Big Data Analytics with R - Coursework 4

1. SVM (Textbook 9.7.8)

1.a. We start by loading the `ISLR` package and setting the seed for the session, after which we create a training set of 800 randomly chosen observations and a test set containing the remaining observations.

```
library(ISLR)
set.seed(1)
training_indices <- sample(nrow(OJ), 800)
train <- OJ[training_indices, ]
test <- OJ[-training_indices, ]
nrow(train)
```

```
## [1] 800
```

```
nrow(test)
```

```
## [1] 270
```

1.b. Next we fit a support vector classifier model, using `Purchase` (a categorical variable indicating whether Citrus Hill (`CH`) or Minute Maid (`MM`) orange juice was bought) as the response variable and the remaining variables as predictors.

We use the SVC as implemented in the e1071 package, with the hyperparameter cost $= 0.01$. Since we are using a support vector classifier, which uses a linear kernel, we set this option as well.

```
library(e1071)
```

```
## Warning: package 'e1071' was built under R version 3.1.1
```

```
fit <- svm(Purchase ~ ., data=OJ, cost=0.01,
           subset=training_indices, kernel="linear")
summary(fit)
```

```
##
## Call:
## svm(formula = Purchase ~ ., data = OJ, cost = 0.01, kernel = "linear",
##     subset = training_indices)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  linear
##        cost:  0.01
##       gamma:  0.05555556
##
## Number of Support Vectors:  432
##
##  ( 215 217 )
```

```
##
##
## Number of Classes:  2
##
## Levels:
##  CH MM
```

From the summary we see that the model was fitted using 432 support vectors.

1.c. We can get the fitted classes from the model object obtained in part (b). Using these, we can calculate the training misclassification error rate.

```
pred <- fit$fitted
actual <- train$Purchase
table(predicted=pred, actual=actual)
```

```
##          actual
## predicted  CH  MM
##        CH 439  78
##        MM  55 228
```

```
mean(pred != actual)
```

```
## [1] 0.16625
```

This model has a training error of 0.16625. For the test error, we predict the classes for the test dataset and again calculate the misclassification error rate.

```
pred <- predict(fit, newdata=test)
actual <- test$Purchase
table(predicted=pred, actual=actual)
```

```
##          actual
## predicted  CH  MM
##        CH 141  31
##        MM  18  80
```

```
mean(pred != actual)
```

```
## [1] 0.1814815
```

This model, applied to the test dataset, has a test error of 0.1814815.

1.d. The `tune()` funcation in the `e1071` package allows us to fit the model using a range of values for hyperparameters. Here we use `tune()` to consider values for cost in the set $\{0.01, 0.05, 0.1, 0.5, 1, 5, 10\}$.

```
fit_tune <- tune(svm, Purchase ~ ., data=train,
                 kernel="linear",
                 ranges=list(cost=c(0.01, 0.05, 0.1, 0.5, 1, 5, 10)))
summary(fit_tune)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost
##  0.05
##
## - best performance: 0.16125
##
## - Detailed performance results:
##    cost   error dispersion
## 1  0.01 0.16625 0.05138701
## 2  0.05 0.16125 0.05318012
## 3  0.10 0.16250 0.04894725
## 4  0.50 0.16500 0.04851976
## 5  1.00 0.16875 0.04723243
## 6  5.00 0.16750 0.05041494
## 7 10.00 0.16500 0.04993051
```

The results suggest that the cost hyperparameter value does not have a large effect on training error. The minimum cost was found for a value of cost = 0.05.

1.e. We can use the optimal value for cost found in part (d) to repeat the analysis from part (c).

```r
best_fit <- fit_tune$best.model
summary(best_fit)
```

```
##
## Call:
## best.tune(method = svm, train.x = Purchase ~ ., data = train,
##     ranges = list(cost = c(0.01, 0.05, 0.1, 0.5, 1, 5, 10)),
##     kernel = "linear")
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  linear
##        cost:  0.05
##       gamma:  0.05555556
##
## Number of Support Vectors:  357
##
##  ( 178 179 )
##
##
## Number of Classes:  2
##
## Levels:
##  CH MM
```

```
pred <- best_fit$fitted
actual <- train$Purchase
table(predicted=pred, actual=actual)
```

```
##          actual
## predicted  CH  MM
##        CH 436  71
##        MM  58 235
```

```
mean(pred != actual)
```

```
## [1] 0.16125
```

With this model the training error is 0.16125. For the test dataset, the test error is given by, which is slightly better than that found for the untuned model (0.1662).

```
pred <- predict(best_fit, newdata=test)
actual <- test$Purchase
table(predicted=pred, actual=actual)
```

```
##          actual
## predicted  CH  MM
##        CH 139  31
##        MM  20  80
```

```
mean(pred != actual)
```

```
## [1] 0.1888889
```

So the test error for cost = 0.05 is 0.1888889. This is slightly worse than that found for the untuned model (0.1815).

1.f. First fit the model to the training data using a radial kernel.

```
fit <- svm(Purchase ~ ., data=OJ, cost=0.01,
           subset=training_indices, kernel="radial")
summary(fit)
```

```
##
## Call:
## svm(formula = Purchase ~ ., data = OJ, cost = 0.01, kernel = "radial",
##     subset = training_indices)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  radial
##        cost:  0.01
##       gamma:  0.05555556
##
```

```
## Number of Support Vectors:  617
##
##  ( 306 311 )
##
##
## Number of Classes:  2
##
## Levels:
##  CH MM
```

From the summary we see that the model was fitted using 617 support vectors.

Next we calculate the training error rate.

```r
pred <- fit$fitted
actual <- train$Purchase
table(predicted=pred, actual=actual)
```

```
##          actual
## predicted  CH  MM
##        CH 494 306
##        MM   0   0
```

```r
mean(pred != actual)
```

```
## [1] 0.3825
```

This model has a training error of 0.3825. For the test error, we predict the classes for the test dataset and again calculate the misclassification error rate.

```r
pred <- predict(fit, newdata=test)
actual <- test$Purchase
table(predicted=pred, actual=actual)
```

```
##          actual
## predicted  CH  MM
##        CH 159 111
##        MM   0   0
```

```r
mean(pred != actual)
```

```
## [1] 0.4111111
```

This model has a test error of 0.4111111.

Next we tune the SVM using the same range of cost values.

```r
fit_tune <- tune(svm, Purchase ~ ., data=train,
                 kernel="radial",
                 ranges=list(cost=c(0.01, 0.05, 0.1, 0.5, 1, 5, 10)))
summary(fit_tune)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost
##   0.5
##
## - best performance: 0.17
##
## - Detailed performance results:
##    cost   error dispersion
## 1  0.01 0.38250 0.06800735
## 2  0.05 0.21250 0.04787136
## 3  0.10 0.18250 0.03917553
## 4  0.50 0.17000 0.05143766
## 5  1.00 0.17625 0.04980866
## 6  5.00 0.17125 0.03955042
## 7 10.00 0.18250 0.03446012
```

Here the cost hyperparameter value has a larger effect on training error than it did for the linear kernel. The minimum cost was found for a value of cost = 0.5.

