Homework 4

Ngoc Duong

11/26/2020

8.4. Problem 5

Given a two-class classification problem (Red and Green), ten bootstrapped samples are obtained from the data. The estimates of $P(Y = Red|X)$ were obtained from a classification tree on these bootstrapped samples as: 0.1, 0.15, 0.2, 0.2, 0.55, 0.6, 0.6, 0.65, 0.7, 0.75.

Under the two approaches to combine the estimated probabilities into a single class prediction:

```
# Q8.4.5
p = c(0.1, 0.15, 0.2, 0.2, 0.55, 0.6, 0.6, 0.65, 0.7, 0.75)
pred.red = mean(p >= 0.5) #majority vote approach
pred.red
```

```
## [1] 0.6
```

```
phat = mean(p) #mean probability approach
phat
```

```
## [1] 0.45
```

- Majority vote approach: there are 6 out of 10 trees that predicted class red as outcome given X, or 60%, so using this approach the final single class prediction is "red".
- Average probability approach: taking the average of these 10 probabilities gives $(0.1 + 0.15 + 0.2 + 0.2 + 0.55 + 0.6 + 0.6 + 0.65 + 0.7 + 0.75)/10 = 0.45 < 0.5$; therefore, the final classification under this approach is "green".

8.4. Problem 9

a) Create training set – random sample of 800, and test set with remaining observations

b) Fit a tree to the data and describe results

```
##
## Classification tree:
## tree(formula = purchase ~ ., data = oj_train)
## Variables actually used in tree construction:
## [1] "loyal_ch"        "price_diff"      "special_ch"      "list_price_diff"
## [5] "pct_disc_mm"
## Number of terminal nodes:  9
## Residual mean deviance:  0.7432 = 587.8 / 791
## Misclassification error rate: 0.1588 = 127 / 800
```

From the summary output, we see the tree has 9 terminal nodes, and the splitting criteria are from these 5 variables: "loyal_ch", "price_diff", "special_ch", list_price_diff", and"pct_disc_mm". The training error rate is 127/800, or 15.88%.

