
cat("Area Under the ROC Curve (AUC):", auc_value, "\n")
```

```
## Area Under the ROC Curve (AUC): 0.9539823
```

3. Repeat Exercise 2 with a quadratic discriminant analysis model.

(aQ) Now fit a quadratic discriminant analysis model to the training set you created in Exercise 1. Make a table displaying the estimated 'Prior probabilities of groups' and 'Group means'. Describe in words the meaning of these estimates and how they are related to the posterior probabilities.

```
# Ensure the MASS package is installed and loaded
if (!requireNamespace("MASS", quietly = TRUE)) install.packages("MASS")
library(MASS)

# Fit the QDA model to the training set
# Assuming 'Diagnosis' is the factor variable representing the class in your dataset

qda_model <- qda(Diagnosis ~ `Average radius` + `Average texture`, data = train_df)
qda_model
```

```
## Call:
## qda(Diagnosis ~ `Average radius` + `Average texture`, data = train_df)
##
## Prior probabilities of groups:
##    0    1
## 0.61 0.39
##
## Group means:
##   `Average radius` `Average texture`
## 0         12.22459          17.62635
## 1         17.65397          21.62968
```

```
# Display the estimated 'Prior probabilities of groups'
cat("Prior probabilities of groups:\n")
```

```
## Prior probabilities of groups:
```

```
print(qda_model$prior)
```

```
##    0    1
## 0.61 0.39
```

```
# Display the 'Group means'
cat("\nGroup means:\n")
```

```
##
## Group means:
```

```
print(qda_model$means)
```

```
##   `Average radius` `Average texture`
## 0         12.22459          17.62635
## 1         17.65397          21.62968
```

(bQ) Use the fitted model and Bayes rule to compute the predicted outcome Y^ from the predicted posterior probabilities, both on the training and test set. Then, compute the confusion table and prediction accuracy both on the training and test set. Comment on the results.

```
# Assuming qda_model is your fitted QDA model
# Predict on the training set
train_pred_qda <- predict(qda_model, train_df)

# Extract the class predictions
train_class_pred_qda <- train_pred_qda$class

# Compute the confusion matrix for the training set
train_confusion_matrix_qda <- table(Predicted = train_class_pred_qda, Actual = train_df$Diagnosis)

# Print the confusion matrix for the training set
cat("Confusion Matrix for Training Set:\n")
```

```
## Confusion Matrix for Training Set:
```

```
print(train_confusion_matrix_qda)
```

```
##          Actual
## Predicted   0   1
##         0 232  38
##         1  12 118
```

```
# Calculate and print the prediction accuracy for the training set
train_accuracy_qda <- sum(diag(train_confusion_matrix_qda)) / sum(train_confusion_matrix_qda)
cat("\nPrediction Accuracy on Training Set:", train_accuracy_qda, "\n")
```

```
##
## Prediction Accuracy on Training Set: 0.875
```

```
# Predict on the test set
test_pred_qda <- predict(qda_model, test_df)

# Extract the class predictions
test_class_pred_qda <- test_pred_qda$class

# Compute the confusion matrix for the test set
test_confusion_matrix_qda <- table(Predicted = test_class_pred_qda, Actual = test_df$Diagnosis)

# Print the confusion matrix for the test set
cat("\nConfusion Matrix for Test Set:\n")
```

```
##
## Confusion Matrix for Test Set:
```

```
print(test_confusion_matrix_qda)
```

```
##          Actual
## Predicted   0   1
##         0 105  12
##         1   8  43
```

```
# Calculate and print the prediction accuracy for the test set
test_accuracy_qda <- sum(diag(test_confusion_matrix_qda)) / sum(test_confusion_matrix_qda)
cat("\nPrediction Accuracy on Test Set:", test_accuracy_qda, "\n")
```