We can use the optimal value for cost found above to again determin the training and test error.

```
best_fit <- fit_tune$best.model
summary(best_fit)
```

```
##
## Call:
## best.tune(method = svm, train.x = Purchase ~ ., data = train,
##     ranges = list(cost = c(0.01, 0.05, 0.1, 0.5, 1, 5, 10)),
##     kernel = "radial")
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  radial
##        cost:  0.5
##       gamma:  0.05555556
##
## Number of Support Vectors:  406
##
##  ( 202 204 )
##
##
## Number of Classes:  2
##
## Levels:
##  CH MM
```

```r
pred <- best_fit$fitted
actual <- train$Purchase
table(predicted=pred, actual=actual)
```

```
##          actual
## predicted  CH  MM
##        CH 453  77
##        MM  41 229
```

```r
mean(pred != actual)
```

```
## [1] 0.1475
```

The training error is now 0.1475, which is significantly better than that found for the untuned model (0.3825). For the test dataset, the test error is given by

```r
pred <- predict(best_fit, newdata=test)
actual <- test$Purchase
table(predicted=pred, actual=actual)
```

```
##          actual
## predicted  CH  MM
##        CH 143  29
##        MM  16  82
```

```r
mean(pred != actual)
```

```
## [1] 0.1666667
```

So the test error for cost = 0.5 is 0.1666667. This is again significantly than that found for the untuned model (0.4111).

1.g. First fit the model to the training data using a polynomial kernel of degree 2.

```r
fit <- svm(Purchase ~ ., data=OJ, cost=0.01,
           subset=training_indices, kernel="polynomial", degree=2)
summary(fit)
```

```
## 
## Call:
## svm(formula = Purchase ~ ., data = OJ, cost = 0.01, kernel = "polynomial",
##     degree = 2, subset = training_indices)
## 
## 
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  polynomial
##        cost:  0.01
##      degree:  2
##       gamma:  0.05555556
##       coef.0:  0
##
## Number of Support Vectors:  620
##
##  ( 306 314 )
##
##
## Number of Classes:  2
##
## Levels:
##  CH MM
```

From the summary we see that the model was fitted using 620 support vectors.

Next we calculate the training error rate.

```r
pred <- fit$fitted
actual <- train$Purchase
table(predicted=pred, actual=actual)
```

```
##          actual
## predicted  CH  MM
##        CH 494 306
##        MM   0   0
```

```r
mean(pred != actual)
```

```
## [1] 0.3825
```

This model has a training error of 0.3825. For the test error, we predict the classes for the test dataset and again calculate the misclassification error rate.

```r
pred <- predict(fit, newdata=test)
actual <- test$Purchase
table(predicted=pred, actual=actual)
```

```
##          actual
## predicted  CH  MM
##        CH 159 111
##        MM   0   0
```

```r
mean(pred != actual)
```

```
## [1] 0.4111111
```

This model has a test error of 0.4111111.

Next we tune the SVM using the same range of cost values.

```r
fit_tune <- tune(svm, Purchase ~ ., data=train,
                 kernel="polynomial", degree=2,
                 ranges=list(cost=c(0.01, 0.05, 0.1, 0.5, 1, 5, 10)))
summary(fit_tune)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost
##    10
##
## - best performance: 0.17125
##
## - Detailed performance results:
##    cost   error dispersion
## 1  0.01 0.38250 0.03872983
## 2  0.05 0.33250 0.05309844
## 3  0.10 0.33000 0.04257347
## 4  0.50 0.20875 0.03387579
## 5  1.00 0.19750 0.03162278
## 6  5.00 0.17375 0.04016027
## 7 10.00 0.17125 0.03729108
```

Here the cost hyperparameter value has a larger effect on training error than it did for the linear kernel. The minimum cost was found for a value of cost = 10.

We can use the optimal value for cost found above to again determin the training and test error.

```r
best_fit <- fit_tune$best.model
summary(best_fit)
```

```
##
## Call:
## best.tune(method = svm, train.x = Purchase ~ ., data = train,
##     ranges = list(cost = c(0.01, 0.05, 0.1, 0.5, 1, 5, 10)),
##     kernel = "polynomial", degree = 2)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  polynomial
##        cost:  10
##      degree:  2
##       gamma:  0.05555556
##      coef.0:  0
##
## Number of Support Vectors:  342
##
##  ( 170 172 )
##
```

```
##
## Number of Classes:  2
##
## Levels:
##  CH MM
```

```
pred <- best_fit$fitted
actual <- train$Purchase
table(predicted=pred, actual=actual)
```

```
##          actual
## predicted  CH  MM
##        CH 450  72
##        MM  44 234
```

```
mean(pred != actual)
```

```
## [1] 0.145
```

The training error is now 0.145, which is significantly better than that found for the untuned model (0.3825). For the test dataset, the test error is given by

```
pred <- predict(best_fit, newdata=test)
actual <- test$Purchase
table(predicted=pred, actual=actual)
```

```
##          actual
## predicted  CH  MM
##        CH 140  31
##        MM  19  80
```

```
mean(pred != actual)
```

```
## [1] 0.1851852
```

So the test error for cost = 10 is 0.1851852. This is again significantly than that found for the untuned model (0.4111).

1.h. From the above we see that the best performing model is the support vector classifier with a linear kernel and cost = 0.05. The test error for this model is 0.1613, compared to test error of 0.1667 for the support vector machine with a radial kernel and cost = 0.5, and test error of 0.1852 for the support vector machine with a polynomial kernel of degree 2 and cost = 10. Note that the best model fit as determined by training error was found for the polynomial kernel of degree 2 and cost = 10 (0.145).

```
# clean up before moving on to next question
rm(list = ls())
```

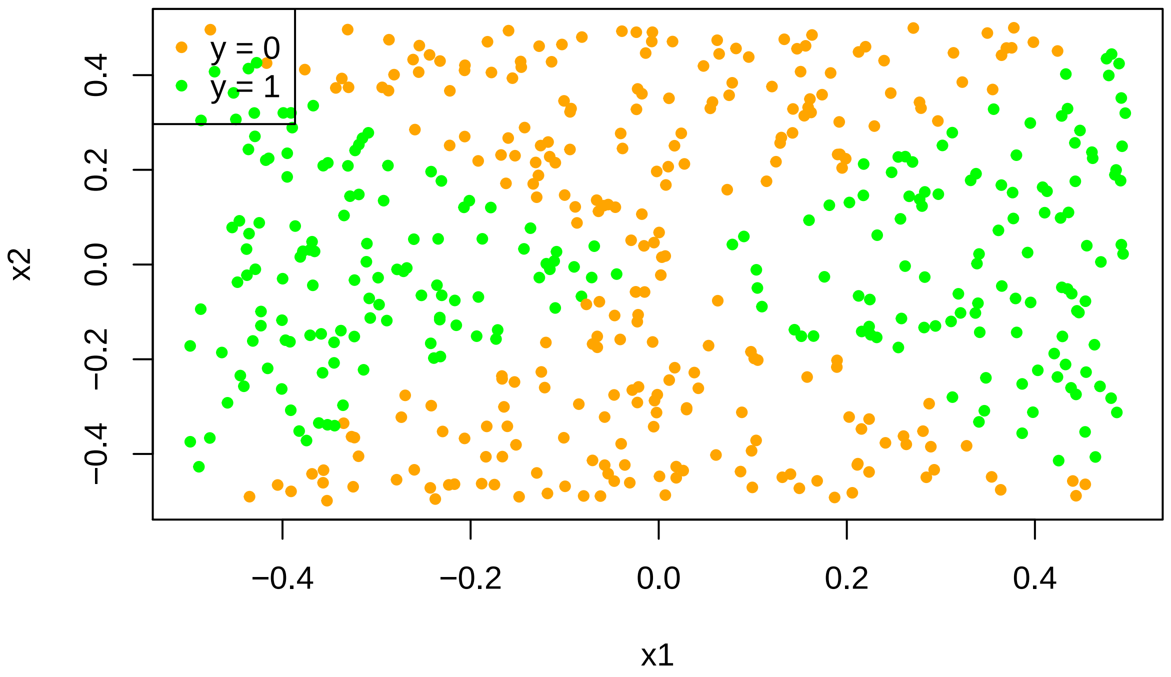
2. SVM and Logistic Regression (Textbook 9.7.5)

