STAT462_Ass3

Chris Chang, Nicole(Wenjuan) Wang, Tom Waldin

2025-10-01

Part A

Classifying wine samples (classification trees)

In this question we will train tree-based classification algorithms to classify samples according to the variable Class, which states the cultivar used for the wine.

Q1.

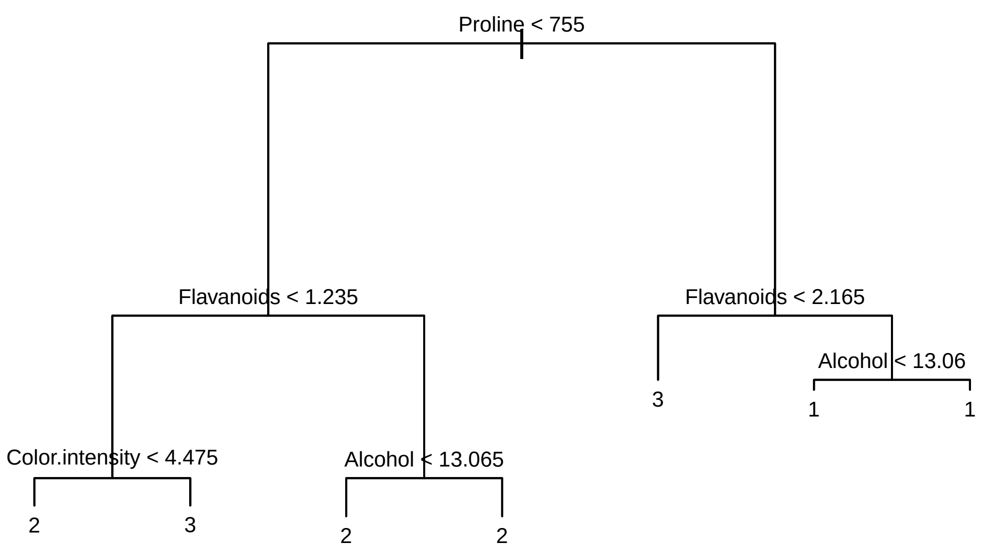
Train a single tree-based classifier on the training set. Use cross-validation to prune this tree suitably. Visualise the classification tree.

```r
# Read in data to train and test sets
train <- read.csv("Data for assignment 3-20251002\\wine_train.csv")
test <- read.csv("Data for assignment 3-20251002\\wine_test.csv")
```

```r
# Ensure Class is categorical
train$Class <- as.factor(train$Class)
test$Class <- as.factor(test$Class)

# train a tree to classify
classification.tree <- tree(Class ~ ., train)
summary(classification.tree)
```

```
##
## Classification tree:
## tree(formula = Class ~ ., data = train)
## Variables actually used in tree construction:
## [1] "Proline"         "Flavanoids"      "Color.intensity" "Alcohol"
## Number of terminal nodes:  7
## Residual mean deviance:  0.2472 = 33.37 / 135
## Misclassification error rate: 0.05634 = 8 / 142
```

```r
# Plot the intital tree
plot(classification.tree)
text(classification.tree, pretty = 0, cex = 0.7)
```

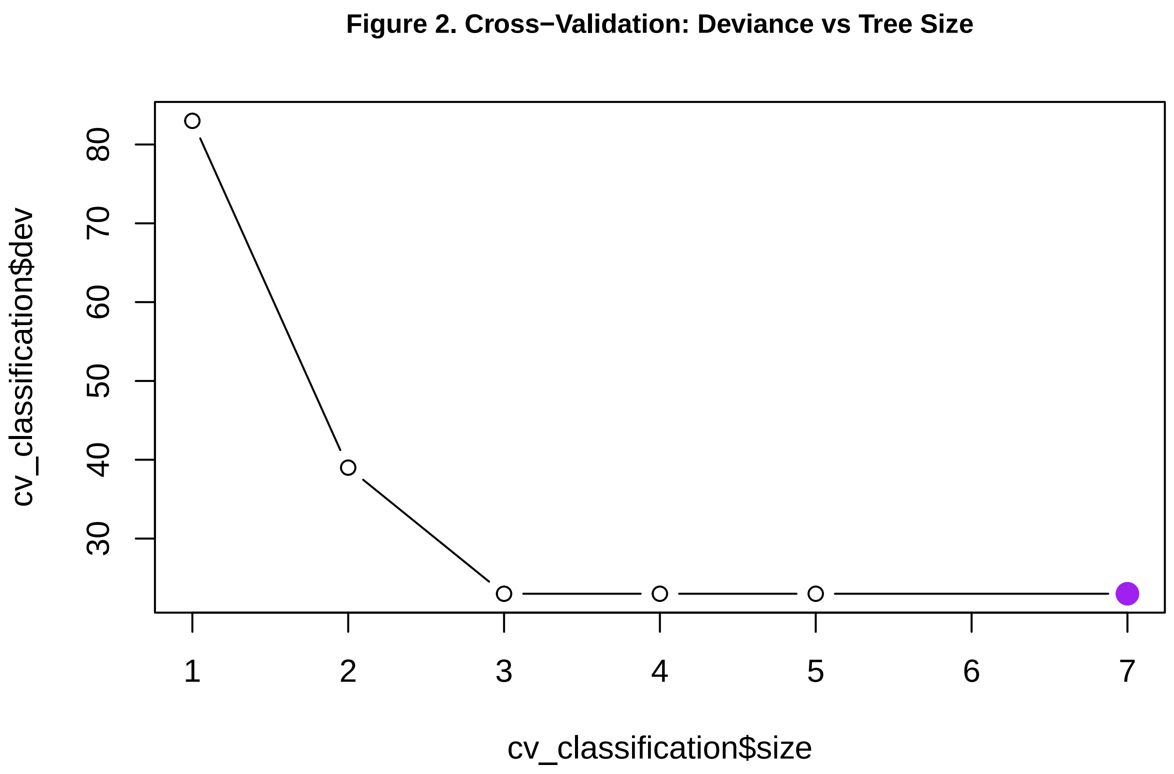


There are 7 terminal nodes in the initial classification tree, and the misclassification error rate is 5.6%.

```r
# Predict on the testing data set using the initial classification tree.
test_pred <- predict(classification.tree, newdata = test, type = "class")

# Compute the misclassification error rate on test data.
pred_error_rate <- mean(test_pred != test$Class)
cat("The predicted error rate(initial tree) on the testing data set is ",
    round(pred_error_rate * 100, 2), "%\n")
```

```
## The predicted error rate(initial tree) on the testing data set is  19.44 %
```

```r
# perform cross-validation
cv_classification <- cv.tree(classification.tree, FUN = prune.misclass)

# Plot the cross-validation results
plot(cv_classification$size, cv_classification$dev, type = "b")
title(main = "Figure 2. Cross-Validation: Deviance vs Tree Size", cex.main = 0.8)


# The optimal tree size corresponding to the lowest deviance.
best_size <- cv_classification$size[which.min(cv_classification$dev)]
best_dev  <- min(cv_classification$dev)

# Highlight the optimal tree size
points(best_size, best_dev, col = "purple", pch = 19, cex = 1.5)
```