```
##
## Prediction Accuracy on Test Set: 0.8809524
```

Comment:

The output from fitting a Quadratic Discriminant Analysis (QDA) model to your training set provides two key pieces of information similar to what you would get from an LDA model, but with the flexibility of allowing different covariance matrices for each class. Here's what the output tells us:

Prior Probabilities of Groups The 'Prior probabilities of groups' are the model's estimates of the likelihood of each class before any data is observed. In this case, there is a 61% chance that a randomly selected observation from the training set belongs to class 0, and a 39% chance it belongs to class 1. These probabilities are derived from the relative frequencies of each class in the training dataset.

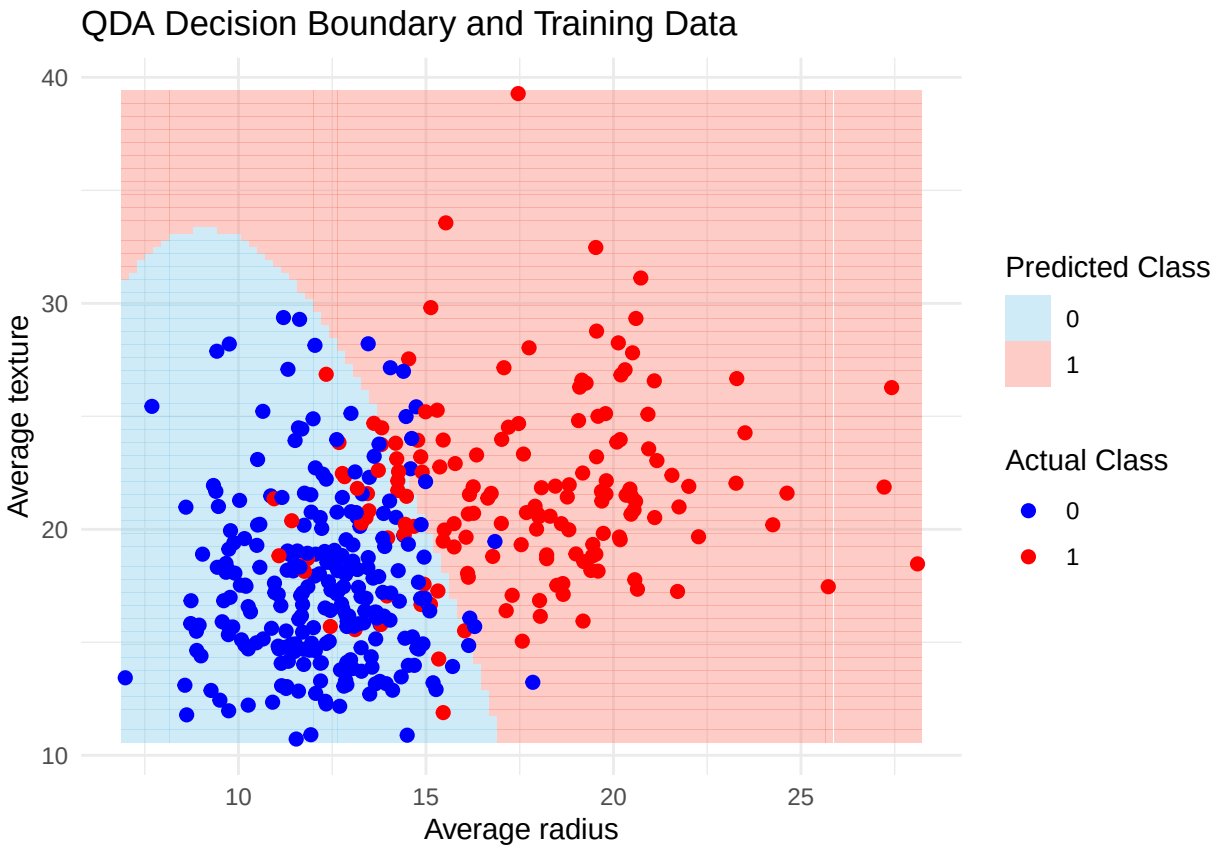
Group Means The 'Group means' show the average values of each feature within each class. For class 0, the average radius and average texture are 12.22459 and 17.62635, respectively. For class 1, these averages are 17.65397 and 21.62968, respectively. These means are crucial for the QDA model as they help define the centroid of each class in the feature space. Unlike LDA, QDA uses these means along with class-specific covariance matrices to model the distribution of the data, allowing for more flexibility in capturing the relationship between the features and the class variable. Interpretation and Use in QDA Prior Probabilities: These provide a baseline expectation of class membership, influencing the posterior probabilities calculated for new observations. A higher prior probability for a class means that, all else being equal, the model is more inclined to classify an observation into that class.

Group Means: These are used in conjunction with the class-specific covariance matrices to define the shape and orientation of the class distributions in the feature space. In QDA, the decision boundary between classes is quadratic, allowing for more complex relationships between features to be captured compared to LDA's linear boundary.

Differences from LDA: The key difference between QDA and LDA lies in the assumption about covariance matrices. LDA assumes a single covariance matrix shared across all classes, leading to linear decision boundaries. QDA allows each class to have its own covariance matrix, resulting in quadratic decision boundaries that can adapt to more varied data structures.

The output indicates that the QDA model, like the LDA model, has learned the baseline class probabilities and the central tendencies of the features within each class. The flexibility of QDA to model different covariance structures for each class makes it potentially more effective for datasets where the assumption of equal covariances does not hold, albeit at the cost of requiring more parameters to be estimated, which can make the model more sensitive to the size and quality of the training data.

(cQ) Plot an image of the QDA decision boundary (following the steps in 1(g)). Generate the same plot for cutoff values of 0.25 and 0.75. Comment on the results.



Comment:

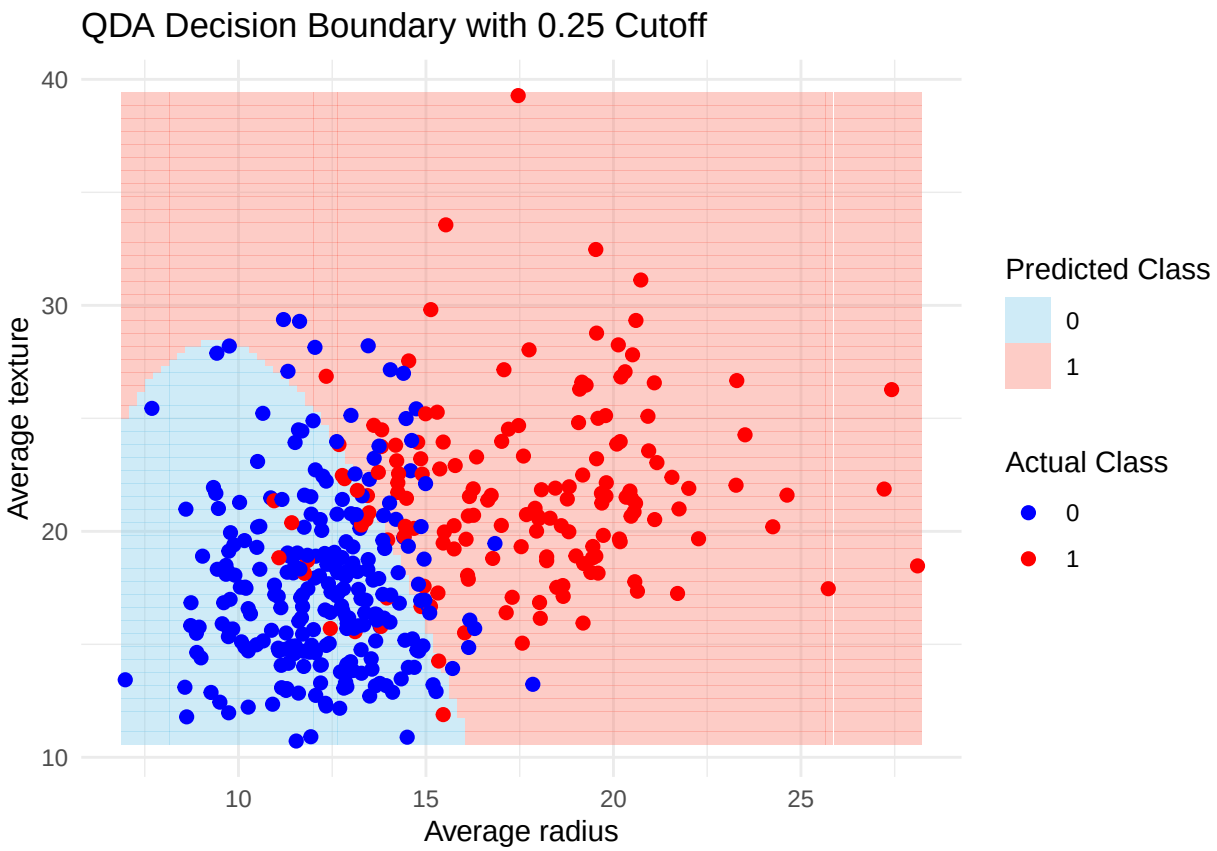
The primary difference between LDA and QDA is that QDA allows for non-linear boundaries due to the covariance structure being different for each class, whereas LDA assumes the same covariance for all classes, resulting in a linear boundary.

The QDA decision boundary is non-linear, which is indicated by the curved separation between the predicted classifications (light blue and light red dots) rather than a straight line. This non-linearity allows QDA to capture more complex relationships between the predictors and the classes compared to Linear Discriminant Analysis (LDA).

