Homework Assignment 5

1. Consider the Gini index, classification error, and cross-entropy in a simple classification setting with two classes. Create a single plot that displays each of these quantities as a function of p^ m1. The X-axis should display p^ m1, ranging from 0 to 1, and the y-axis should display the value of the Gini index, classification error, and entropy.

Definitions:

- `pmk`: Proportion of training observations in the m^{th} region from the k^{th} class.
- In scenarios with two classes (`K=2`):
  - `pm1 = 1 - pm2`

Classification Error (E):

Classification error is calculated as the complement of the maximum proportion of class observations within a region:

- **When `1 > pm2 > 0.5`** (Class 1 is most common):
  - `E = 1 - pm1`
- **When `0.5 > pm2 > 0`** (Class 1 is least common):
  - `E = 1 - pm2`
  - Simplifies to `E = 1 - (1 - pm1)`

Gini Index:

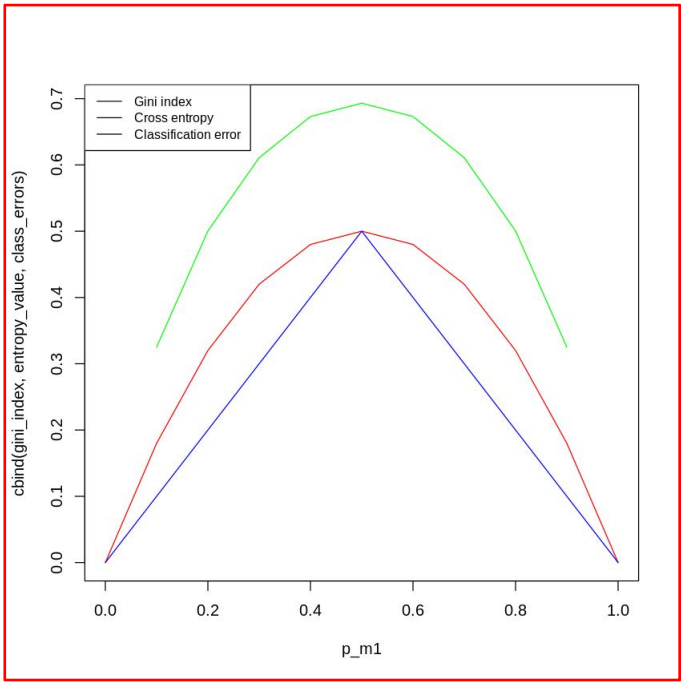
The Gini index for a binary classification scenario is calculated as:

- `G = Summation from k=1 to K of (pmk * (1 - pmk))`
- For `pm1`, this formula simplifies to:
  - `G = 2 * pm1 * (1 - pm1)`

Cross Entropy (D):

Cross entropy, particularly binary cross entropy for two classes, is defined as:

- `D = -1 * summation of k=1 to K of (pmk * log(pmk))`
- For `pm1`, this becomes:
  - `D = -1 * (pm1 * log(pm1) + (1 - pm1) * log(1 - pm1)) (this is binary cross entropy)`



2. This problem involves the OJ data set which is part of the ISLR package.

a. Create a training set containing a random sample of 800 observations, and a test set containing the remaining observations.

```
8.4 Q9

[ ]  1  install.packages(c("ISLR","MASS","tree"))
     2  library(ISLR)
     3  library(MASS)
     4  library(tree)
     5  set.seed(123)

   Installing packages into '/usr/local/lib/R/site-library'
   (as 'lib' is unspecified)

[ ]  1  sum(is.na(OJ))

   0

[ ]  1  names(OJ)

   'Purchase' · 'WeekofPurchase' · 'StoreID' · 'PriceCH' · 'PriceMM' · 'DiscCH' · 'DiscMM' · 'SpecialCH' · 'SpecialMM' · 'LoyalCH' · 'SalePriceMM' · 'SalePriceCH' · 'PriceDiff' · 'Store7' · 'PctDiscMM' · 'PctDiscCH' · 'ListPriceDiff' · 'STORE'

(a)

[ ]  1  test_index=sample(nrow(OJ),270,replace=FALSE)
     2  train_set=OJ[-test_index,]
     3  test_set=OJ[test_index,]
     4  print(c(nrow(train_set),nrow(test_set),nrow(train_set)+nrow(test_set),nrow(OJ)))

   [1]  800  270 1070 1070
```