2.a. First we set a seed for the session and then generate the training data by making draws from the uniform distribution.

```r
set.seed(1)
x1 <- runif(500) - 0.5
x2 <- runif(500) - 0.5
y <- 1 * (x1^2 - x2^2 > 0)
```

2.b. We can plot a scatterplot of the predictor variables using the response variable categories to colour the individual data points.

```r
plot(x1, x2, xlab="x1", ylab="x2",
     pch=20, cex=1, col=ifelse(y == 0, "orange", "green"))
legend(x="topleft", legend=c("y = 0", "y = 1"), col=c("orange", "green"),
       pch=20, cex=1)
```



2.c. To fit a logistic regression model, we use the `glm` function with "binomial" as the value for the `family` parameter.

```r
lr_fit <- glm(y ~ x1 + x2, family="binomial")
summary(lr_fit)
```

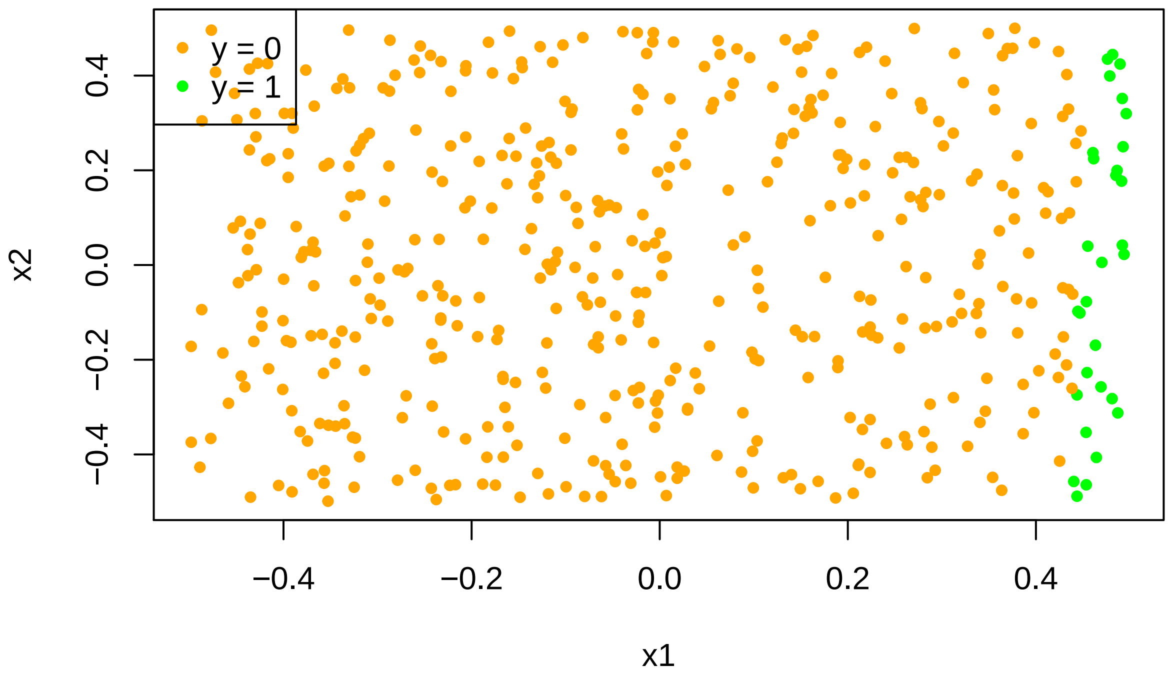
```
##
## Call:
## glm(formula = y ~ x1 + x2, family = "binomial")
##
## Deviance Residuals:
##    Min      1Q  Median      3Q     Max
## -1.179  -1.139  -1.112   1.206   1.257
##
## Coefficients:
##              Estimate Std. Error z value Pr(>|z|)
## (Intercept) -0.087260   0.089579  -0.974    0.330
## x1           0.196199   0.316864   0.619    0.536
## x2          -0.002854   0.305712  -0.009    0.993
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 692.18  on 499  degrees of freedom
## Residual deviance: 691.79  on 497  degrees of freedom
## AIC: 697.79
##
## Number of Fisher Scoring iterations: 3
```

2.d. Using the model obtained in part (c), we can predict the log-odds response and then assign a value of 1 to those observations for which the predicted probability is greater than 0.5.

```
probs <- predict(lr_fit, type="response")
preds <- rep(0, length(y))
preds[probs > 0.5] <- 1
```

Using these predictions, we can again plot the data, this time using the predicted labels to colour the training observations.

```
plot(x1, x2, xlab="x1", ylab="x2",
     pch=20, cex=1, col=ifelse(preds == 0, "orange", "green"))
legend(x="topleft", legend=c("y = 0", "y = 1"), col=c("orange", "green"),
       pch=20, cex=1)
```



We see that the logistic regression model has yielded a linear decision boundary. To see how the model has performed on the training data, we can plot a confusion matrix.

```
table(predicted=preds, actual=y)
```

```
##          actual
## predicted   0   1
##         0 258 212
##         1   3  27
```

```
mean(preds != y)
```

```
## [1] 0.43
```

We see that using logistic regression and a linear combination of the predictors, we have a misclassification error rate of 0.43 for the training data.

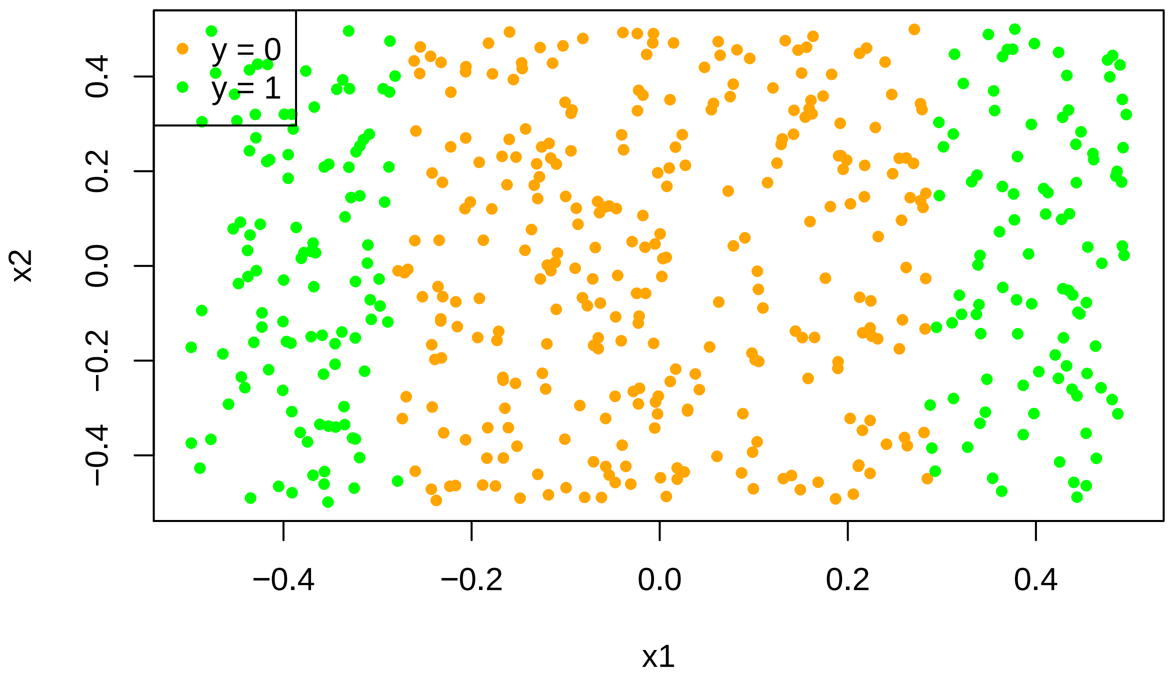
2.e-f. To attempt to obtain a non-linear decision boundary, we can transform the predictors and refit the model as above. Here we use a quadratic term based on `x1`.

```
lr_fit <- glm(y ~ poly(x1, 2, raw=TRUE), family="binomial")
summary(lr_fit)
```

```
##
## Call:
## glm(formula = y ~ poly(x1, 2, raw = TRUE), family = "binomial")
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.6484  -0.6916  -0.5477   0.7176   1.9632
##
## Coefficients:
##                          Estimate Std. Error z value Pr(>|z|)
## (Intercept)               -1.8223     0.1783 -10.220   <2e-16 ***
## poly(x1, 2, raw = TRUE)1  -0.1451     0.4349  -0.334    0.739
## poly(x1, 2, raw = TRUE)2  23.0165     2.0369  11.300   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 692.18  on 499  degrees of freedom
## Residual deviance: 482.70  on 497  degrees of freedom
## AIC: 488.7
##
## Number of Fisher Scoring iterations: 5
```

```
probs <- predict(lr_fit, type="response")
preds1 <- rep(0, length(y))
preds1[probs > 0.5] <- 1

plot(x1, x2, xlab="x1", ylab="x2",
     pch=20, cex=1, col=ifelse(preds1 == 0, "orange", "green"))
legend(x="topleft", legend=c("y = 0", "y = 1"), col=c("orange", "green"),
       pch=20, cex=1)
```



```r
table(predicted=preds1, actual=y)
```

```
##          actual
## predicted   0   1
##         0 217  73
##         1  44 166
```

```r
mean(preds1 != y)
```

```
## [1] 0.234
```