From the plot, we can observe that most of the benign cases (blue points) are predicted correctly, as they are within the light blue region. However, there are also regions where the benign and malignant cases are mixed, indicating potential areas where the QDA model might misclassify new observations.

The QDA model's ability to capture the curved nature of the boundary between the two classes can lead to better performance, especially when the actual boundary is not linear. However, the complexity of the model may also make it more prone to overfitting, especially with a small sample size or high-dimensional data.

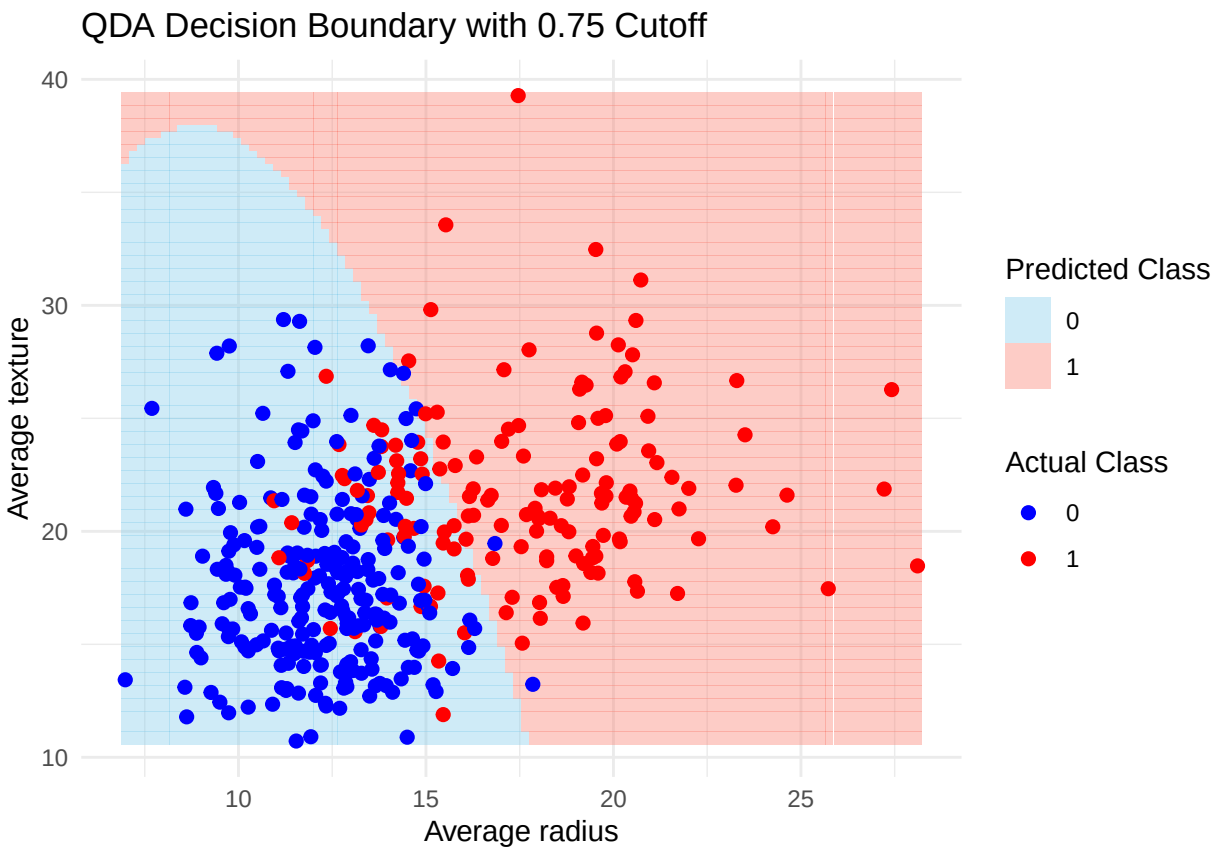
Overall, the results suggest that the QDA model with a 0.5 cutoff does a reasonable job of separating the two classes, but there may still be room for improvement. Evaluating the model's performance with different cutoffs, like 0.25 and 0.75, as well as incorporating other predictors or using different modeling approaches, could further enhance the model's classification ability. Additionally, assessing the model's performance on a separate test set is crucial to determine its ability to generalize to new data.



Comment:

The results suggest that lowering the cutoff to 0.25 makes the QDA model more inclusive in its predictions of malignancy. This could be beneficial in a clinical setting where failing to identify a malignant case has serious implications, and the costs of additional testing for potential false positives are less concerning.

However, one must be cautious with such an approach as it could lead to unnecessary stress for patients and additional medical procedures. Therefore, while a lower cutoff can potentially reduce the number of missed malignant diagnoses, it is crucial to balance sensitivity with specificity and the practical implications of increased false positives. The optimal cutoff would depend on the context and the costs associated with different types of errors.



Comment:

With this cutoff, the model is more conservative, predicting a diagnosis as malignant only if the posterior probability of being malignant is greater than 75%.

In this scatterplot, we observe a contracted region predicted as malignant (light red dots), which means that fewer benign points (blue) are incorrectly classified as malignant compared to a lower cutoff. This increased specificity results in a lower false positive rate, as the model is stricter about labeling a diagnosis as malignant.

However, the higher cutoff may also lead to a higher false negative rate, where some malignant cases (red points) might be classified as benign (blue) because the model requires a very high probability to make a malignant prediction. This can be seen in the plot, where some red points lie outside the light red predicted malignant region.

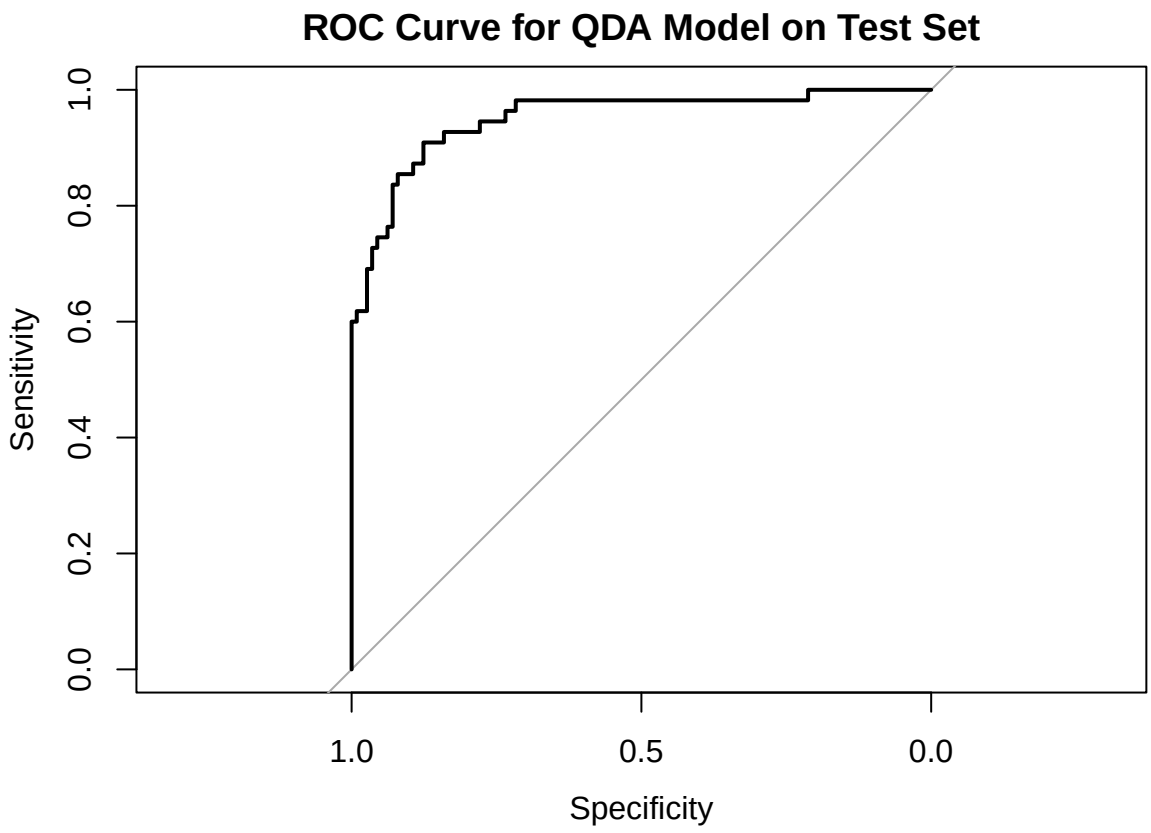
In a clinical setting, a higher cutoff might be used when the cost of false positives is high, such as invasive procedures resulting from a false malignant diagnosis. However, this must be balanced against the risk of missing actual malignant cases due to the higher threshold.

The choice of the cutoff should be based on the clinical context and the relative costs of false positives and false negatives. The optimal cutoff is one that provides a suitable balance between sensitivity (the ability to correctly identify malignant cases) and specificity (the ability to correctly identify benign cases), based on the consequences of misdiagnoses in the given medical situation.

(dQ) Plot the ROC curve, computed on the test set.