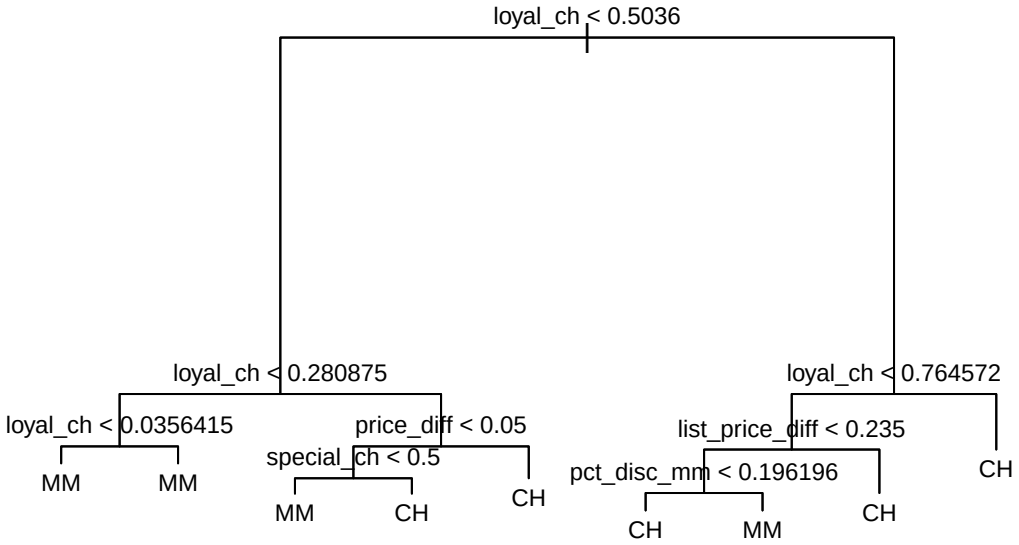
c) Detailed text output and interpretation

```
## node), split, n, deviance, yval, (yprob)
##       * denotes terminal node
##
##  1) root 800 1073.00 CH ( 0.60625 0.39375 )
##    2) loyal_ch < 0.5036 365  441.60 MM ( 0.29315 0.70685 )
##      4) loyal_ch < 0.280875 177  140.50 MM ( 0.13559 0.86441 )
##        8) loyal_ch < 0.0356415 59   10.14 MM ( 0.01695 0.98305 ) *
##        9) loyal_ch > 0.0356415 118  116.40 MM ( 0.19492 0.80508 ) *
##      5) loyal_ch > 0.280875 188  258.00 MM ( 0.44149 0.55851 )
##       10) price_diff < 0.05 79   84.79 MM ( 0.22785 0.77215 )
##         20) special_ch < 0.5 64   51.98 MM ( 0.14062 0.85938 ) *
##         21) special_ch > 0.5 15   20.19 CH ( 0.60000 0.40000 ) *
##       11) price_diff > 0.05 109  147.00 CH ( 0.59633 0.40367 ) *
##    3) loyal_ch > 0.5036 435  337.90 CH ( 0.86897 0.13103 )
##      6) loyal_ch < 0.764572 174  201.00 CH ( 0.73563 0.26437 )
##       12) list_price_diff < 0.235 72   99.81 MM ( 0.50000 0.50000 )
##         24) pct_disc_mm < 0.196196 55   73.14 CH ( 0.61818 0.38182 ) *
##         25) pct_disc_mm > 0.196196 17   12.32 MM ( 0.11765 0.88235 ) *
##       13) list_price_diff > 0.235 102   65.43 CH ( 0.90196 0.09804 ) *
##      7) loyal_ch > 0.764572 261   91.20 CH ( 0.95785 0.04215 ) *
```

The nodes with asterisk signs denote terminal nodes. We can randomly pick a terminal node (9). This node gives the split criterion (loyal_ch > 0.03564), but we also need to incorporate the parent nodes' splitting criteria (loyal_ch < 0.5036 and loyal_ch < 0.2808), so combined together, we have the overall predicted class "MM" for "loyal_ch" between 0.0356 and 0.2754.

The number of observations falling into this branch is 118, and 19.42% (or 23 observations) of those have class "CH" while 80.51% have class "MM" (or 95 observations). The deviance (which measures node impurity for classification tree) is 116.4 for this branch.

d) Tree plot and interpretation



The most important predictor of "purchase" seems to be "loyal_ch", since the first split differentiates "CH" (loyal_ch > 0.7645) from "MM" (loyal_ch < 0.2808) using this criterion.

For regions of loyal_ch between 0.2808 and 0.7645, "price_diff" and "list_price_diff" determines the split points for the next branches, followed by special_ch and "pct_disc_mm" which return the terminal nodes.

Overall, it seems like observations with lower "loyal_ch", "price_diff" and "special_ch" are more likely to be classified as "MM".

e) Predict reponse on test data and produce confusion matrix

```
##
## tree.pred  CH  MM
##        CH 160  38
##        MM   8  64
```

```
## [1] 0.1703704
```