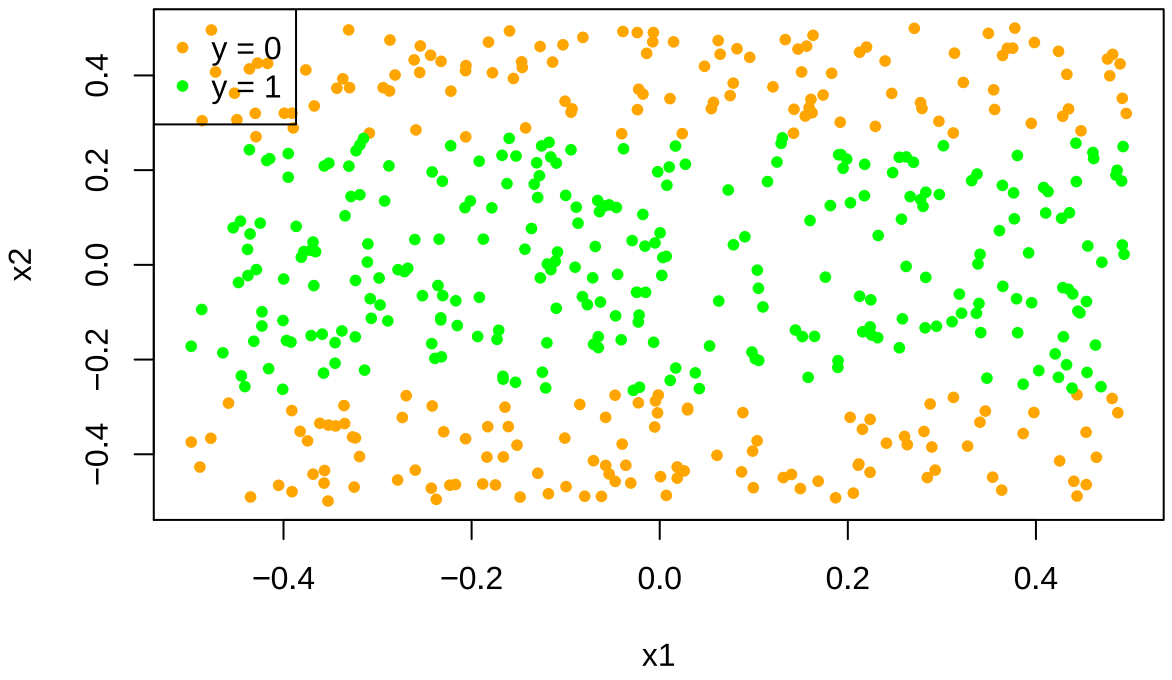
Next we generate a logistic regression model using a quadratic term based on `x2`.

```r
lr_fit <- glm(y ~ poly(x2, 2, raw=TRUE), family="binomial")
summary(lr_fit)
```

```
##
## Call:
## glm(formula = y ~ poly(x2, 2, raw = TRUE), family = "binomial")
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -1.8408  -0.7699  -0.2486   0.7684   2.2999
##
## Coefficients:
##                           Estimate Std. Error z value Pr(>|z|)
## (Intercept)               1.496166   0.166428   8.990   <2e-16 ***
## poly(x2, 2, raw = TRUE)1  0.005988   0.428505   0.014    0.989
## poly(x2, 2, raw = TRUE)2 -20.607640   1.910114 -10.789   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 692.18  on 499  degrees of freedom
## Residual deviance: 503.37  on 497  degrees of freedom
## AIC: 509.37
##
## Number of Fisher Scoring iterations: 5
```

```
probs <- predict(lr_fit, type="response")
preds2 <- rep(0, length(y))
preds2[probs > 0.5] <- 1

plot(x1, x2, xlab="x1", ylab="x2",
     pch=20, cex=1, col=ifelse(preds2 == 0, "orange", "green"))
legend(x="topleft", legend=c("y = 0", "y = 1"), col=c("orange", "green"),
       pch=20, cex=1)
```



```
table(predicted=preds2, actual=y)
```

```
##          actual
## predicted   0   1
##         0 182  48
##         1  79 191
```

```
mean(preds2 != y)
```

```
## [1] 0.254
```

For both, the model produces a non-linear decision boundary and has improved performance on the training data, with misclassifcation error rates of 0.234 and 0.254.

Note that while we know that our model contains quadratic terms based on `x1` and `x2`, for technical reasons we cannot use a model containing both quadratic terms. The `glm` package produces errors when the predicted probabilities are very close to 0 or 1, which could be expected for our data. See this StackOverflow thread for more information.

2.g. Next we fit a support vector classifier (i.e. a support vector machine with a linear kernel) to the data. The `svm()` function is in the `e1071` package. First we create a dataframe to hold `y`, `x1` and `x2`.

```r
library(e1071)
df <- data.frame(y=as.factor(y), x1=x1, x2=x2)
```

Next, since we do not have a sense of what level of cost is appropriate for fitting the support vector classifier model, we use the `tune()` function to try a range of cost values.

```r
svc_tune <- tune(svm, y ~ ., data=df, kernel="linear", scale=FALSE,
                 ranges=list(cost=c(0.01, 0.1, 1, 10, 100)))
summary(svc_tune)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost
##  0.01
##
## - best performance: 0.478
##
## - Detailed performance results:
##    cost error dispersion
## 1 1e-02 0.478 0.06285786
## 2 1e-01 0.478 0.06285786
## 3 1e+00 0.478 0.06285786
## 4 1e+01 0.478 0.06285786
## 5 1e+02 0.478 0.06285786
```

From the tuning we see that the best value for cost is 0.01. We now fit the support vector classifier using this hyperparameter value, then make predictions and plot the data again, using the predicted labels to specify the colours.