b. Fit a tree to the training data, with Purchase as the response and the other variables except for Buy as predictors. Use the summary () function to produce summary statistics about the tree, and describe the results obtained. What is the training error rate? How many terminal nodes does the tree have?

```
[ ]  1  tree_fit_oj=tree(Purchase~.,data=train_set)

[ ]  1  summary(tree_fit_oj)

    Classification tree:
    tree(formula = Purchase ~ ., data = train_set)
    Variables actually used in tree construction:
    [1] "LoyalCH"       "PriceDiff"     "SalePriceMM"   "ListPriceDiff"
    [5] "StoreID"
    Number of terminal nodes:  8
    Residual mean deviance:  0.7304 = 578.5 / 792
    Misclassification error rate: 0.1725 = 138 / 800
```

The training error rate of 0.1725 indicates that approximately 18 out of every 100 purchases in our training data are incorrectly predicted by our model, and there are 8 terminal nodes. Our model includes 8 distinct groups that significantly differ from each other. These groups are instrumental in predicting whether a person will purchase orange juice. Additionally, the residual mean deviance is a critical measure of how well our model fits the data. A high residual mean deviance suggests that the model does not adequately fit the data, leaving a significant amount of information unexplained. This unreliability in the model's fit compromises the accuracy of our predictions.

c. Type in the name of the tree object in order to get a detailed text output. Pick one of the terminal nodes, and interpret the information displayed.

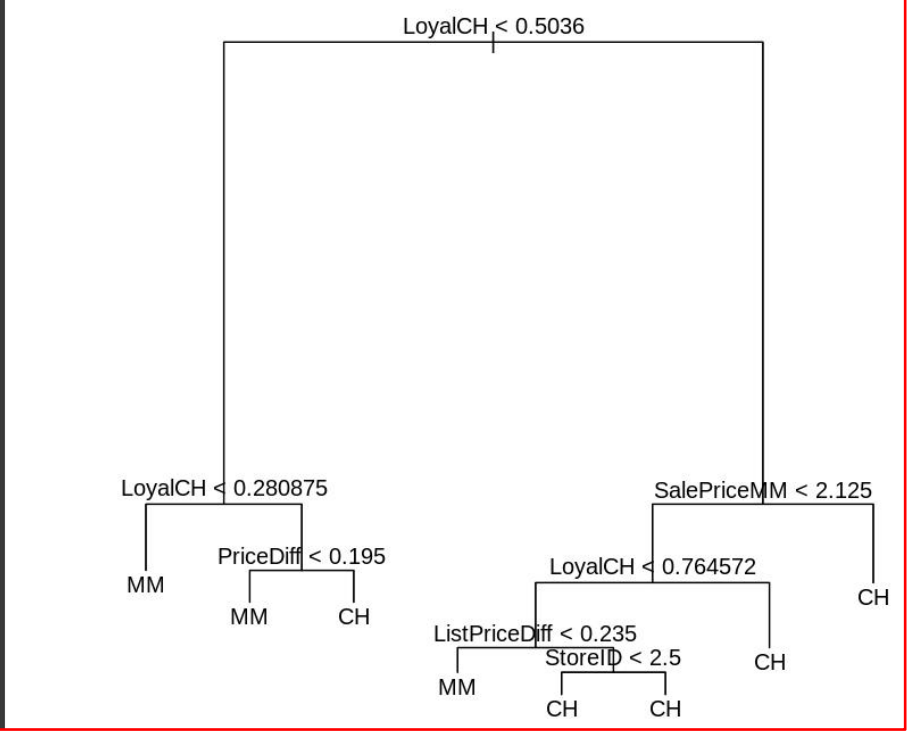
```
1  tree_fit_oj
```

```
node), split, n, deviance, yval, (yprob)
      * denotes terminal node

 1) root 800 1069.00 CH ( 0.61125 0.38875 )
   2) LoyalCH < 0.5036 346  408.70 MM ( 0.27746 0.72254 )
     4) LoyalCH < 0.280875 156  107.40 MM ( 0.10897 0.89103 ) *
     5) LoyalCH > 0.280875 190  258.00 MM ( 0.41579 0.58421 )
      10) PriceDiff < 0.195 82   88.78 MM ( 0.23171 0.76829 ) *
      11) PriceDiff > 0.195 108  148.40 CH ( 0.55556 0.44444 ) *
   3) LoyalCH > 0.5036 454  358.30 CH ( 0.86564 0.13436 )
     6) SalePriceMM < 2.125 260  276.00 CH ( 0.77692 0.22308 )
      12) LoyalCH < 0.764572 125  167.40 CH ( 0.60800 0.39200 )
        24) ListPriceDiff < 0.235 67   92.15 MM ( 0.44776 0.55224 ) *
        25) ListPriceDiff > 0.235 58   59.14 CH ( 0.79310 0.20690 )
          50) StoreID < 2.5 21   29.06 CH ( 0.52381 0.47619 ) *
          51) StoreID > 2.5 37   15.56 CH ( 0.94595 0.05405 ) *
      13) LoyalCH > 0.764572 135   66.13 CH ( 0.93333 0.06667 ) *
     7) SalePriceMM > 2.125 194   30.97 CH ( 0.98454 0.01546 ) *
```