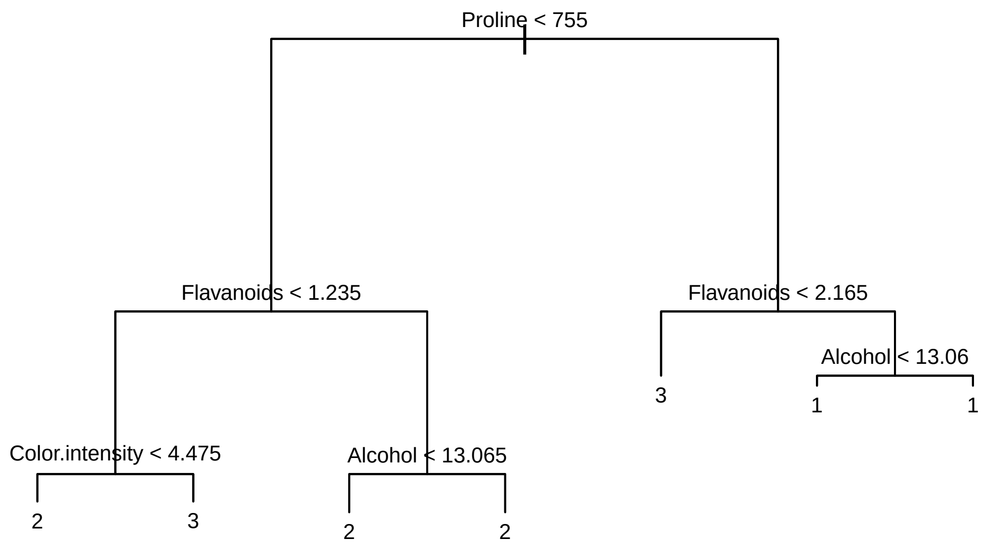


```r
cat("The optimal tree size of the pruned tree with the lowest deviance is ", best_size)
```

```
## The optimal tree size of the pruned tree with the lowest deviance is  7
```

```r
# Use the optimal tree size to prune the tree
pruned_tree <- prune.misclass(classification.tree, best = best_size)
plot(pruned_tree)
text(pruned_tree, pretty = 0, cex = 0.7)
title(main = "Figure 3. Pruned Classification Tree (After Cross-Validation)",
      cex.main = 0.8)
```

Figure 3. Pruned Classification Tree (After Cross−Validation)

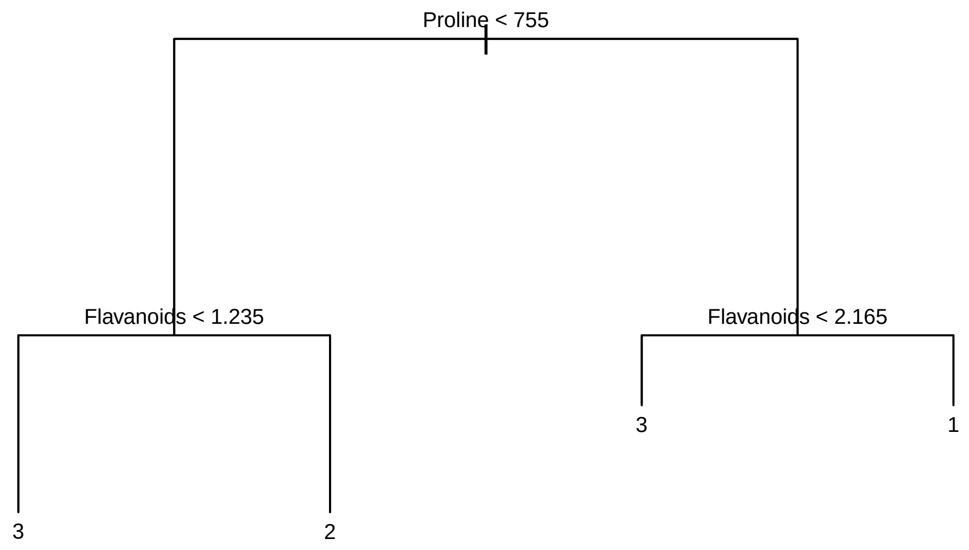


Using the training datasets, the cross-validation procedure chose the same tree size as the initial tree (7 terminal nodes). Thus, pruning did not reduce the complexity of the tree, and the predicted error rate on the testing data remained unchanged (about 19.44 %).

Q2.

Prune your tree enough so that you only need two features to make predictions. Visualise your data in these two dimensions, and illustrate the decision tree of your classifier graphically.

```r
# pruning to two branches
pruned.tree_2 <- prune.misclass(classification.tree, best = 4)

# plot pruned tree with only 2 branches
plot(pruned.tree_2)
text(pruned.tree_2, pretty = 0, cex = 0.7)
```



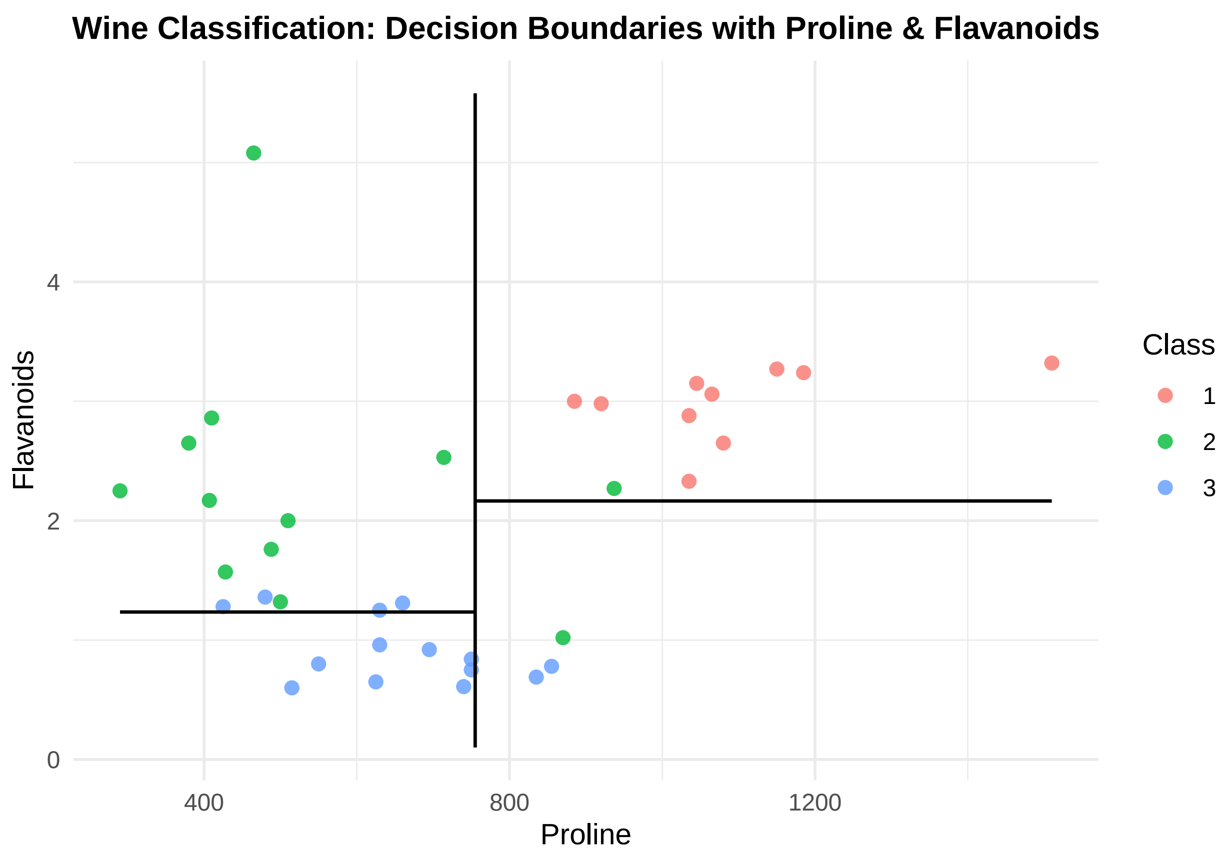
```r
summary(pruned.tree_2)
```

```
##
## Classification tree:
## snip.tree(tree = classification.tree, nodes = c(5L, 7L, 4L))
## Variables actually used in tree construction:
## [1] "Proline"    "Flavanoids"
## Number of terminal nodes:  4
## Residual mean deviance:  0.5007 = 69.09 / 138
## Misclassification error rate: 0.06338 = 9 / 142
```

Based on the classification tree shown in Figure 1, the tree should be pruned to a size of 4 in order to retain only two predictive features: Proline and Flavanoids. We therefore prune the initial tree to this size.

```r
# Visualize the pruned tree with two features

df_plot <- test[, c("Proline", "Flavanoids", "Class")]

xmin <- min(df_plot$Proline, na.rm = TRUE)
xmax <- max(df_plot$Proline, na.rm = TRUE)
ymin <- min(df_plot$Flavanoids-0.5 , na.rm = TRUE)
ymax <- max(df_plot$Flavanoids+0.5, na.rm = TRUE)

x_root <- 755
y_left <- 1.235
```

```r
y_right <- 2.165

segs <- data.frame(
  xstart = c(x_root, xmin, x_root),
  xend   = c(x_root, x_root, xmax),
  ystart = c(ymin, y_left, y_right),
  yend   = c(ymax, y_left, y_right),
  type = c("root_v", "left_h", "right_h")
)

ggplot(df_plot, aes(x = Proline, y = Flavanoids, color = Class)) +
  geom_point(size = 2, alpha = 0.8) +
  geom_segment(data = segs, aes(x = xstart, xend = xend, y = ystart, yend = yend),
               inherit.aes = FALSE, color = "black", linewidth = 0.6) +
  theme_minimal()+
  labs(title = "Wine Classification: Decision Boundaries with Proline & Flavanoids")+
  theme(plot.title = element_text(hjust = 0.5, size = 12, face = "bold"))
```



Q3.