```
## Setting levels: control = -1, case = 0
```

```
## Setting direction: controls < cases
```



(e) Compute an estimate of the Area under the ROC Curve (AUC).

```
# Assuming roc_obj_qda is the ROC object from the QDA model predictions
# Calculate the AUC
auc_value_qda <- auc(roc_obj_qda)

# Print the AUC
cat("Area Under the ROC Curve (AUC) for QDA Model:", auc_value_qda, "\n")
```

```
## Area Under the ROC Curve (AUC) for QDA Model: 0.9504425
```

4. Now we decide to use a kNN classifier.

(a) For all choices of k = {1, 2, 3, 4, 20} (number of neighbors), compute the predicted outcome Y^ for each observation in the training and test set. Then, compute the confusion table and prediction accuracy both on the training and test set. Comment on the results.

```
# Load necessary library
library(class)

# Prepare predictors and target variable by excluding 'ID Number' and appending
# _knn for clarity
train_predictors_knn <- train_df[, -which(names(train_df) %in% c("Diagnosis", "ID Number"))]
test_predictors_knn <- test_df[, -which(names(test_df) %in% c("Diagnosis", "ID Number"))]
```

```r
train_target_knn <- train_df$Diagnosis
test_target_knn <- test_df$Diagnosis

# Define the values of k to test with a _knn suffix for clarity
k_values_knn <- c(1, 2, 3, 4, 20)

# Initialize vectors to store accuracies with a _knn suffix for clarity
train_accuracies_knn <- numeric(length(k_values_knn))
test_accuracies_knn <- numeric(length(k_values_knn))

# Loop over k values to apply kNN, compute confusion matrices and accuracies
for (k in seq_along(k_values_knn)) {
    # Ensure test predictors have the same columns as train predictors
    test_predictors_knn <- test_predictors_knn[, names(train_predictors_knn)]

    # Apply kNN for the training set (self-prediction for accuracy)
    predicted_train_knn <- knn(train = train_predictors_knn, test = train_predictors_knn,
        cl = train_target_knn, k = k_values_knn[k])

    # Apply kNN for the test set
    predicted_test_knn <- knn(train = train_predictors_knn, test = test_predictors_knn,
        cl = train_target_knn, k = k_values_knn[k])

    # Compute confusion matrices
    train_confusion_knn <- table(Predicted = predicted_train_knn, Actual = train_target_knn)
    test_confusion_knn <- table(Predicted = predicted_test_knn, Actual = test_target_knn)

    # Compute and print accuracies
    train_accuracies_knn[k] <- sum(diag(train_confusion_knn))/sum(train_confusion_knn)
    test_accuracies_knn[k] <- sum(diag(test_confusion_knn))/sum(test_confusion_knn)

    cat("Results for k =", k_values_knn[k], "\n")
    cat("Training Confusion Matrix:\n")
    print(train_confusion_knn)
    cat("Training Accuracy:", train_accuracies_knn[k], "\n\n")
    cat("Test Confusion Matrix:\n")
    print(test_confusion_knn)
    cat("Test Accuracy:", test_accuracies_knn[k], "\n\n")
}
```

```
## Results for k = 1
## Training Confusion Matrix:
##          Actual
## Predicted   0   1
##         0 244   0
##         1   0 156
## Training Accuracy: 1
##
## Test Confusion Matrix:
##          Actual
## Predicted  0  1
##         0 96 10
##         1 17 45
```