Branch 5 results in a terminal node *"5) LoyalCH > 0.280875 190 258.00 MM (0.41579 0.58421)"*
The decision tree output refers to a terminal node in a decision tree model. The split condition *LoyalCH > 0.280875* indicates that this node includes observations where loyalty scores exceed 0.280875. The number *190* represents the count of observations in this node. The figure *258.00 MM* likely refers to a measure of error or fit for this node, hinting at how predictions match actual outcomes. The probabilities *(0.41579 0.58421)* show the likelihood of each of the two possible outcomes for these observations, suggesting a majority likelihood of the second outcome at about *58.42%*.

d. Create a plot of the tree, and interpret the results.

```
plot(tree_fit_oj)
text(tree_fit_oj,pretty=0)
```



e. Predict the response on the test data, and produce a confusion matrix comparing the test labels to the predicted test labels. What is the test error rate?

```
pred_tree_fit_oj=predict(tree_fit_oj,newdata=test_set,type="class")
# Confusion Matrix
print(table(test_set[,"Purchase"],pred_tree_fit_oj))
```

```
   pred_tree_fit_oj
     CH  MM
 CH 131  33
 MM  19  87
```

```
# Miss Classification Rate
round((1 - ( (131 + 87) / (131 + 33 + 19 + 87)))*100,2)
```

```
19.26
```

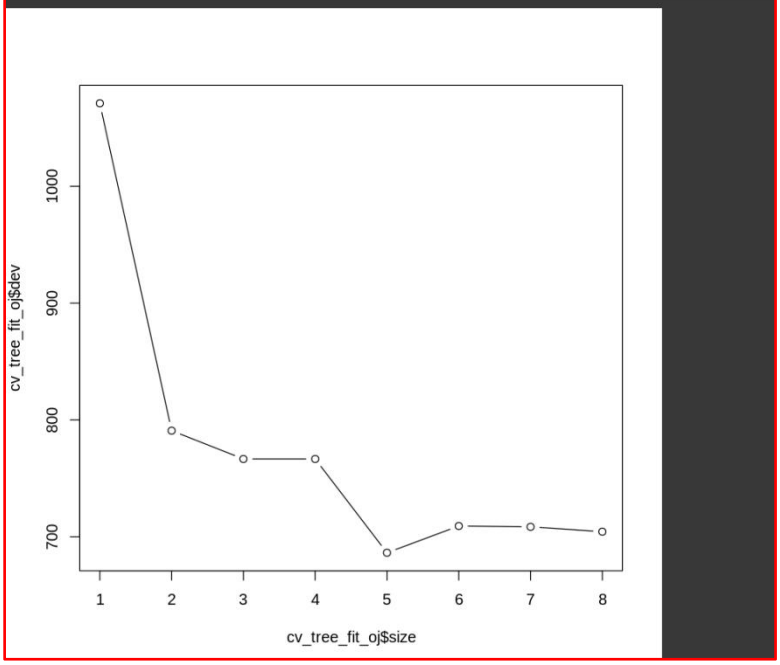
The test error rate of 0.1926 indicates that approximately 20 out of every 100 predictions made by our model on the test dataset are incorrect. This error rate is somewhat higher than what we observed during training, which is generally anticipated. It is typical for models to exhibit slightly poorer performance on new, unseen data as opposed to the data on which they were trained.