Fit a bagged classification tree model and/or a random forest to see whether you can improve on your single tree's performance.

```r
# Count the number of predictors. Class needs to be excluded.
num_predictors <- ncol(train) -1

# Fit a bagged tree
bag.tree <- randomForest(Class ~ . , data = train, mtry = num_predictors, importance=TRUE)
bag.tree
```

```
##
## Call:
##  randomForest(formula = Class ~ ., data = train, mtry = num_predictors,      importance = TRUE)
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 13
##
##         OOB estimate of  error rate: 3.52%
## Confusion matrix:
##    1  2  3 class.error
## 1 48  1  0  0.02040816
## 2  2 55  2  0.06779661
## 3  0  0 34  0.00000000
```

```r
# bagged tree predictions on test set
bag.pred <- predict(bag.tree, newdata = test)

# compute misclassification rate (error)
bag.er <- mean(bag.pred != test$Class)
cat("The predicted error rate(bagged tree) on the testing data set is :",
    round(bag.er * 100, 2), "%\n")
```

```
## The predicted error rate(bagged tree) on the testing data set is : 0 %
```

```r
# Fit a random_forests using two features
random_forests <- randomForest(Class ~ ., data = train, mtry = 2, importance=TRUE)

# Predict on the testing set using the random_forests
random_forests_pred <- predict(random_forests, newdata = test)

# Compute the error rate on the testing data set
random_forests_error_rate <- mean(random_forests_pred != test$Class)
cat("The predicted error rate(random_forests) on the testing data set is :",
    round(random_forests_error_rate * 100, 2), "%\n")
```

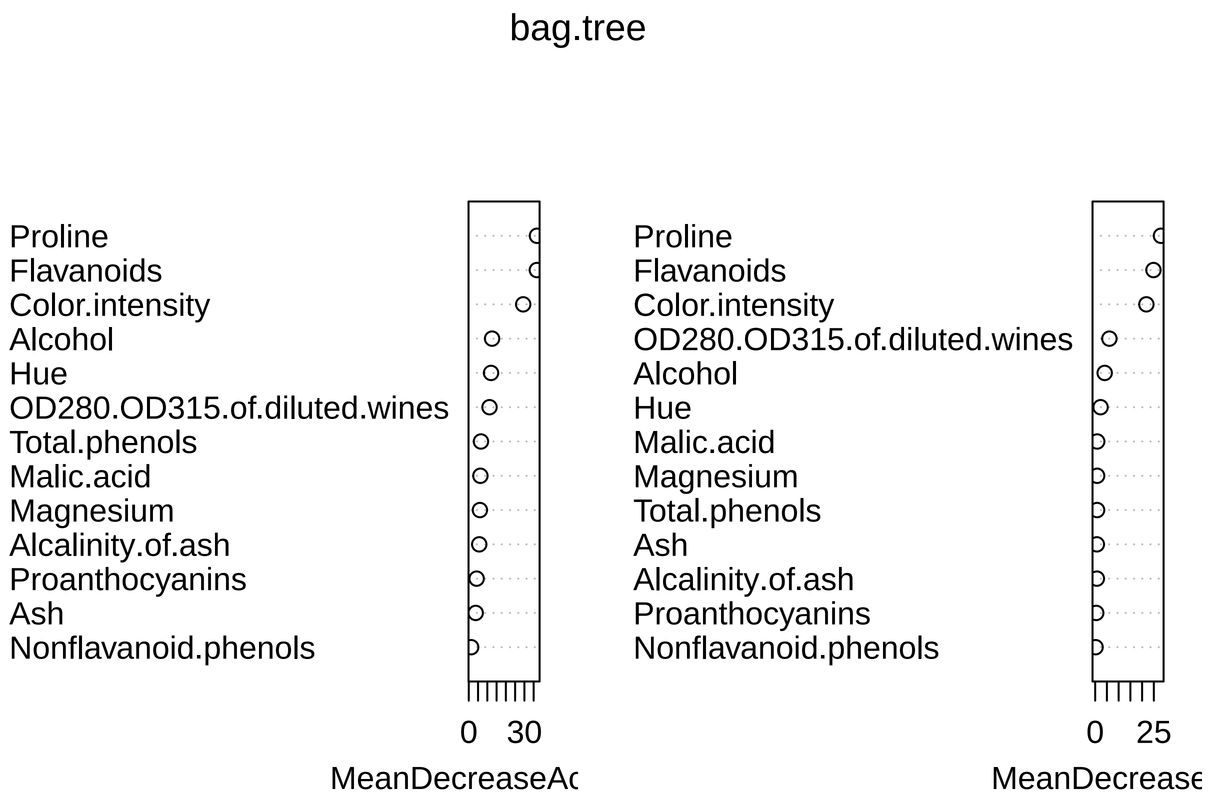
```
## The predicted error rate(random_forests) on the testing data set is : 0 %
```

Using the bagged classification tree and random forests, the predicted error rate on the testing data set drops to 0%, indicating that these ensemble methods substantially outperform the single classification tree (both the initial(19.44%) and pruned versions(19.44% and 16.67%)).

The 0% error rate is remarkable, and would normally be cause for skepticism. Looking at the class counts, we can see that the classes are well-balanced, and the amount of data, while on the smaller side, is still sufficient for machine learning. We can conclude, then that there are strong, learnable patterns, with well-separated classes, and that the decision boundary must be complex, as the single trees struggled, but not too complex, as the models were able to classify perfectly.

```
##   Class Count Proportion Dataset
## 1     1    49      0.345   Train
## 2     2    59      0.415   Train
## 3     3    34      0.239   Train
## 4     1    10      0.278    Test
## 5     2    12      0.333    Test
## 6     3    14      0.389    Test
```

We can also infer from the training metrics that we may have been lucky, as the model did not have perfect performance on the training set, despite the finding that the proline and flavanoids features are the most informative.