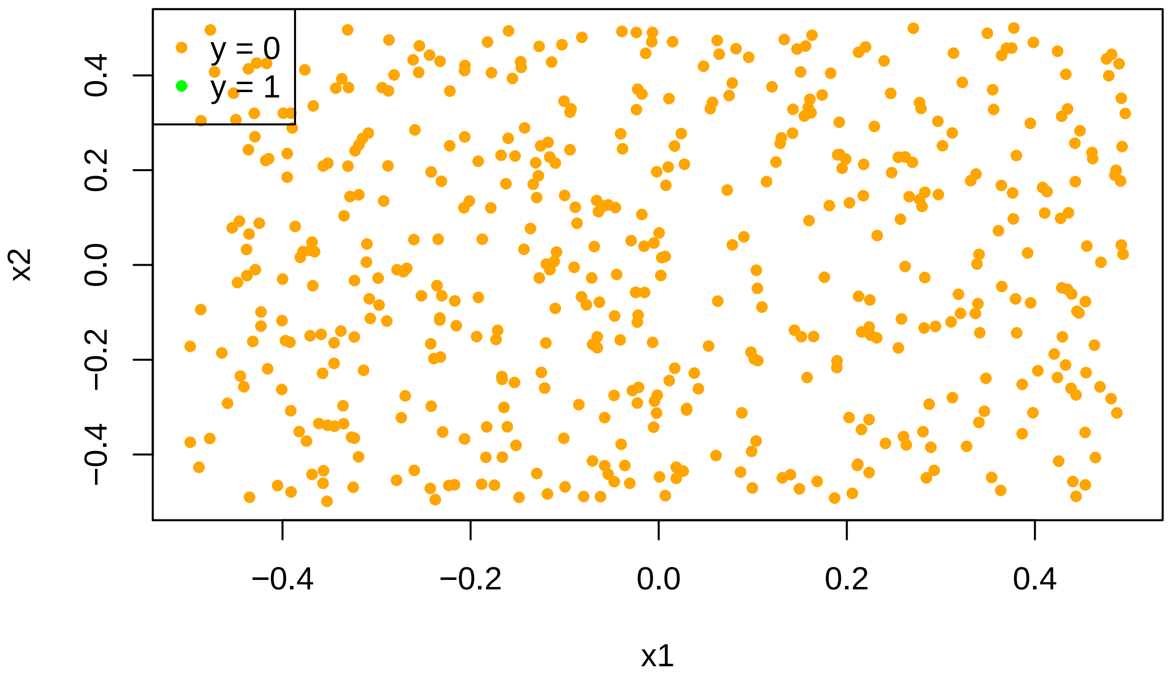
```r
c <- svc_tune$best.parameters$cost
svc_fit <- svm(y ~ ., data=df, scale=FALSE, cost=c, kernel="linear")
summary(svc_fit)
```

```
##
## Call:
## svm(formula = y ~ ., data = df, cost = c, kernel = "linear",
##     scale = FALSE)
##
```

```
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  linear
##        cost:  0.01
##       gamma:  0.5
##
## Number of Support Vectors:  479
##
##  ( 239 240 )
##
##
## Number of Classes:  2
##
## Levels:
##  0 1
```

```
svc_preds <- predict(svc_fit, df, scale=FALSE)

plot(x1, x2, xlab="x1", ylab="x2",
     pch=20, cex=1, col=ifelse(svc_preds == 0, "orange", "green"))
legend(x="topleft", legend=c("y = 0", "y = 1"), col=c("orange", "green"),
       pch=20, cex=1)
```



```
table(predicted=svc_preds, actual=y)
```

```
##          actual
## predicted   0   1
##         0 261 239
##         1   0   0
```

```r
mean(svc_preds != y)
```

```
## [1] 0.478
```

The support vector classifier, using a linear kernel, predicted a label of 0 for all values and so has a misclassification error rate of 0.478 on the training data.

2.h. Given that the data was generated using quadratic features, it is perhaps no surprise that the linear kernel was not able to perform well. Therefore we now retry the analysis using a non-linear kernel.

Starting with the polynomial kernel, we can again using the `tune()` function to try a range of degrees and cost values.

```r
svm_tune <- tune(svm, y ~ ., data=df, kernel="polynomial", scale=FALSE,
                 ranges=list(cost=c(0.01, 0.1, 1, 10, 100, 200, 500),
                             degree=c(2, 3)))
summary(svm_tune)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost degree
##   100      2
##
## - best performance: 0.028
##
## - Detailed performance results:
##     cost degree error dispersion
## 1  1e-02      2 0.478 0.05116422
## 2  1e-01      2 0.478 0.05116422
## 3  1e+00      2 0.116 0.06653320
## 4  1e+01      2 0.050 0.03431877
## 5  1e+02      2 0.028 0.02347576
## 6  2e+02      2 0.028 0.02859681
## 7  5e+02      2 0.028 0.03011091
## 8  1e-02      3 0.478 0.05116422
## 9  1e-01      3 0.478 0.05116422
## 10 1e+00      3 0.478 0.05116422
## 11 1e+01      3 0.478 0.05116422
## 12 1e+02      3 0.478 0.05116422
## 13 2e+02      3 0.472 0.05266245
## 14 5e+02      3 0.424 0.08044322
```

From the model tuning, the best parameter combination is given by

```r
svm_tune$best.parameters
```

```
##   cost degree
## 5  100      2
```

```r
c <- svm_tune$best.parameters$cost
d <- svm_tune$best.parameters$degree
```