f. Apply the cv.tree() function to the training set in order to determine the optimal tree size.

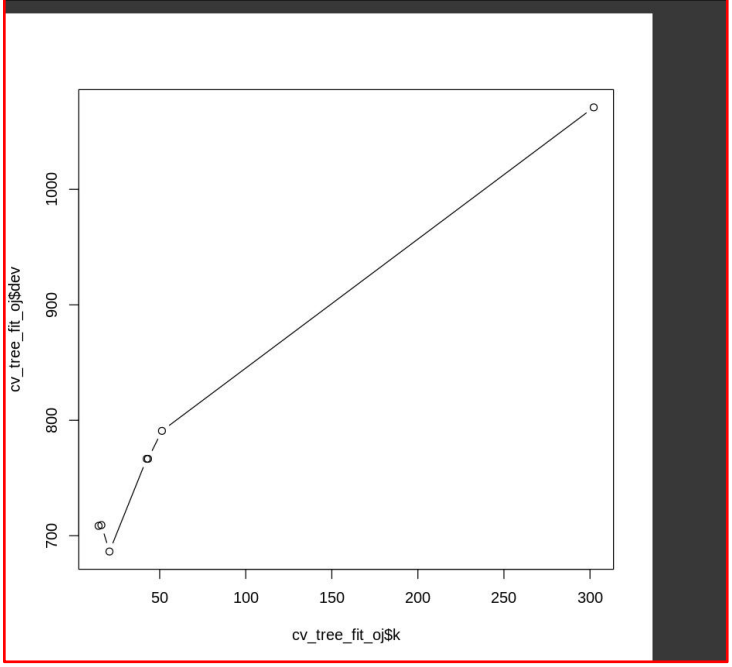
```
cv_tree_fit_oj=cv.tree(tree_fit_oj,FUN=prune.tree)
```

g. Produce a plot with tree size on the x-axis and cross-validated classification error rate on the y-axis.

```
plot(cv_tree_fit_oj$size,cv_tree_fit_oj$dev,type="b")
```




```
plot(cv_tree_fit_oj$k,cv_tree_fit_oj$dev,type="b")
```



h. Which tree size corresponds to the lowest cross-validated classification error rate?

```
cv_tree_fit_oj$dev
```

```
704.334774349694 · 708.512893339225 · 709.23661690041 · 686.310577317589 · 766.597400795312 · 766.597400795312 · 790.81565262698 · 1071.01265535034
```

```
order(cv_tree_fit_oj$dev,decreasing=TRUE)
```

```
8·7·5·6·3·2·1·4
```

```
cv_tree_fit_oj$dev[order(cv_tree_fit_oj$dev,decreasing=TRUE)]
```

```
1071.01265535034 · 790.81565262698 · 766.597400795312 · 766.597400795312 · 709.23661690041 · 708.512893339225 · 704.334774349694 · 686.310577317589
```

```
which.min(cv_tree_fit_oj$dev)
```

```
4
```

```
cv_tree_fit_oj$size
```

```
8·7·6·5·4·3·2·1
```

```
cv_tree_fit_oj$size[which.min(cv_tree_fit_oj$dev)]
```