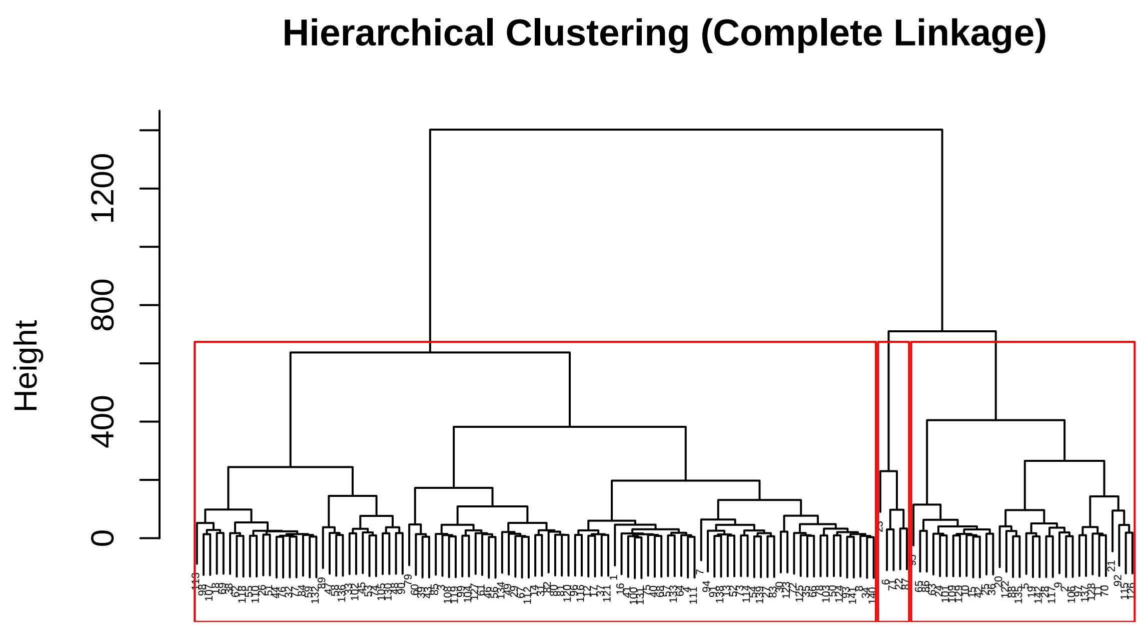
Part B

Clustering the wine dataset (Hierarchical clustering and k-means)

Q1.

Perform hierarchical clustering on the wine dataset, but do not include the Class feature. Group the data into three clusters and check/visualise whether this is a good reconstruction of the (actual) classes recorded in the Class feature.

```r
# Remove Class column for clustering
train_no_class <- train[, !(names(train) %in% "Class")]

# Distance matrix
dist_matrix <- dist(train_no_class)

# perform hierarchical clustering
hc.complete <- hclust(dist_matrix, method = "complete")
# plot(hc.complete, main = "Hierarchical Clustering (Complete Linkage)", xlab = "", sub = "", cex = .3)

# Cut tree into 3 clusters
clusters <- cutree(hc.complete, k = 3)

plot(hc.complete, main = "Hierarchical Clustering (Complete Linkage)", xlab = "", sub = "", cex = .3)
# Sepcify a chosen number of clusters
rect.hclust(hc.complete, k = 3, border = "red")
```



```r
# Get cluster assignments
clusters <- cutree(hc.complete, 3)
# Compare clusters to actual classes
cm <- table(Predicted = clusters, Actual = train$Class)
cm
```

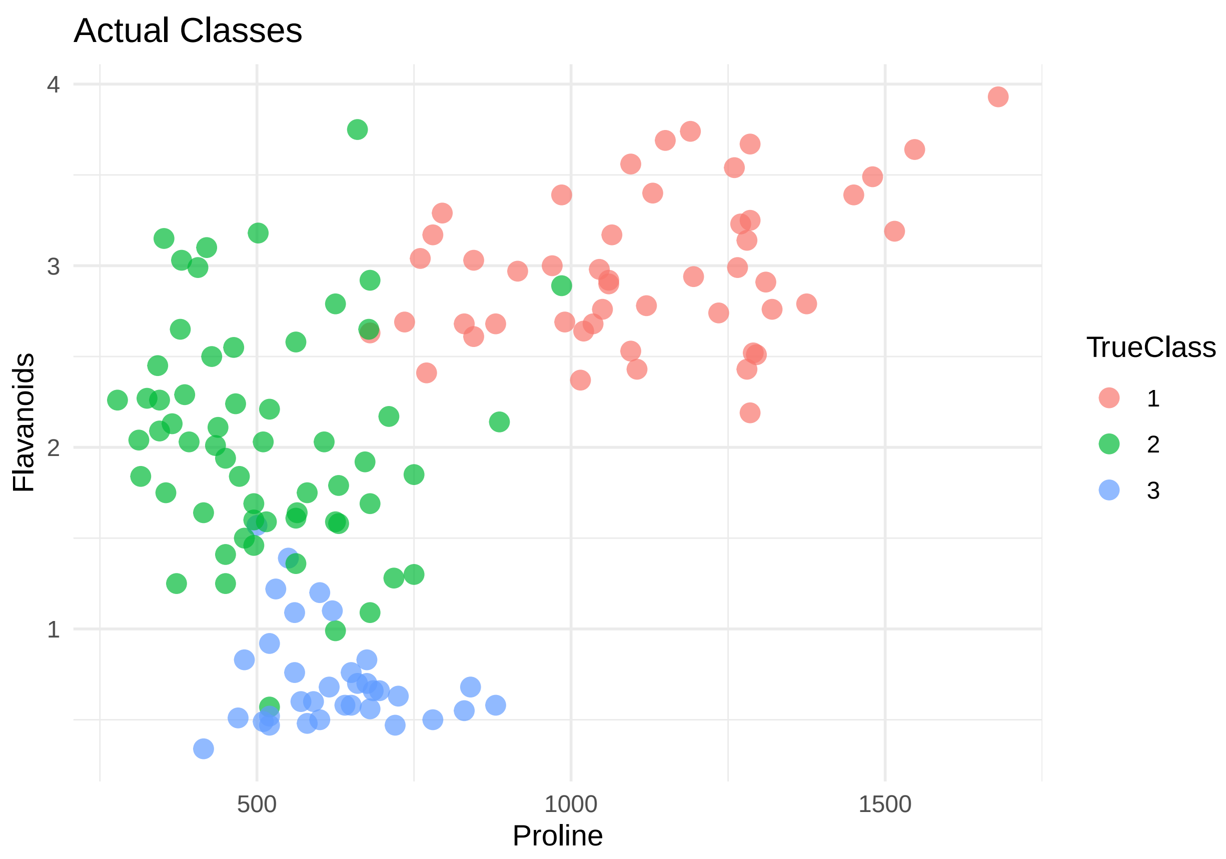
```
##           Actual
## Predicted  1  2  3
##          1 11 58 34
##          2 33  1  0
##          3  5  0  0
```

```
# get purity
purity <- sum(apply(cm, 1, max)) / sum(cm)
cat("\nPurity is", round(purity, 2))
```