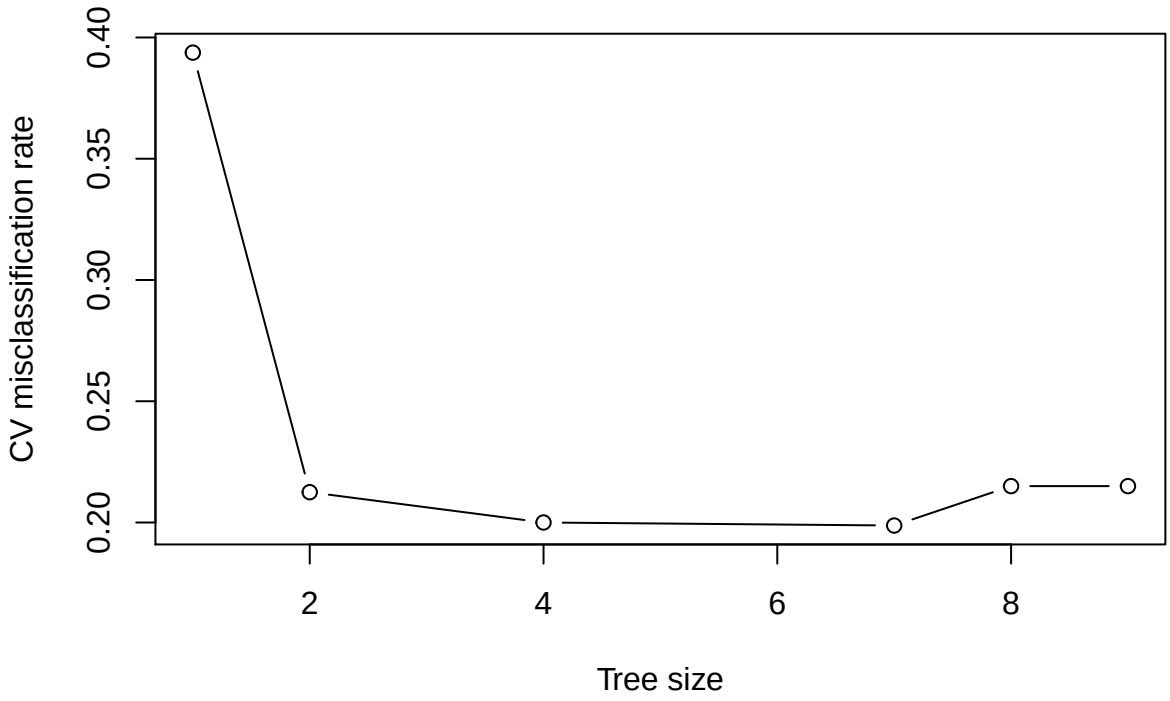
Applying the classification tree obtained from training data to test data, we have the test error rate as $(38 + 8)/270 = 0.1703704$.

f) Apply cv.tree function

We use the argument FUN = prune.misclass to specify misclassification rate as the criterion to guide the cross-validation and pruning process.

```
set.seed(77)
cv.tree.mod = cv.tree(tree.mod, FUN=prune.misclass, K = 10)
```

g) Plot CV error against tree sizes



h) Optimal tree size (CV)

```
#size of tree with minimum error
cv.tree.mod$size[which.min(cv.tree.mod$dev)]
```