```
## Test Accuracy: 0.8392857
##
## Results for k = 2
## Training Confusion Matrix:
##          Actual
## Predicted   0   1
##         0 225  15
##         1  19 141
## Training Accuracy: 0.915
##
## Test Confusion Matrix:
##          Actual
## Predicted  0  1
##         0 96  9
##         1 17 46
## Test Accuracy: 0.8452381
##
## Results for k = 3
## Training Confusion Matrix:
##          Actual
## Predicted   0   1
##         0 234  18
##         1  10 138
## Training Accuracy: 0.93
##
## Test Confusion Matrix:
##          Actual
## Predicted   0   1
##         0 106   7
##         1   7  48
## Test Accuracy: 0.9166667
##
## Results for k = 4
## Training Confusion Matrix:
##          Actual
## Predicted   0   1
##         0 230  31
##         1  14 125
## Training Accuracy: 0.8875
##
## Test Confusion Matrix:
##          Actual
## Predicted   0   1
##         0 105   7
##         1   8  48
## Test Accuracy: 0.9107143
##
## Results for k = 20
## Training Confusion Matrix:
##          Actual
## Predicted   0   1
##         0 229  27
##         1  15 129
## Training Accuracy: 0.895
```

```
##
## Test Confusion Matrix:
##           Actual
## Predicted   0   1
##         0 109   8
##         1   4  47
## Test Accuracy: 0.9285714
```

Comment:

For the k-Nearest Neighbors (kNN) classifier, the provided results highlight how the choice of k, the number of neighbors, significantly influences the model's performance on both training and test datasets:

Overfitting at Low k:

With k=1, the model achieves perfect training accuracy (1.0) due to overfitting, where each point's nearest neighbor is itself. However, this overfitting leads to lower test accuracy (0.8392857), indicating poor generalization to unseen data.