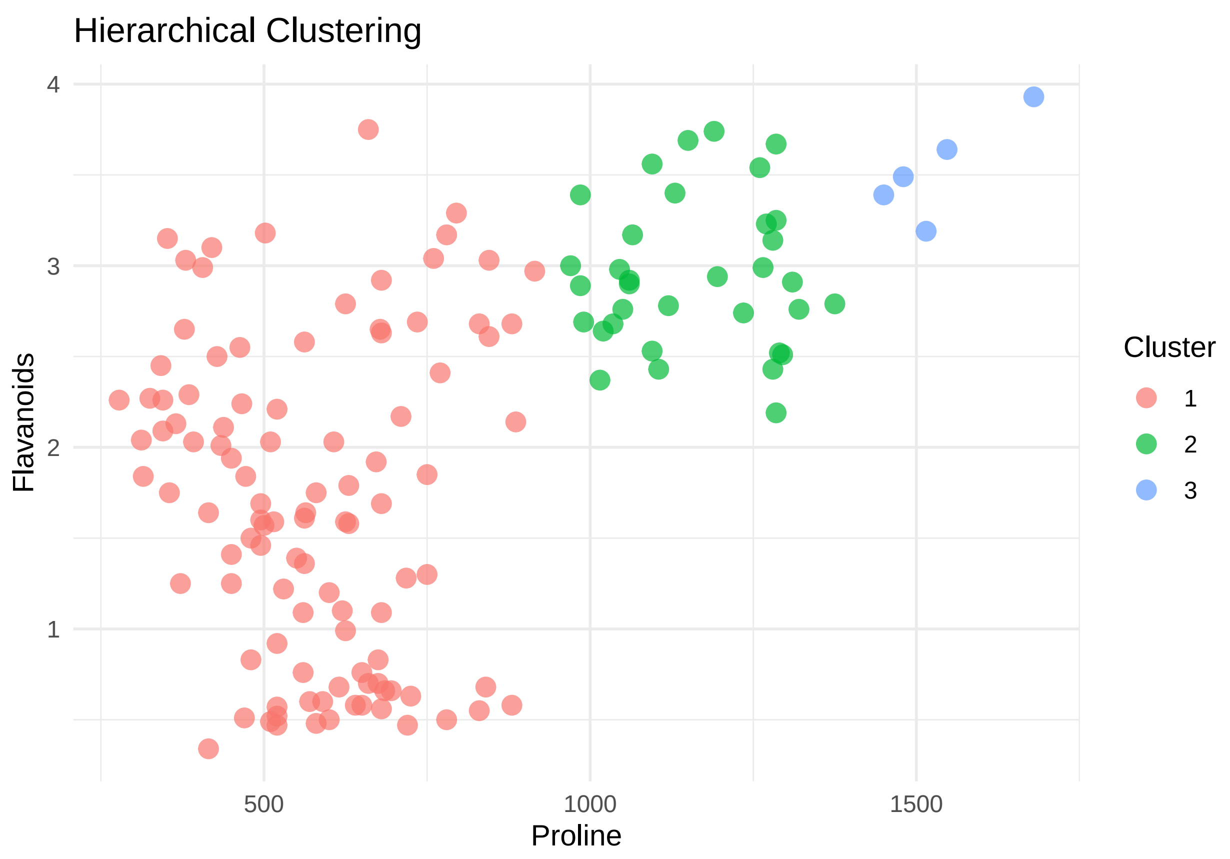
```
##
## Purity is 0.68
```

The confusion matrix shows unsatisfactory reconstruction, with the purity of the confusion matrix being 0.68. During this process, no standardization was performed; therefore, variables with larger absolute magnitudes exerted a dominant effect on the clustering outcome.

```
# Visualise the actual classes compared to the hierarchical clustering results.
plot_data <- data.frame(
  Proline = train$Proline,
  Flavanoids = train$Flavanoids,
  TrueClass = as.factor(train$Class),
  Cluster = as.factor(clusters)
)

# Ground Truth
p1 <- ggplot(plot_data, aes(x = Proline, y = Flavanoids, color = TrueClass)) +
  geom_point(size = 3, alpha = 0.7) +
  labs(title = "Actual Classes",
       x = "Proline", y = "Flavanoids") +
  theme_minimal()
p1
```



```r
# The clustering results
p2 <- ggplot(plot_data, aes(x = Proline, y = Flavanoids, color = Cluster)) +
  geom_point(size = 3, alpha = 0.7) +
  labs(title = "Hierarchical Clustering",
       x = "Proline", y = "Flavanoids") +
  theme_minimal()
p2
```



Q2.

Do the same using the k-means algorithm, for k=3. For visualisation, you can pick the two features that were sufficient for classification in question A, and plot datapoints in these two dimensions, comparing actual classes and predicted cluster labels.

```r
set.seed(1)
k <- 3
km <- kmeans(train_no_class, centers = k, nstart = 50)
km_clusters <- km$cluster

# Compute the confusion matrix
cm_km <- table(Cluster = km_clusters, Class = train$Class)
print("Confusion matrix (k-means clusters vs true Class):")
```

```
## [1] "Confusion matrix (k-means clusters vs true Class):"
```

```r
print(cm_km)
```

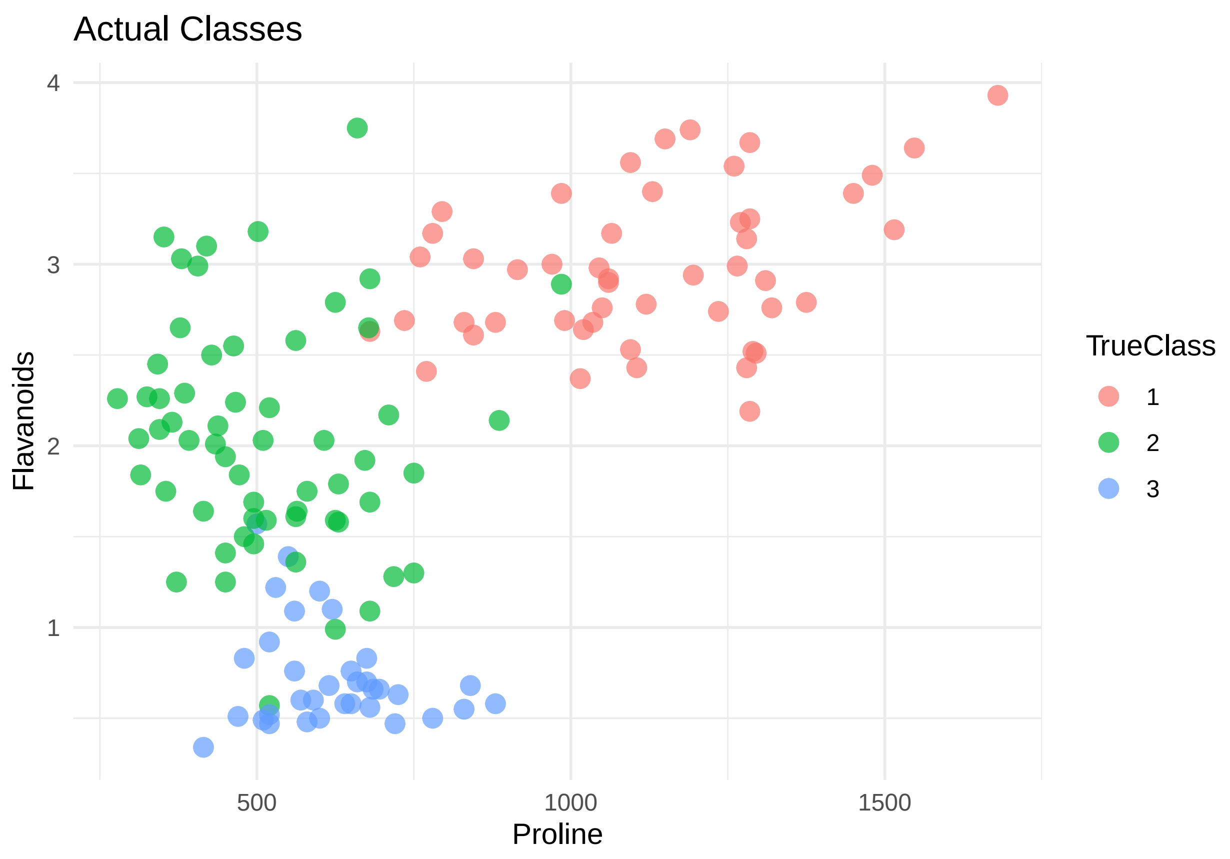
```
##        Class
## Cluster  1  2  3
##       1 11 17 20
##       2 38  1  0
##       3  0 41 14
```

```
# Compute the purity of the confusion matrix
purity <- sum(apply(cm_km, 1, max)) / sum(cm_km)
cat("The purity of the confusion matrix using the unscaled training data is",
    round(purity,2))
```