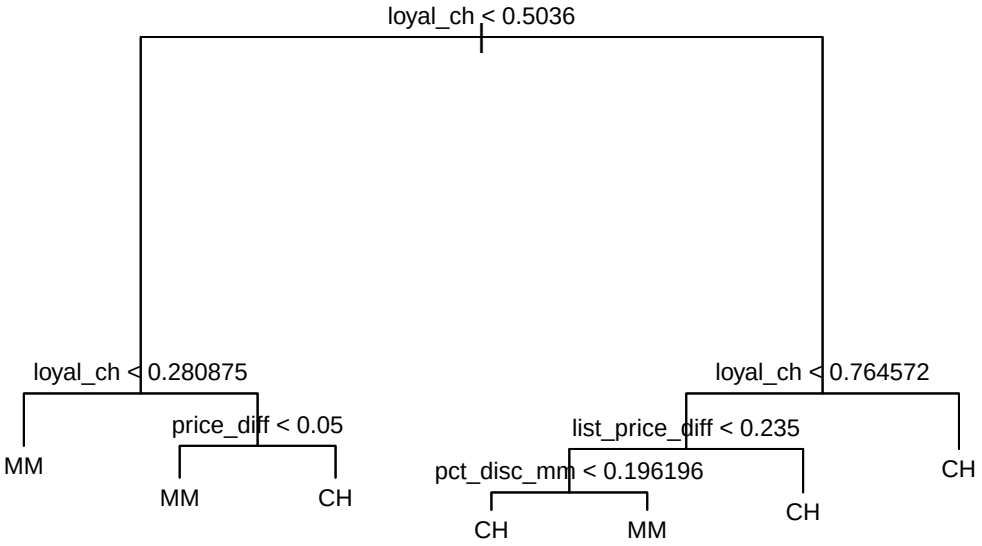
```
## [1] 7
```

From the plot, we can see tree size 7 corresponds to the lowest CV-classification error rate in this case. This is optimal tree size is specific to this particular training data which also depends on the set seed.

i) Pruned tree

```
##
```

```
## Classification tree:
## snip.tree(tree = tree.mod, nodes = c(10L, 4L))
## Variables actually used in tree construction:
## [1] "loyal_ch"        "price_diff"      "list_price_diff" "pct_disc_mm"
## Number of terminal nodes:  7
## Residual mean deviance:  0.7748 = 614.4 / 793
## Misclassification error rate: 0.1625 = 130 / 800
```



j) Compare training error rate between pruned and unpruned tree

On the training set, the pruned tree gives slightly higher training misclassification rate than the unpruned tree (16.25% vs. 15.88%). This is reasonable, as the unpruned tree fits the training data very well, but may cause overfitting/high variance when applied to test set.

k) Compare test error rate between pruned and unpruned tree

```
##
## prune.tree.pred  CH  MM
##              CH 160  36
##              MM   8  66
```

```
## [1] 0.162963
```

The test error rate for the pruned tree is $(36 + 8)/270 = 0.163$ which is lower than the test error rate for the unpruned tree (0.1704). Pruned trees might have lower variance compared to unpruned trees.

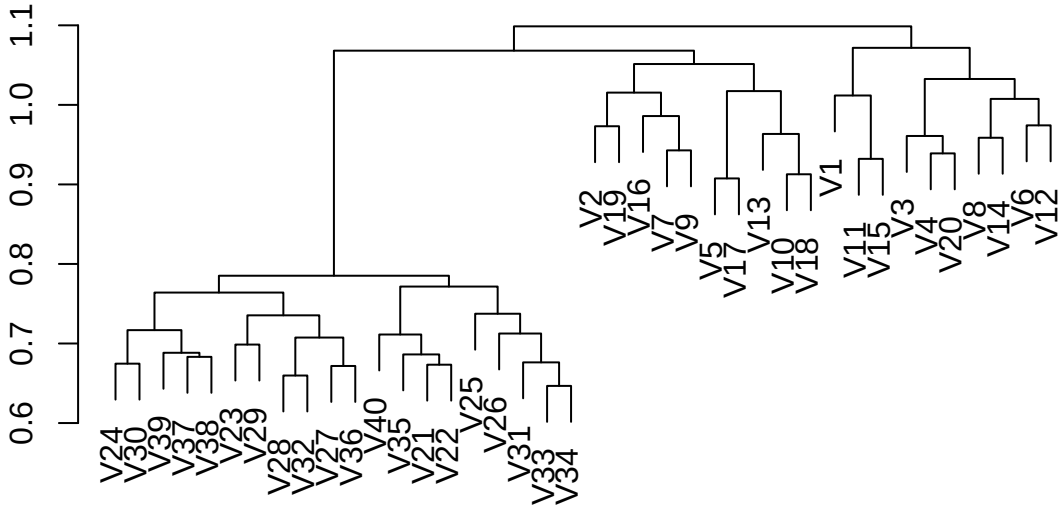
10.7. Problem 11

a) Load in the dataset. The data consist 40 tissue samples with measurements on 1,000 genes. The first 20 samples are from healthy patients, while the second 20 are from a diseased group.