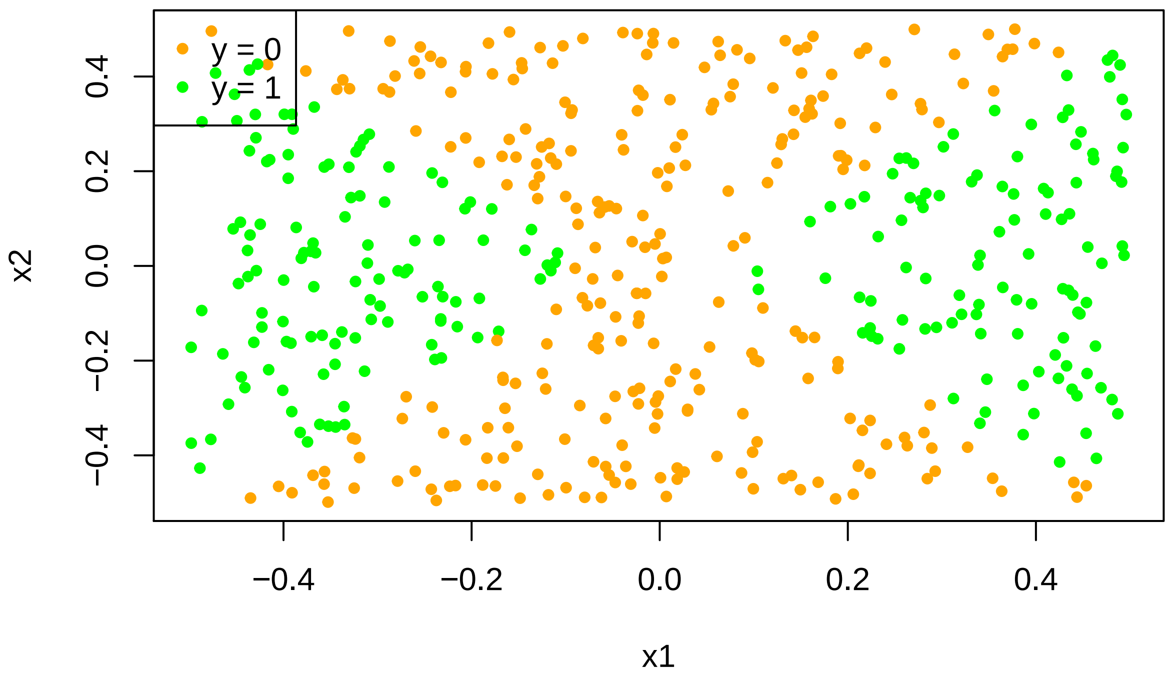
So we now fit the support vector machine with a polynomial kernel with degree 2 and cost 100.

```r
svm_fit <- svm(y ~ ., data=df, scale=FALSE, cost=c,
               kernel="polynomial", degree=d)
summary(svm_fit)
```

```
##
## Call:
## svm(formula = y ~ ., data = df, cost = c, kernel = "polynomial",
##     degree = d, scale = FALSE)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  polynomial
##        cost:  100
##      degree:  2
##       gamma:  0.5
##      coef.0:  0
##
## Number of Support Vectors:  146
##
##  ( 72 74 )
##
##
## Number of Classes:  2
##
## Levels:
##  0 1
```

```r
svm_preds <- predict(svm_fit, df, scale=FALSE)

plot(x1, x2, xlab="x1", ylab="x2",
     pch=20, cex=1, col=ifelse(svm_preds == 0, "orange", "green"))
legend(x="topleft", legend=c("y = 0", "y = 1"), col=c("orange", "green"),
       pch=20, cex=1)
```



```r
table(predicted=svm_preds, actual=y)
```

```
##          actual
## predicted   0   1
##         0 260  14
##         1   1 225
```

```r
mean(svm_preds != y)
```

```
## [1] 0.03
```

This tuned model has a misclassification error rate of 0.03.

2.i. We see that using a tuned support vector machine our model has a misclassification error rate of just 0.03 on the training data, vastly outperforming the support vector classifier model and the logistic regression models we were able to fit using the `glm` package.

Of course, it should be noted that these results are for the training data set, which may have overfitting to the training data.

```r
# clean up before moving on to next question
rm(list = ls())
```

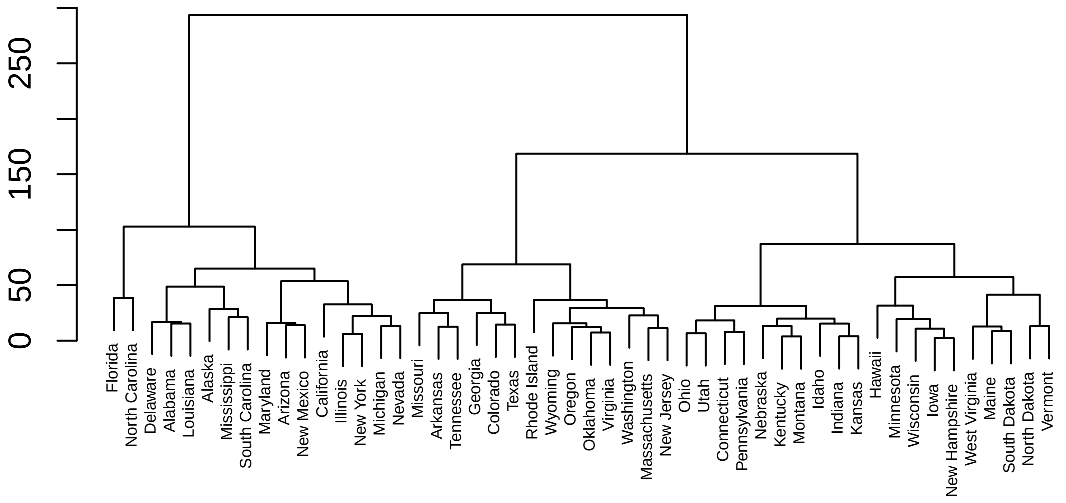
3. Hierarchical Clustering (Textbook 10.7.9)

3.a. First, load the `ISLR` package to access the `USArrests` dataset, and set the seed for the session.

```r
library(ISLR)
set.seed(1)
```

Next we use the `hclust()` method using the `dist()` function for Euclidean distance and specifying the linkage as complete. The dendrogram is given below.

```r
hfit <- hclust(d=dist(USArrests), method="complete")
plot(hfit, xlab="", ylab="", main="", sub="", cex=0.5)
```



3.b. To cut the dendrogram, we use the `cutree()` function, specifying that we want three clusters by setting the `k` parameter.

```r
cut_fit <- cutree(hfit, k=3)
```

The 16 states in the first cluster are

```r
names(cut_fit[cut_fit == 1])
```

```
##  [1] "Alabama"        "Alaska"         "Arizona"        "California"
##  [5] "Delaware"       "Florida"        "Illinois"       "Louisiana"
##  [9] "Maryland"       "Michigan"       "Mississippi"    "Nevada"
## [13] "New Mexico"     "New York"       "North Carolina" "South Carolina"
```

The 14 states in the second cluster are

```r
names(cut_fit[cut_fit == 2])
```

```
##  [1] "Arkansas"      "Colorado"      "Georgia"       "Massachusetts"
##  [5] "Missouri"      "New Jersey"    "Oklahoma"      "Oregon"
##  [9] "Rhode Island"  "Tennessee"     "Texas"         "Virginia"
## [13] "Washington"    "Wyoming"
```

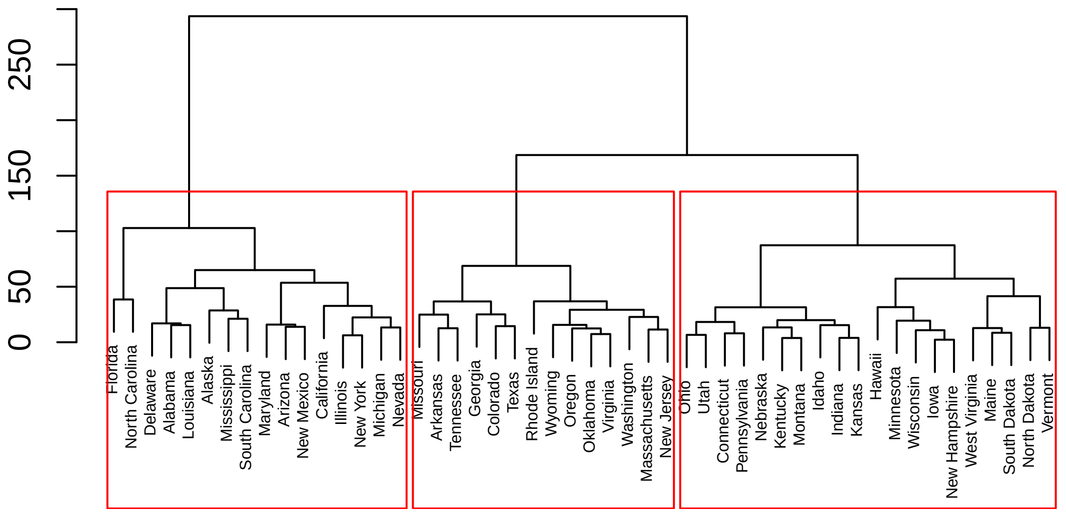
The 20 states in the third cluster are

```r
names(cut_fit[cut_fit == 3])
```

```
##  [1] "Connecticut"   "Hawaii"        "Idaho"         "Indiana"
##  [5] "Iowa"          "Kansas"        "Kentucky"      "Maine"
##  [9] "Minnesota"     "Montana"       "Nebraska"      "New Hampshire"
## [13] "North Dakota"  "Ohio"          "Pennsylvania"  "South Dakota"
## [17] "Utah"          "Vermont"       "West Virginia" "Wisconsin"
```