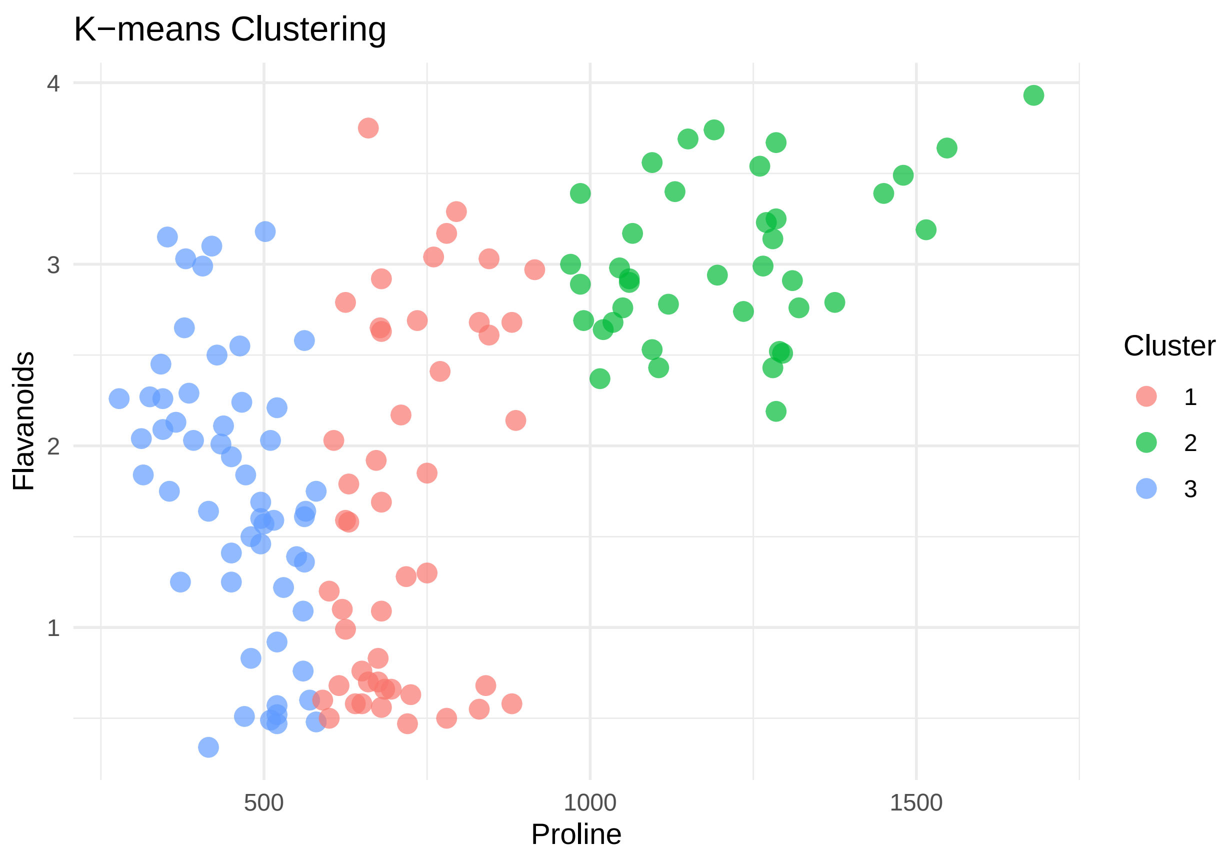
```
## The purity of the confusion matrix using the unscaled training data is 0.7
```

The confusion matrix shows that the clustering results using k-means has not resulted in much improvement, showing an incremental increase in purity score to 0.7.

```
# Visualise the actual classes compared to the k-means clustering results.
plot_data <- data.frame(
  Proline = train$Proline,
  Flavanoids = train$Flavanoids,
  TrueClass = as.factor(train$Class),
  Cluster = as.factor(km_clusters)
)

# Ground Truth
p1 <- ggplot(plot_data, aes(x = Proline, y = Flavanoids, color = TrueClass)) +
  geom_point(size = 3, alpha = 0.7) +
  labs(title = "Actual Classes",
       x = "Proline", y = "Flavanoids") +
  theme_minimal()
p1
```



```r
# The clustering results
p2 <- ggplot(plot_data, aes(x = Proline, y = Flavanoids, color = Cluster)) +
  geom_point(size = 3, alpha = 0.7) +
  labs(title = "K-means Clustering",
       x = "Proline", y = "Flavanoids") +
  theme_minimal()
p2
```



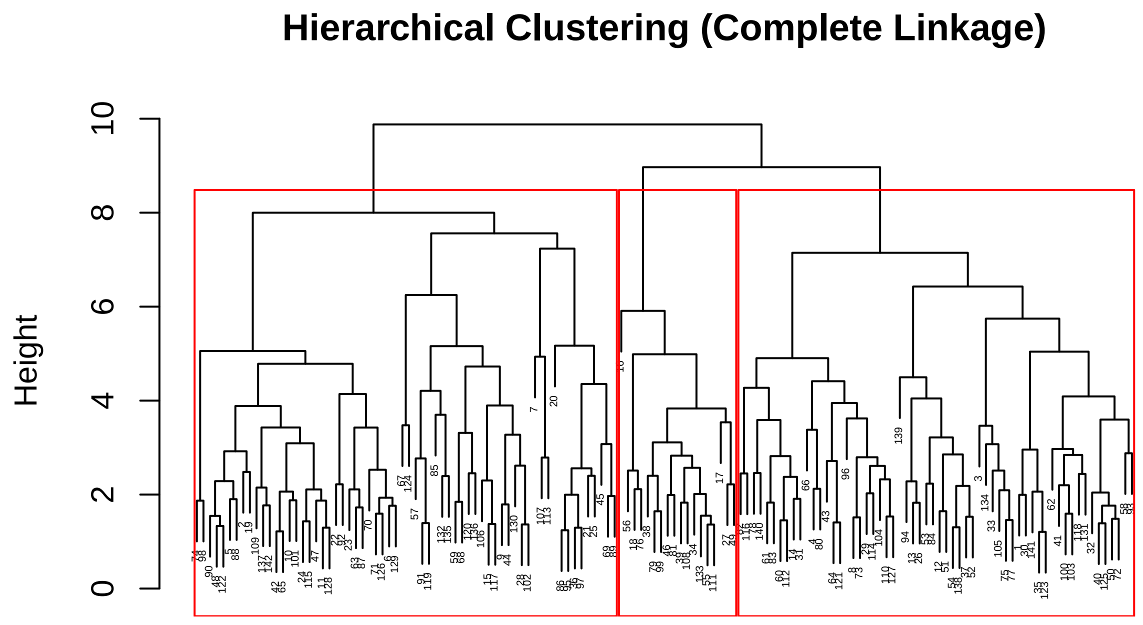
Q3.

You will likely not get great results, because your features vary on very different orders of magnitude (for example, Nonflavanoid.phenols is mostly between 0 and 1, but Proline is in the 1000 range). Normalise all numerical features using either z-score transformation or min-max normalisation, which will bring them into comparable orders of magnitude. Then repeat parts 1. and 2., and see whether your results improve.

Q3.1 - Hierarchical Clustering with scaled data

```r
# Remove Class column for clustering
train_no_class <- train[, !(names(train) %in% "Class")]

# Scale all predictors
scaled_train <- scale(train_no_class)
scaled_train <- as.data.frame(scaled_train)

# Distance matrix
```

```r
dist_matrix_scaled <- dist(scaled_train)

# perform hierarchical clustering on scaled data using complete linkage
hc.complete_scaled <- hclust(dist_matrix_scaled, method = "complete")

# Cut tree into 3 clusters
hc_clusters_scaled <- cutree(hc.complete_scaled, k = 3)
plot(hc.complete_scaled, main = "Hierarchical Clustering (Complete Linkage)", xlab = "", sub = "", cex =

# Sepcify a chosen number of clusters
rect.hclust(hc.complete_scaled, k = 3, border = "red")
```



```r
# Compare clusters to actual classes
cm_clusters_scaled <- table(Predicted = hc_clusters_scaled, Actual = train$Class)
cm_clusters_scaled
```

```
##            Actual
## Predicted  1  2  3
##          1  0 26 34
##          2 49 15  0
##          3  0 18  0
```