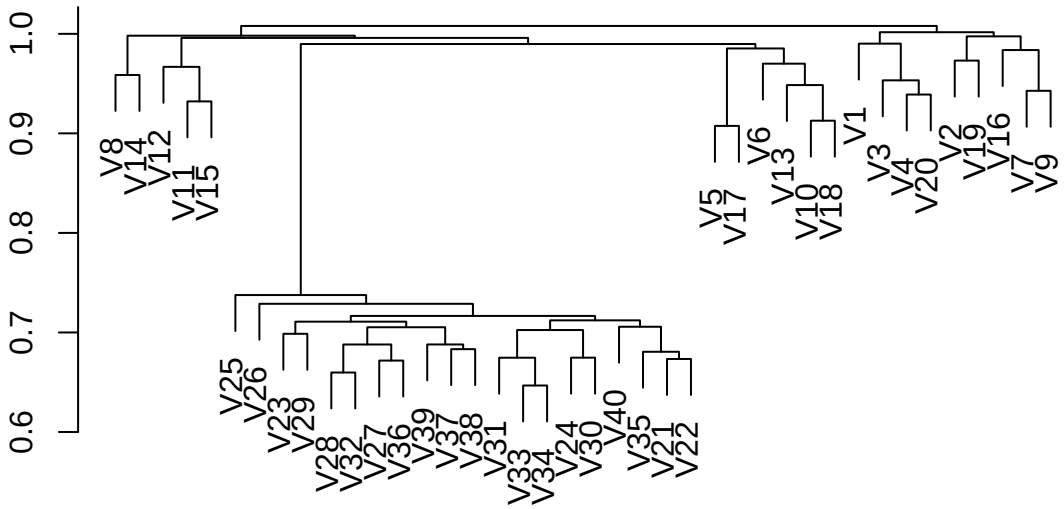
```r
# Q10.7.11
data = read.csv("ch10ex11.csv", header = FALSE)

data_label = as.data.frame(t(data)) #transpose
data_label$status = c(rep("Healthy", 20), rep("Diseased", 20)) #add disease status label
```

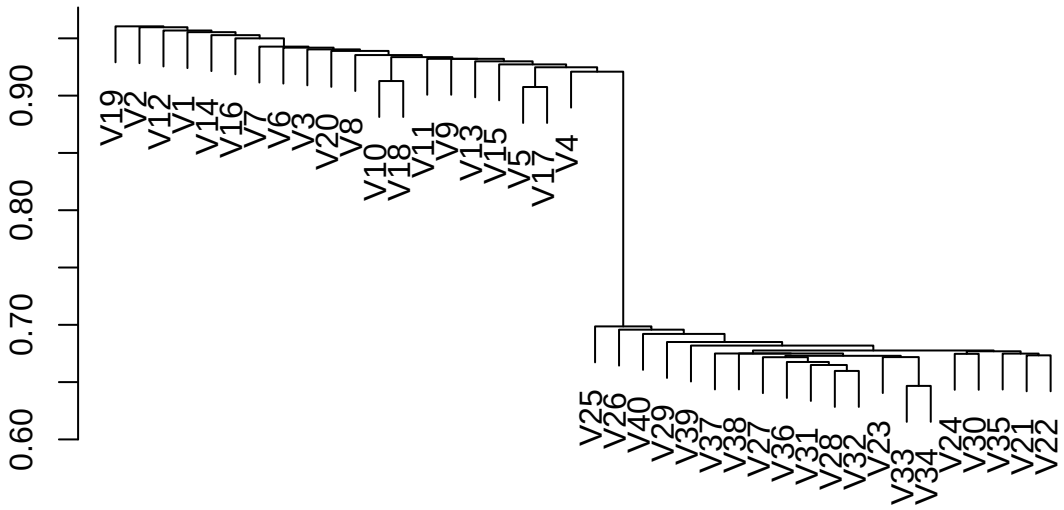
b) Apply hierarchical clustering using correlation-based distance, and plot the dendogram.