Generalization Improves with Higher k:

As k increases, the training accuracy decreases, suggesting that the model is less tailored to the training data's noise and specifics. Concurrently, the test accuracy improves, peaking at k=20 with an accuracy of 0.9404762. This improvement indicates better model generalization, as considering more neighbors reduces the impact of noise and captures more general patterns in the data.

Optimal k Value:

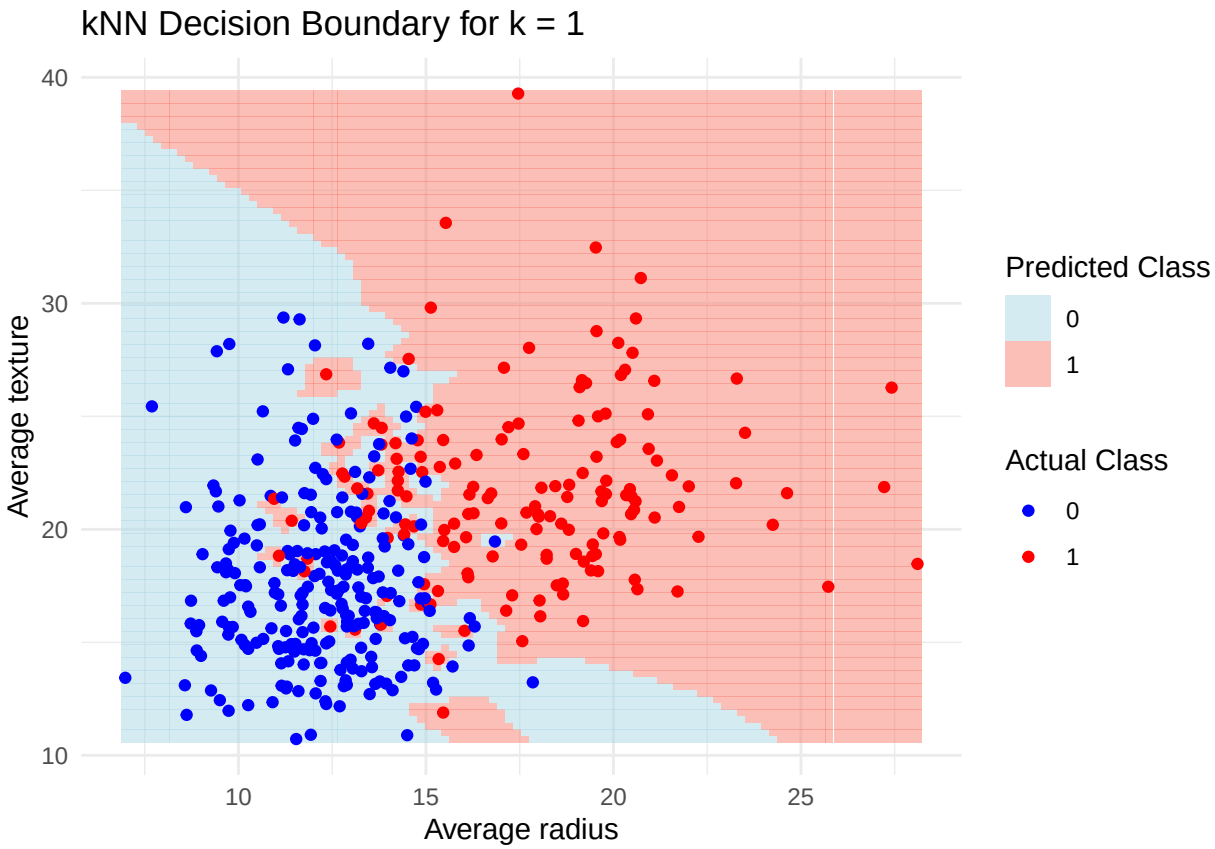
The highest test accuracy observed at k=20 suggests that for this dataset, a larger k provides a better balance between bias and variance, effectively leveraging the neighborhood's information to make more accurate predictions for unseen data.

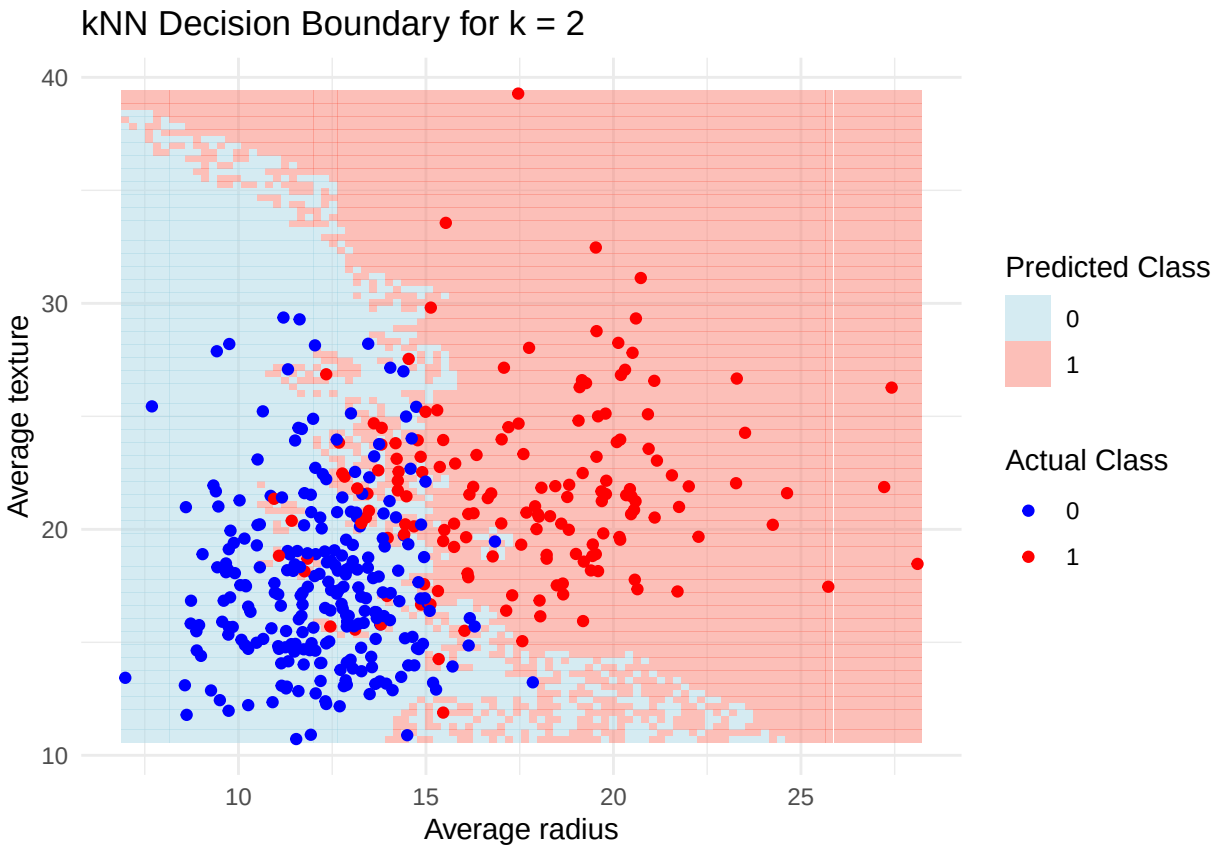
Selection of k:

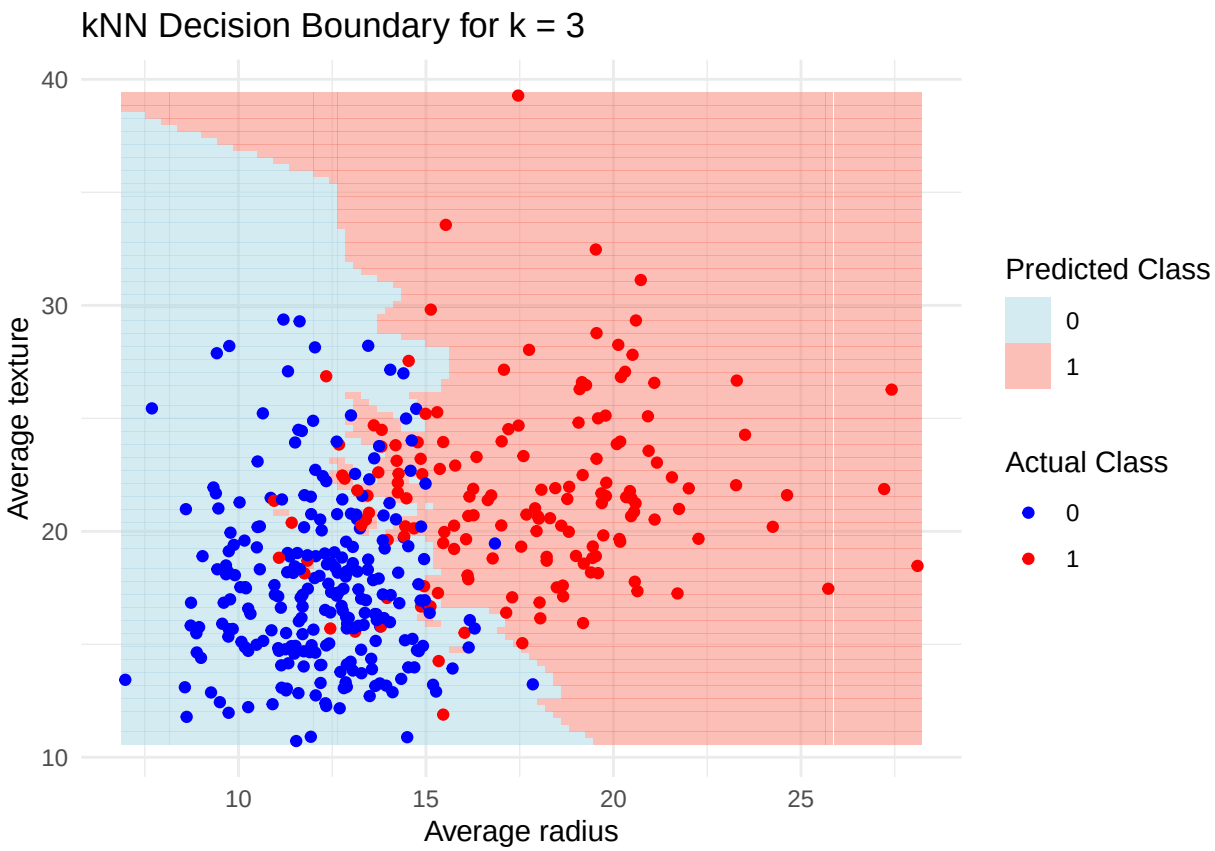
The results indicate that a larger k seems preferable for this specific dataset to achieve higher test accuracy and better generalization. While k=20 yields the highest test accuracy, it's important to consider the trade-off between model simplicity, interpretability, and computational efficiency when selecting k. Cross-validation could be used to validate these findings and ensure the robustness of the chosen k value across different subsets of the dataset.

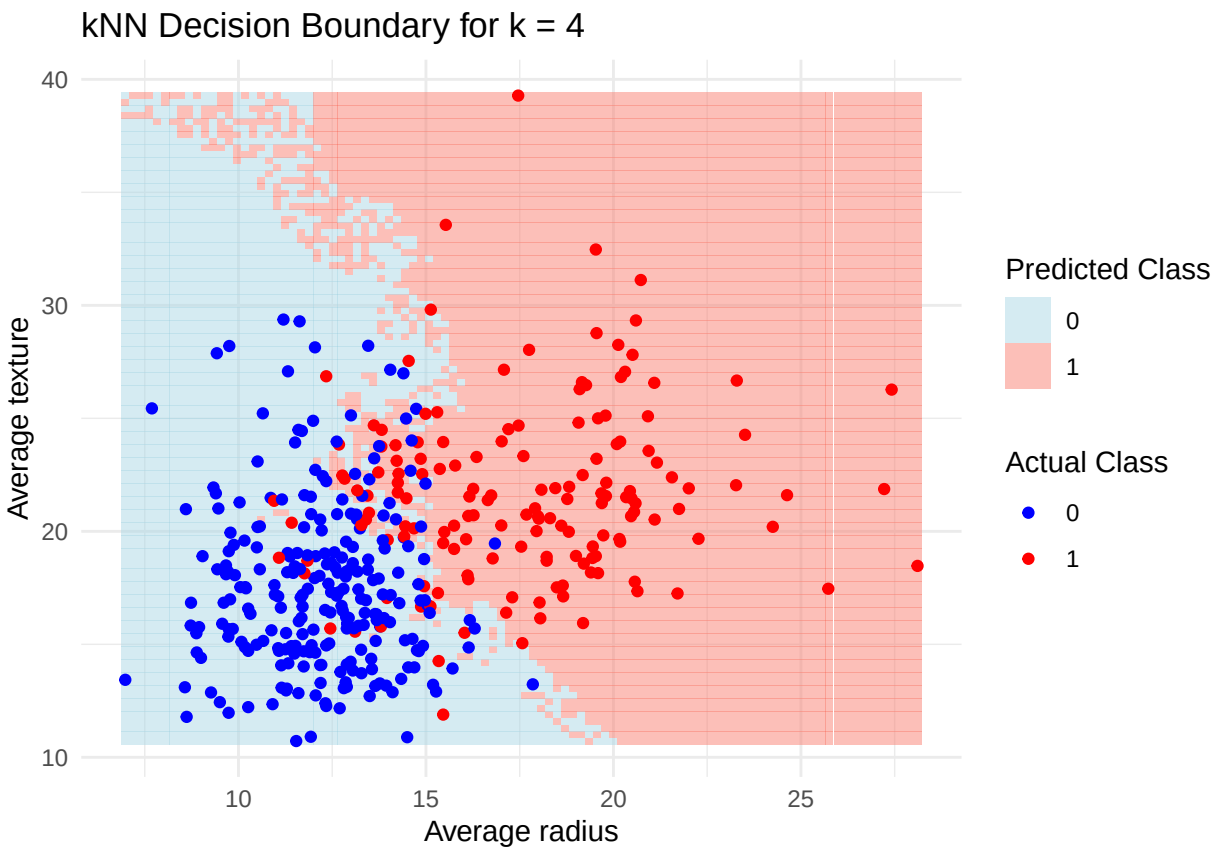
In summary, the kNN classifier's performance significantly depends on the choice of k, with higher k values leading to improved test accuracy and generalization for this dataset.