We can visualize the clustering by using the `rect.hclust()` function.

```
hfit <- hclust(d=dist(USArrests), method="complete")
plot(hfit, xlab="", ylab="", main="", sub="", cex=0.5)
rect.hclust(hfit, 3)
```



3.c. To scale the data in the `USArrests` dataset we can use the `scale()` function.

```
scaled <- scale(USArrests)
summary(USArrests)
```

```
##      Murder          Assault         UrbanPop          Rape      
##  Min.   : 0.800   Min.   : 45.0   Min.   :32.00   Min.   : 7.30  
##  1st Qu.: 4.075   1st Qu.:109.0   1st Qu.:54.50   1st Qu.:15.07  
##  Median : 7.250   Median :159.0   Median :66.00   Median :20.10  
##  Mean   : 7.788   Mean   :170.8   Mean   :65.54   Mean   :21.23  
##  3rd Qu.:11.250   3rd Qu.:249.0   3rd Qu.:77.75   3rd Qu.:26.18  
##  Max.   :17.400   Max.   :337.0   Max.   :91.00   Max.   :46.00
```

```
var(USArrests)
```

```
##              Murder   Assault   UrbanPop      Rape
## Murder    18.970465  291.0624   4.386204  22.99141
## Assault  291.062367 6945.1657 312.275102 519.26906
## UrbanPop   4.386204  312.2751 209.518776  55.76808
## Rape      22.991412  519.2691  55.768082  87.72916
```

```
summary(scaled)
```

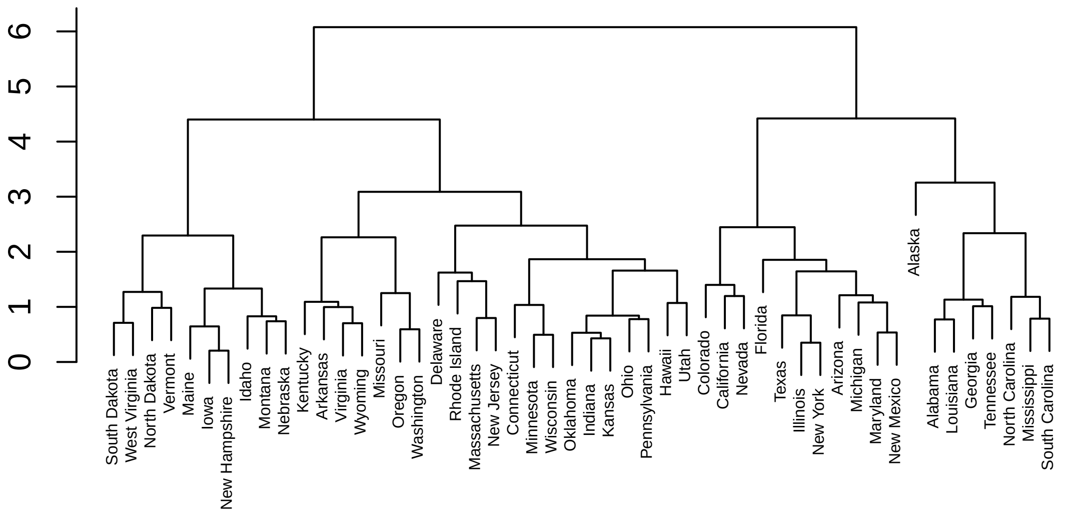
```
##      Murder            Assault            UrbanPop              Rape        
##  Min.   :-1.6044   Min.   :-1.5090   Min.   :-2.31714   Min.   :-1.4874  
##  1st Qu.:-0.8525   1st Qu.:-0.7411   1st Qu.:-0.76271   1st Qu.:-0.6574  
##  Median :-0.1235   Median :-0.1411   Median : 0.03178   Median :-0.1209  
##  Mean   : 0.0000   Mean   : 0.0000   Mean   : 0.00000   Mean   : 0.0000  
##  3rd Qu.: 0.7949   3rd Qu.: 0.9388   3rd Qu.: 0.84354   3rd Qu.: 0.5277  
##  Max.   : 2.2069   Max.   : 1.9948   Max.   : 1.75892   Max.   : 2.6444
```

```
var(scaled)
```

```
##              Murder  Assault   UrbanPop      Rape
## Murder   1.00000000 0.8018733 0.06957262 0.5635788
## Assault  0.80187331 1.0000000 0.25887170 0.6652412
## UrbanPop 0.06957262 0.2588717 1.00000000 0.4113412
## Rape     0.56357883 0.6652412 0.41134124 1.0000000
```

If we now perform clustering as before, we get the following.

```
hfit2 <- hclust(d=dist(scaled), method="complete")
plot(hfit2, xlab="", ylab="", main="", sub="", cex=0.5)
```



3.d. By introducing scaling, we have effectively specified that the magnitude and spread of the parameters is of less importance. In the USArrests dataset, the three categories of violent crimes considered are `Murder`, `Assault` and `Rape`, each of which is reported as a the rate per 100,000 people. `UrbanPop` on the other hand reports the percent of the population that lives in an urban environment.

While scaling is important for principal components analysis when there are different units and variances, as is the case here, that may not necessarily mean that we need to scale the data here.

Given that the three categories of violent crime rates are already expressed per 100,000 people, it may not make sense to scale the data, since it may be useful to know which states have greater dispersion in these variables thereby indicating a more heterogeneous population. Keeping the variables unscaled may also allow us to determine similarity based on the three violent crimes separately rather than considering them to be equivalent.

```
# clean up before moving on to next question
rm(list = ls())
```

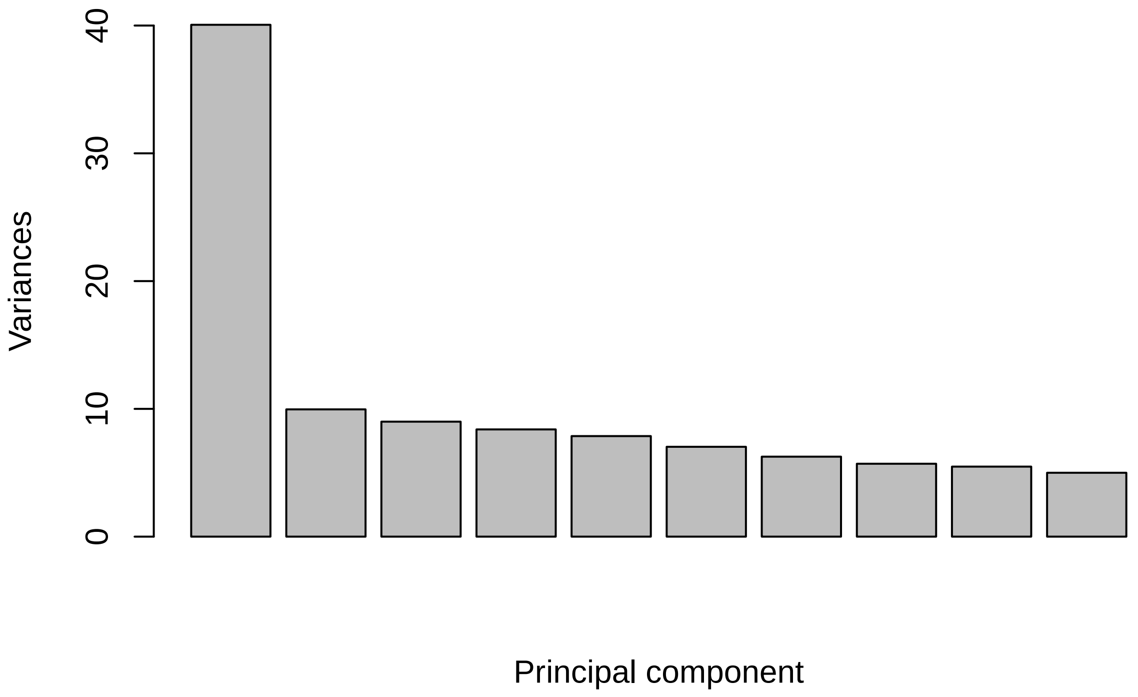
4. PCA and K-Means Clustering (Textbook 10.7.10)

4.a. We can generate the data by making draws from the normal distribution, with a mean shift for the second and third groups to allow separation. We also create a vector of colour labels to use in subsequent plots and a vector holding the original groupings.