Complete linkage



Average Linkage





Do the genes separate the samples into two groups? Do results depend on linkage type?

The genes generally separate the samples into two groups (for complete linkage and single linkage) using correlation-based distance. The two groups are a little less distinctive when using "average linkage" (there can be 3 or 4 groups depending on where to cut the tree/dendogram). Since we end up with different clusterings, the results depend on the type of linkage used.

Which genes differ the most across the two groups?

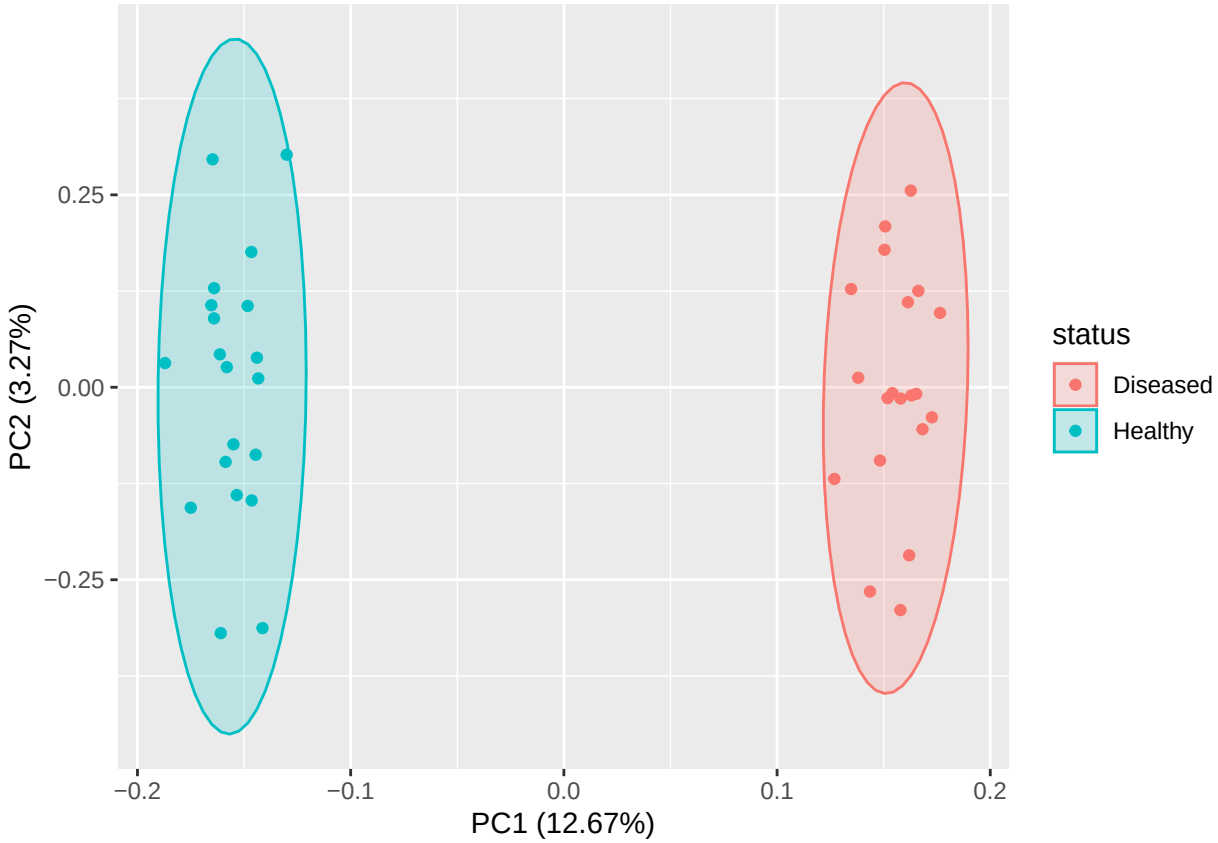
To look at which genes differ the most across the two types of patients, we can first check to see if PC's from PCA separate the two classes well. Then, we can look at the loading vectors from PCA to see which genes can explain the variance the most.