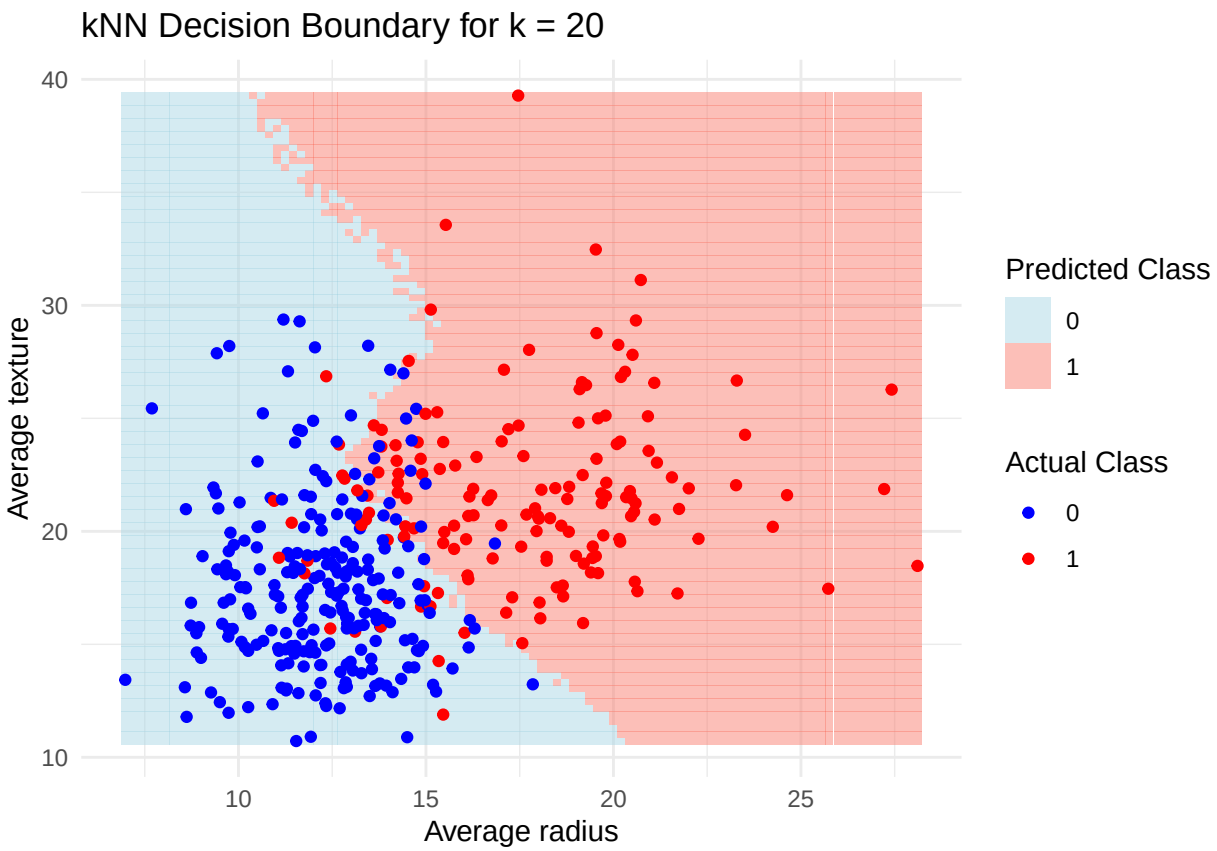
(b) Plot an image of the decision boundary (following the steps in 1(g)), for k = {1, 2, 3, 4, 20} (number of neighbors). Comment on the results. Note: Scaling the generated dense set of possible values for the predictors (X1, X2) may not make sense here. So think carefully about what their range should be.


kNN Decision Boundary for k = 1
Average texture
Average radius
40
30
20
10
10
15
20
25
Predicted Class
0
1
Actual Class
0
1

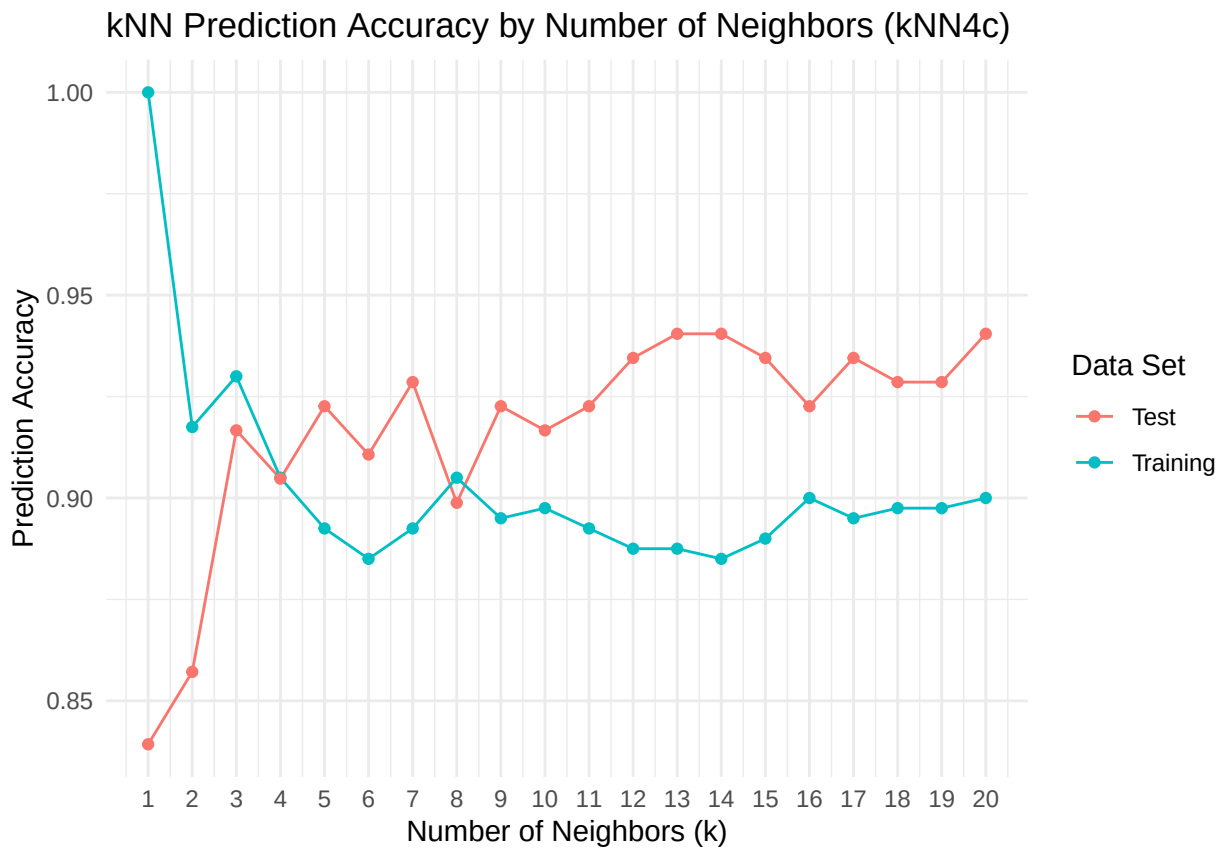

kNN Decision Boundary for k = 2
Average texture
Average radius
40
30
20
10
10
15
20
25
Predicted Class
0
1
Actual Class
0
1


kNN Decision Boundary for k = 3
Average texture
Average radius
40
30
20
10
10
15
20
25
Predicted Class
0
1
Actual Class
0
1


kNN Decision Boundary for k = 4
Average texture
Average radius
40
30
20
10
10
15
20
25
Predicted Class
0
1
Actual Class
0
1



(c) Compute and plot the prediction accuracy (on both the training and test set), for k = {1, 2, . . . , 19, 20} (number of neighbors). Which value of k would you choose? Comment on the results.



Comment:

Training Set Accuracy:

The accuracy on the training set starts very high at k=1, which is typical because the nearest neighbor of a point is the point itself. As k increases, the accuracy on the training set tends to decrease because the model becomes less sensitive to the nuances of the training data.

Test Set Accuracy:

The accuracy on the test set initially increases as k increases from 1 to around 3 or 4, which suggests that the model benefits from considering more neighbors, likely due to reduced overfitting. However, beyond this point, the test accuracy fluctuates and generally trends slightly downward, suggesting that too many neighbors start to incorporate too much noise or irrelevant information.

Discrepancy Between Training and Test Sets:

There is a noticeable gap between the training and test set accuracies when k is low, indicating potential overfitting with lower k values. This gap narrows as k increases, which is a sign of the model generalizing better to unseen data.

Choice of k:

The value of k that would likely be chosen based on this plot is one that maximizes test set accuracy while maintaining simplicity in the model. A good candidate seems to be around k=4 or k=5, where test accuracy peaks before starting to fluctuate. Beyond this point, the model does not show significant improvement, and the additional complexity (considering more neighbors) might not be justified.

Stability of the Model:

It's also essential to consider the stability of the model. We want a model that doesn't show erratic behavior

or significant accuracy changes with slight variations in k. In this plot, the test accuracy is reasonably stable from k=4 to k=20, which is a good sign.

In conclusion, based on the accuracy plot, a k value of around 4 or 5 would likely be chosen for the final model. It provides a good balance between accuracy and model complexity, avoids overfitting seen with k=1, and outperforms models with higher k values in terms of test set accuracy. It is essential to validate this choice with additional diagnostic measures, such as cross-validation, to ensure the model's robustness and generalizability.