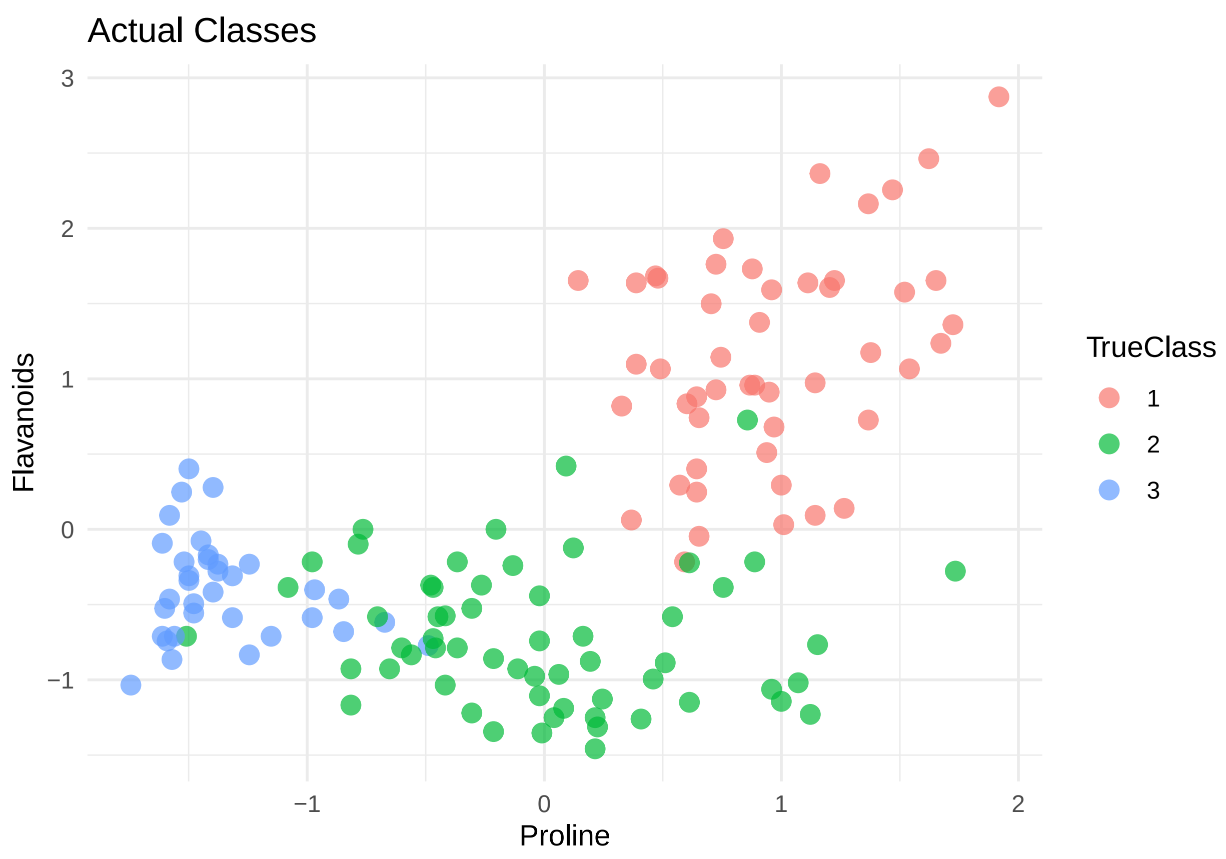
```r
# Compute the purity of the confusion matrix
purity <- sum(apply(cm_clusters_scaled, 1, max)) / sum(cm_clusters_scaled)
```

```r
cat("\nThe purity of the confusion matrix using the clustered data is",
    round(purity,2))
```

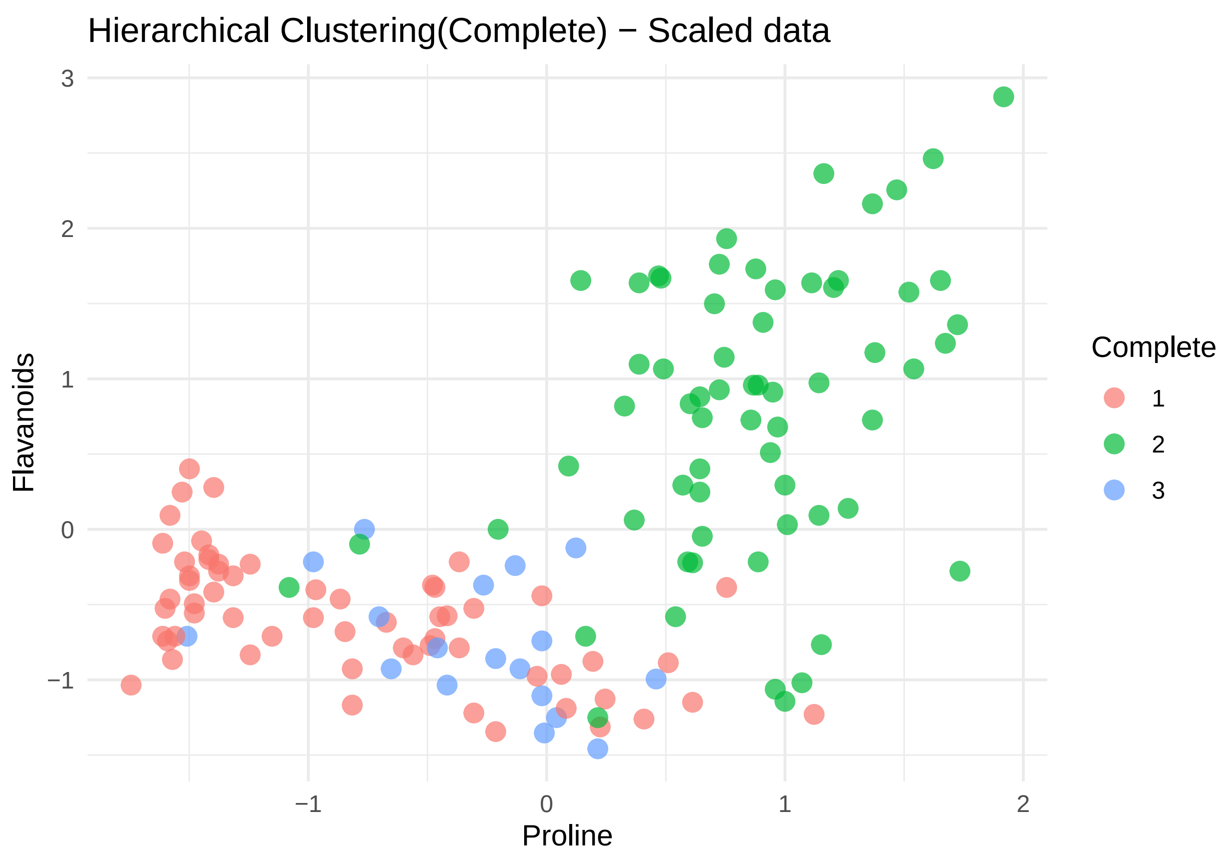
```
##
## The purity of the confusion matrix using the clustered data is 0.71
```

When applying hierarchical clustering on the normalized training data, the purity improvement of the confusion matrix is negligible(from 0.68 to 0.71), suggesting that normalization only slightly enhances the alignment between the clusters and the true classes. Therefore,in hierarchical clustering, the advantages of data standardization are not necessarily significant.

```r
# Visualise the actual classes compared to the hierarchical clustering results(Normalized).
plot_data <- data.frame(
  Proline = scaled_train[, "Proline"],
  Flavanoids = scaled_train[, "Flavanoids"],
  TrueClass = as.factor(train$Class),
  Complete = as.factor(hc_clusters_scaled)
)

# Ground Truth
p1 <- ggplot(plot_data, aes(x = Flavanoids, y = Proline, color = TrueClass)) +
  geom_point(size = 3, alpha = 0.7) +
  labs(title = "Actual Classes",
       x = "Proline", y = "Flavanoids") +
  theme_minimal()
p1
```



```
# The clustering results using complete linkage
p2 <- ggplot(plot_data, aes(x = Flavanoids, y =Proline , color = Complete)) +
  geom_point(size = 3, alpha = 0.7) +
  labs(title = "Hierarchical Clustering(Complete) - Scaled data",
       x = "Proline", y = "Flavanoids") +
  theme_minimal()
p2
```



The comparison plots further confirm that hierarchical clustering with complete linkage on the scaled data improves performance only slightly, with some class confusion still remaining.