We can look at the variations captured by the PCs individually and cumulatively

```
## Importance of components:
##                             PC1     PC2     PC3     PC4     PC5     PC6     PC7
## Standard deviation     11.9409 6.06818 5.93476 5.83115 5.75209 5.70031 5.63448
## Proportion of Variance  0.1267 0.03271 0.03129 0.03021 0.02939 0.02887 0.02821
## Cumulative Proportion   0.1267 0.15939 0.19068 0.22089 0.25029 0.27915 0.30736
##                             PC8     PC9    PC10    PC11    PC12    PC13    PC14
## Standard deviation     5.57726 5.54943 5.50625 5.48852 5.46025 5.40230 5.33441
## Proportion of Variance 0.02764 0.02736 0.02694 0.02676 0.02649 0.02593 0.02528
## Cumulative Proportion  0.33499 0.36236 0.38929 0.41605 0.44254 0.46847 0.49375
##                            PC15    PC16    PC17    PC18    PC19    PC20    PC21
## Standard deviation     5.27756 5.21594 5.20000 5.15140 5.11600 5.05591 5.03836
## Proportion of Variance 0.02475 0.02417 0.02402 0.02358 0.02325 0.02271 0.02255
## Cumulative Proportion  0.51850 0.54267 0.56669 0.59027 0.61352 0.63623 0.65878
##                            PC22    PC23    PC24    PC25    PC26    PC27    PC28
## Standard deviation     5.01868 4.95965 4.91393 4.86397 4.81796 4.80811 4.73485
## Proportion of Variance 0.02238 0.02185 0.02145 0.02102 0.02062 0.02054 0.01992
## Cumulative Proportion  0.68116 0.70301 0.72447 0.74548 0.76611 0.78665 0.80656
##                            PC29    PC30    PC31    PC32    PC33    PC34    PC35
## Standard deviation     4.70098 4.65564 4.61621 4.56733 4.53032 4.49528 4.36502
## Proportion of Variance 0.01963 0.01926 0.01893 0.01853 0.01823 0.01795 0.01693
## Cumulative Proportion  0.82620 0.84545 0.86439 0.88292 0.90115 0.91910 0.93603
##                            PC36    PC37    PC38    PC39      PC40
```

```
## Standard deviation     4.35858 4.26700 4.20277 4.13922 5.251e-15
## Proportion of Variance 0.01688 0.01618 0.01569 0.01522 0.000e+00
## Cumulative Proportion  0.95291 0.96909 0.98478 1.00000 1.000e+00
```

```
## Warning: `select_()` is deprecated as of dplyr 0.7.0.
## Please use `select()` instead.
## This warning is displayed once every 8 hours.
## Call `lifecycle::last_warnings()` to see where this warning was generated.
```



- After getting the loadings for each gene, the first 20 genes with highest loadings and their loadings maginitude are as follows

```
##  [1] 865  68 911 428 624  11 524 803 980 822 529 765 801 771 570 654 451 237 373
## [20] 959
```

```
##      V865       V68      V911      V428      V624       V11      V524      V803
## 0.7765416 0.7137785 0.7099501 0.6363706 0.6195945 0.5885202 0.5583279 0.5535498
##      V980      V822      V529      V765      V801      V771      V570      V654
## 0.5217130 0.4981997 0.4868558 0.4846791 0.4814929 0.4795595 0.4777313 0.4679100
##      V451      V237      V373      V959
## 0.4632665 0.4599649 0.4586729 0.4576528
```

From the plot, we can see that the first 2 PCs separate the classes well, and by obtaining the loading vectors for the genes, we can rank the loadings to get the top 20 differentiating genes (genes 865, 68, 911, 428, 624, 11, etc.)