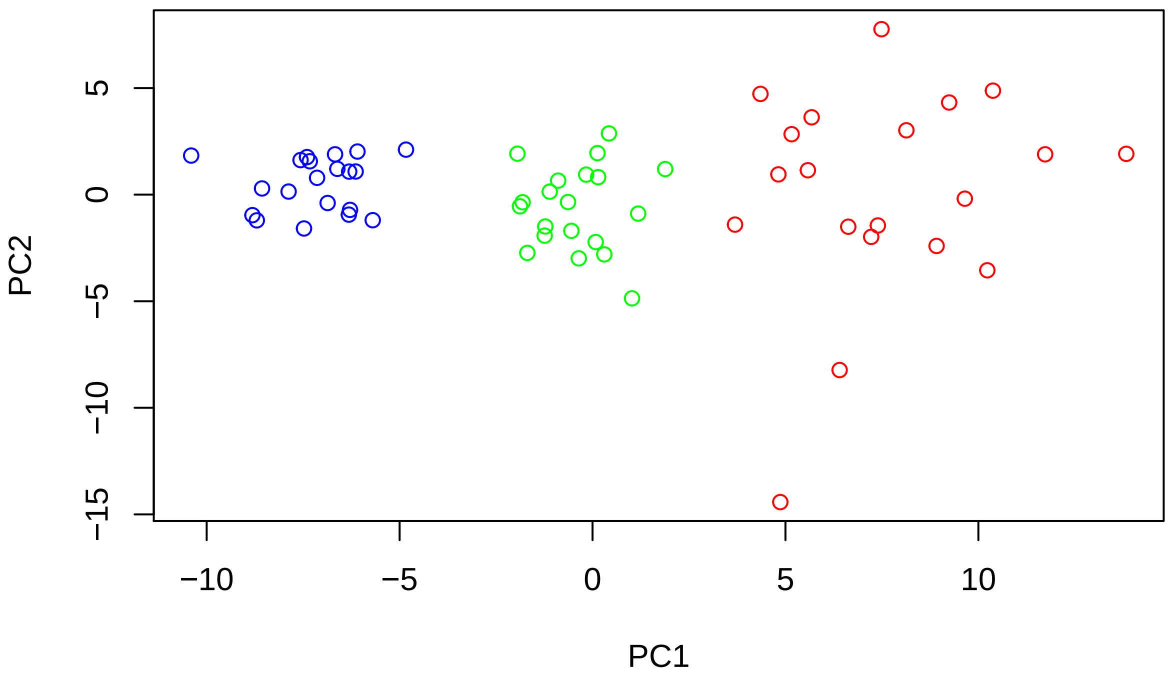
```r
set.seed(1)
g1 <- matrix(rnorm(20*50, mean=0, sd=1), ncol=50)
g2 <- matrix(rnorm(20*50, mean=1, sd=sqrt(2)), ncol=50)
g3 <- matrix(rnorm(20*50, mean=2, sd=2), ncol=50)
df <- rbind(g1, g2, g3)
grp <- c(rep(1, 20), rep(2, 20), rep(3, 20))
colours <- c(rep("blue", 20), rep("green", 20), rep("red", 20))
```

4.b. Next we compute the principal components for the simulated dataset and plot a scree plot, showing the variance explained by each of the principal components, and also a scatter plot using the first two principal components.

```r
pc <- prcomp(df)
plot(pc, xlab="Principal component", ylim=c(0, 40), main="")
```



```r
plot(pc$x[, 1], pc$x[, 2], col=colours, xlab="PC1", ylab="PC2")
```



The scatter plot shows that the groups are separable by the first two principal components.

4.c. To perform k-means clustering, we use the `kmeans()` function and set the parameter `centers` = 3. Note that for reproducibility we set the seed, since the observations are randomly assigned to clusters to begin with.

```
set.seed(1)
km_fit <- kmeans(df, centers=3)
table(predicted=km_fit$cluster, actual=grp)
```

```
##          actual
## predicted  1  2  3
##         1 20  0  0
##         2  0 20  4
##         3  0  0 16
```

From the table we see that the algorithm was able to identify three distinct groups. There appear to be 4 observations that have been misclassified using this approach.

4.d. For two clusters, we repeat the process in part (c) but set `centers` = 2.

```
set.seed(1)
km_fit <- kmeans(df, centers=2)
table(predicted=km_fit$cluster, actual=grp)
```

```
##          actual
## predicted  1  2  3
##         1 20 19  0
##         2  0  1 20
```

The results show that two clusters have been identified, and that the majority of the second set of 20 observations have been classified in the same cluster as those from the first 20 observations. Only 1 of the second group has been classified together with the third group.

4.e. For four clusters, we repeat the process in part (c) but set `centers` = 4.

```r
set.seed(1)
km_fit <- kmeans(df, centers=4)
table(predicted=km_fit$cluster, actual=grp)
```

```
##          actual
## predicted  1  2  3
##         1 20  0  0
##         2  0 20  3
##         3  0  0 10
##         4  0  0  7
```

This time the results show that the observations from the third group of observations have been spread across three different clusters, while the observations from the first two groups have been clustered together as before.

4.f. First we create a new dataset containg the first two principal components. We then use this new data and set `centers` = 3.

```r
pcs <- pc$x[, c(1, 2)]
km_fit <- kmeans(pcs, centers=3)
table(predicted=km_fit$cluster, actual=grp)
```

```
##          actual
## predicted  1  2  3
##         1 20 14  0
##         2  0  0 12
##         3  0  6  8
```

The results show that the observations from the second and third groups of observations are now less well separated than they were when using the full dataset.

From the scree plot in part (b) we see that the first two principal component eigenvectors capure a large amount of the variation in the data but not all of it. Several more principal components also capture variation in the data, and it is this loss of information that is likely resulting in the separation being less efficient in this scenario than when using the full dataset.

4.g. We start by scaling the data using the `scale()` function, and then performing k-means clusetring with three centres as before.

```r
df_scaled <- scale(df)
km_fit <- kmeans(df_scaled, centers=3)
table(predicted=km_fit$cluster, actual=grp)
```

```
##          actual
## predicted  1  2  3
##         1  0  0 16
##         2  0 20  4
##         3 20  0  0
```

Comparing the table with that obtained in part (c) we see that scaling does not appear to have had an effect on the ability of k-means to assign the points to the three clusters. This may be down to the fact that the difference in the means in the three simulated groups ($\mu = 0, 1, 2$) is large enough to account for the relatively small differences in variances (sd $= 1, \sqrt{2}, 2$).