Q3.2 - K-means with scaled data

```
# K-means clustering with k=3
km.wine_scaled <- kmeans(scaled_train, centers = 3, nstart = 25)
# Get cluster assignments
km.clusters_scaled <- km.wine_scaled$cluster

# Compare clusters to actual classes
cm_km_scaled <- table(Predicted = km.clusters_scaled, Actual = train$Class)
cm_km_scaled
```

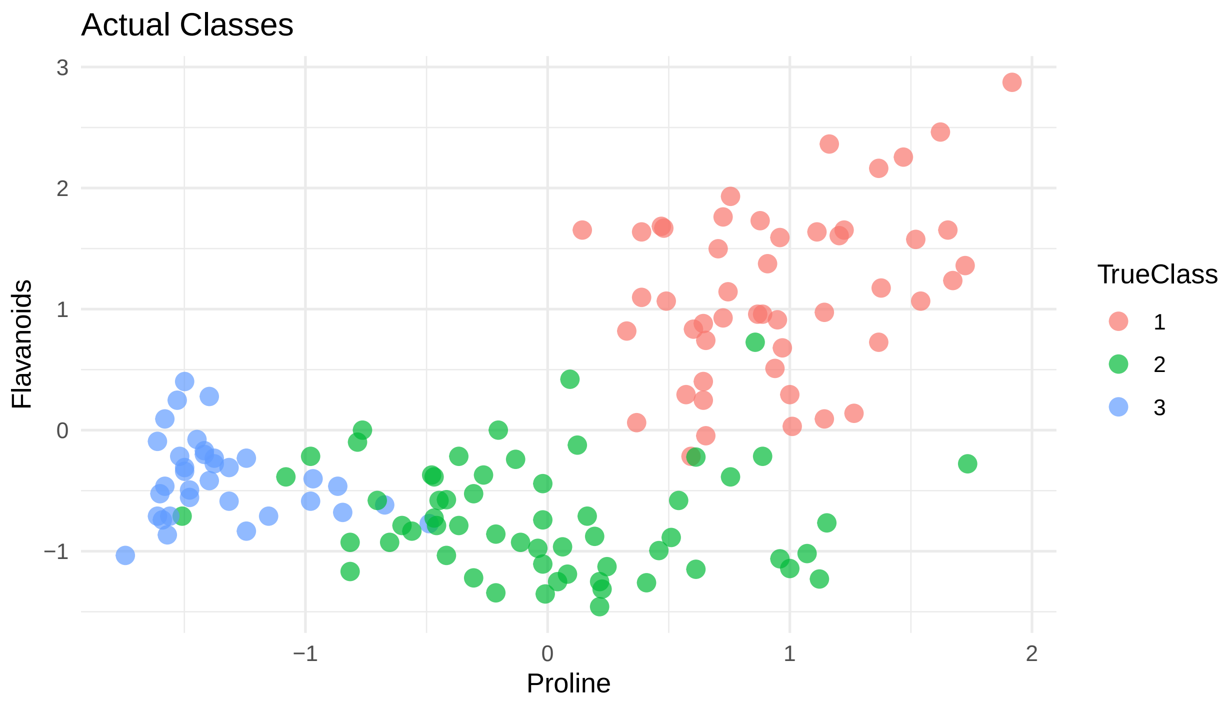
```
##          Actual
## Predicted  1  2  3
##         1  0 55  0
##         2 49  1  0
##         3  0  3 34
```

```
# Compute the purity of the confusion matrix
purity <- sum(apply(cm_km_scaled, 1, max)) / sum(cm_km_scaled)
cat("\nThe purity of the confusion matrix using the k-means clustered data is",
    round(purity,2))
```

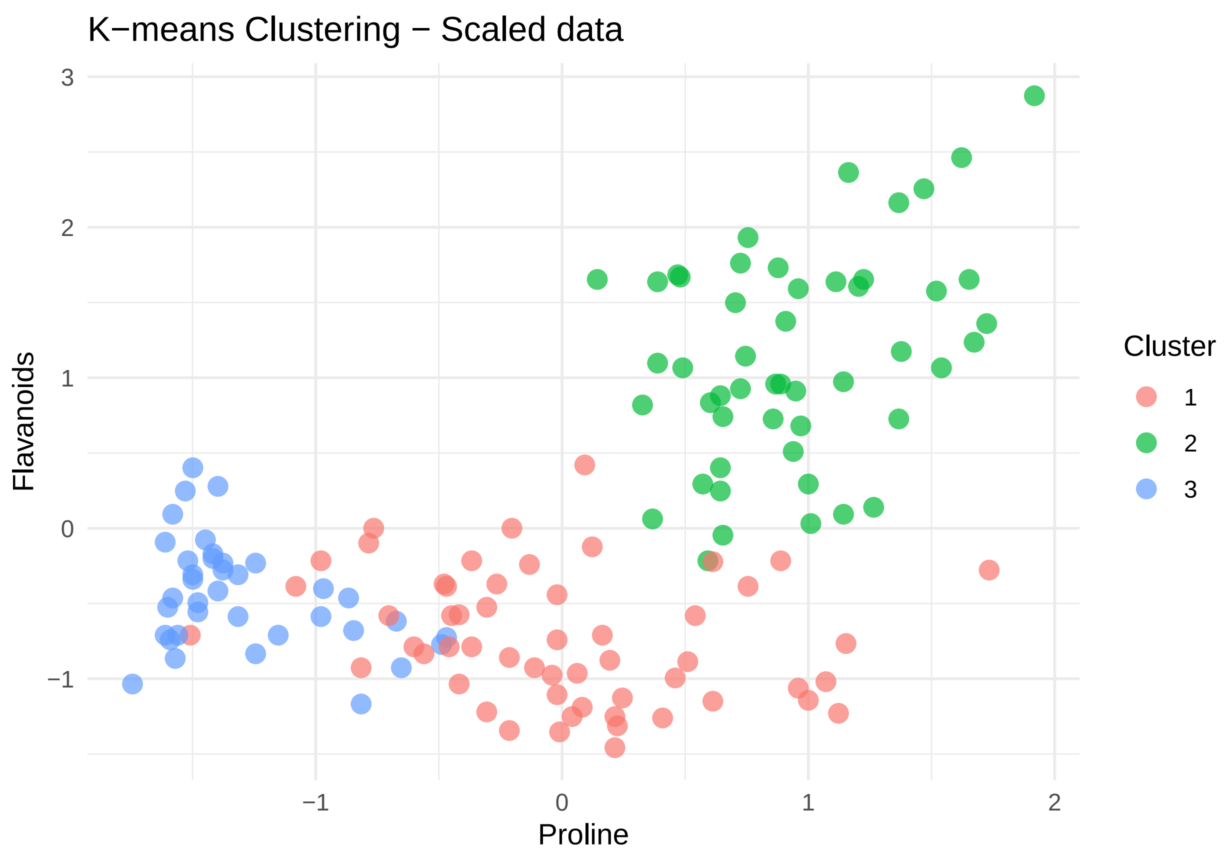
```
##
## The purity of the confusion matrix using the k-means clustered data is 0.97
```

When applying k-means clustering on the normalized training data, the purity of the confusion matrix improved markedly from 0.70 to 0.97, suggesting that normalization greatly enhances the alignment between the clusters and the true classes. As a result, features with smaller numeric ranges (e.g., Flavanoids) contribute equally to the clustering, leading to a more balanced and accurate partition of the data.

```r
# Visualize the actual classes compared to the k-means clustering results(Normalized).
plot_data <- data.frame(
  Proline = scaled_train[, "Proline"],
  Flavanoids = scaled_train[, "Flavanoids"],
  TrueClass = as.factor(train$Class),
  Cluster = as.factor(km.clusters_scaled)
)

# Ground Truth
p1 <- ggplot(plot_data, aes(x = Flavanoids, y = Proline, color = TrueClass)) +
  geom_point(size = 3, alpha = 0.7) +
  labs(title = "Actual Classes",
       x = "Proline", y = "Flavanoids") +
  theme_minimal()
p1
```



```r
# The clustering results
p2 <- ggplot(plot_data, aes(x = Flavanoids, y =Proline , color = Cluster)) +
  geom_point(size = 3, alpha = 0.7) +
  labs(title = "K-means Clustering - Scaled data",
       x = "Proline", y = "Flavanoids") +
  theme_minimal()
p2
```



The comparison plots further confirm that, after normalization, the k-means clustering results are very close to the actual class labels, with only a few points remaining inconsistent.