Appendix

```
knitr::opts_chunk$set(echo = TRUE)
library(tidyverse)
library(janitor)
library(e1071)
library(ModelMetrics)
library(caret)
library(microbenchmark)
library(broom)
library(tree)
library(ISLR)
library(ggfortify)
# Q8.4.5
p = c(0.1, 0.15, 0.2, 0.2, 0.55, 0.6, 0.6, 0.65, 0.7, 0.75)
pred.red = mean(p >= 0.5) #majority vote approach
pred.red
phat = mean(p) #mean probability approach
phat
data(OJ)
#create training set with 800 obs and test set with remaining obs
oj_data = OJ %>% janitor::clean_names() %>%
  mutate(purchase = as.factor(purchase))

set.seed(1)
rowTrain = sample(nrow(oj_data), 800)
oj_train = oj_data[rowTrain,]
oj_test = oj_data[-rowTrain,]
#b) fit a tree
tree.mod = tree(purchase ~., data = oj_train)
summary(tree.mod)
#c) detailed text output
tree.mod
#d) create plot for tree
plot(tree.mod)
text(tree.mod, pretty = 0, cex = 0.75)
tree.pred = predict(tree.mod, oj_test[,-1], type = "class") #predict on test data
table(tree.pred, oj_test$purchase) #confusion matrix
#misclassification rate
(table(tree.pred, oj_test$purchase)[1,2] + table(tree.pred, oj_test$purchase)[2,1])/270
set.seed(77)
cv.tree.mod = cv.tree(tree.mod, FUN=prune.misclass, K = 10)
#plot CV error vs. tree size
plot(cv.tree.mod$size, cv.tree.mod$dev/nrow(oj_train),
     type = "b", xlab = "Tree size", ylab = "CV misclassification rate")
#cv.tree.mod$size
#cv.tree.mod$dev
#size of tree with minimum error
cv.tree.mod$size[which.min(cv.tree.mod$dev)]
pruned.tree.mod = prune.tree(tree.mod, best = 7)
summary(pruned.tree.mod)
plot(pruned.tree.mod)
text(pruned.tree.mod, pretty = 0, cex = 0.75)
```

```r
prune.tree.pred = predict(pruned.tree.mod, oj_test[,-1], type = "class")
cm.prune = table(prune.tree.pred, oj_test$purchase)
cm.prune

(cm.prune[1,2]+cm.prune[2,1])/270
# Q10.7.11
data = read.csv("ch10ex11.csv", header = FALSE)

data_label = as.data.frame(t(data)) #transpose
data_label$status = c(rep("Healthy", 20), rep("Diseased", 20)) #add disease status label
#hierarchical clustering
cor.dist <-  as.dist(1 - cor(scale(data))) #correlation-based distance

#specify clustering types
hc.complete <-hclust(cor.dist, method = "complete")
hc.average <-hclust(cor.dist, method = "average")
hc.single <-hclust(cor.dist, method = "single")

#plot dendograms
plot(hc.complete, labels = colnames(data), main = "Complete linkage", xlab = "", ylab = "", sub = "")
plot(hc.average, labels = colnames(data), main = "Average Linkage", xlab = "", ylab = "", sub = "")
plot(hc.single, labels = colnames(data), main = "Single Linkage", xlab = "", ylab = "", sub = "")
#PCA
pca = prcomp(data_label[,-1001])
summary(pca) #variation captured by 40PCs
autoplot(pca, data = data_label, colour = "status",
         frame = TRUE, frame.type = "norm") #clustering of subjects based on first 2PCs
load.total = apply(pca$rotation, 1, sum) #obtain loadings for each gene
index = order(abs(load.total), decreasing = TRUE) #order the genes with highest loading first
index[1:20] #first 20 genes with highest loadings
abs(load.total[index[1:20]]) #the loadings for these 20 genes
```