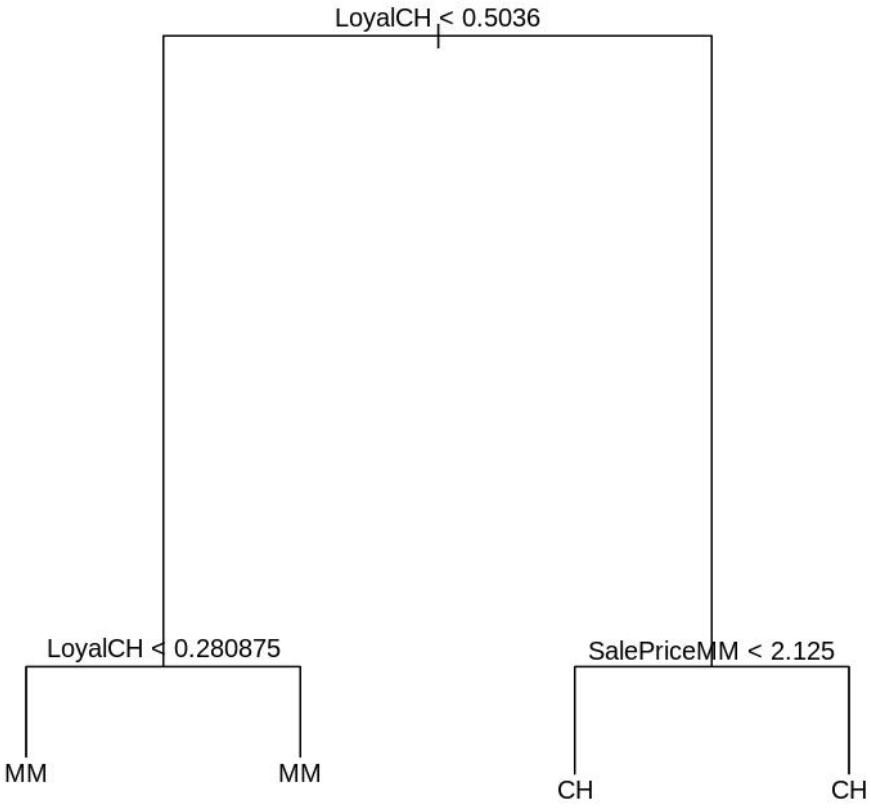
```
5
```

```
cv_tree_fit_oj$size[5]
```

```
4
```

i. Produce a pruned tree corresponding to the optimal tree size obtained using cross-validation. If cross-validation does not lead to selection of a pruned tree, then create a pruned tree with five terminal nodes.

```
# Best size=4
pruned_tree_fit_oj=prune.tree(tree_fit_oj,best=4)

plot(pruned_tree_fit_oj)
text(pruned_tree_fit_oj,pretty=0)
```



j. Compare the training error rates between the pruned and unpruned trees. Which is higher?

```
summary(pruned_tree_fit_oj)
```

```
Classification tree:
snip.tree(tree = tree_fit_oj, nodes = 5:6)
Variables actually used in tree construction:
[1] "LoyalCH"     "SalePriceMM"
Number of terminal nodes:  4
Residual mean deviance:  0.8447 = 672.4 / 796
Misclassification error rate: 0.1962 = 157 / 800
```

The test error rate of 0.1962 indicates that approximately 20 out of every 100 predictions made by our model on the test dataset are incorrect. This error rate is somewhat higher than what we observed during training, which is generally anticipated. It is typical for models to exhibit slightly poorer performance on new, unseen data as opposed to the data on which they were trained. The pruned have the lowest number of incorrect predictions which is 18.

k. Compare the test error rates between the pruned and unpruned trees. Which is higher?

```
pred_pruned_tree_fit_oj=predict(pruned_tree_fit_oj,newdata=test_set,type="class")
# Confusion Matrix
print(table(test_set[,"Purchase"],pred_pruned_tree_fit_oj))
```

```
   pred_pruned_tree_fit_oj
     CH  MM
 CH 127  37
 MM  20  86
```

```
# Miss Classification Rate
round((1 - ( (127 + 86) / (127 + 37 + 20 + 86)))*100,2)
```

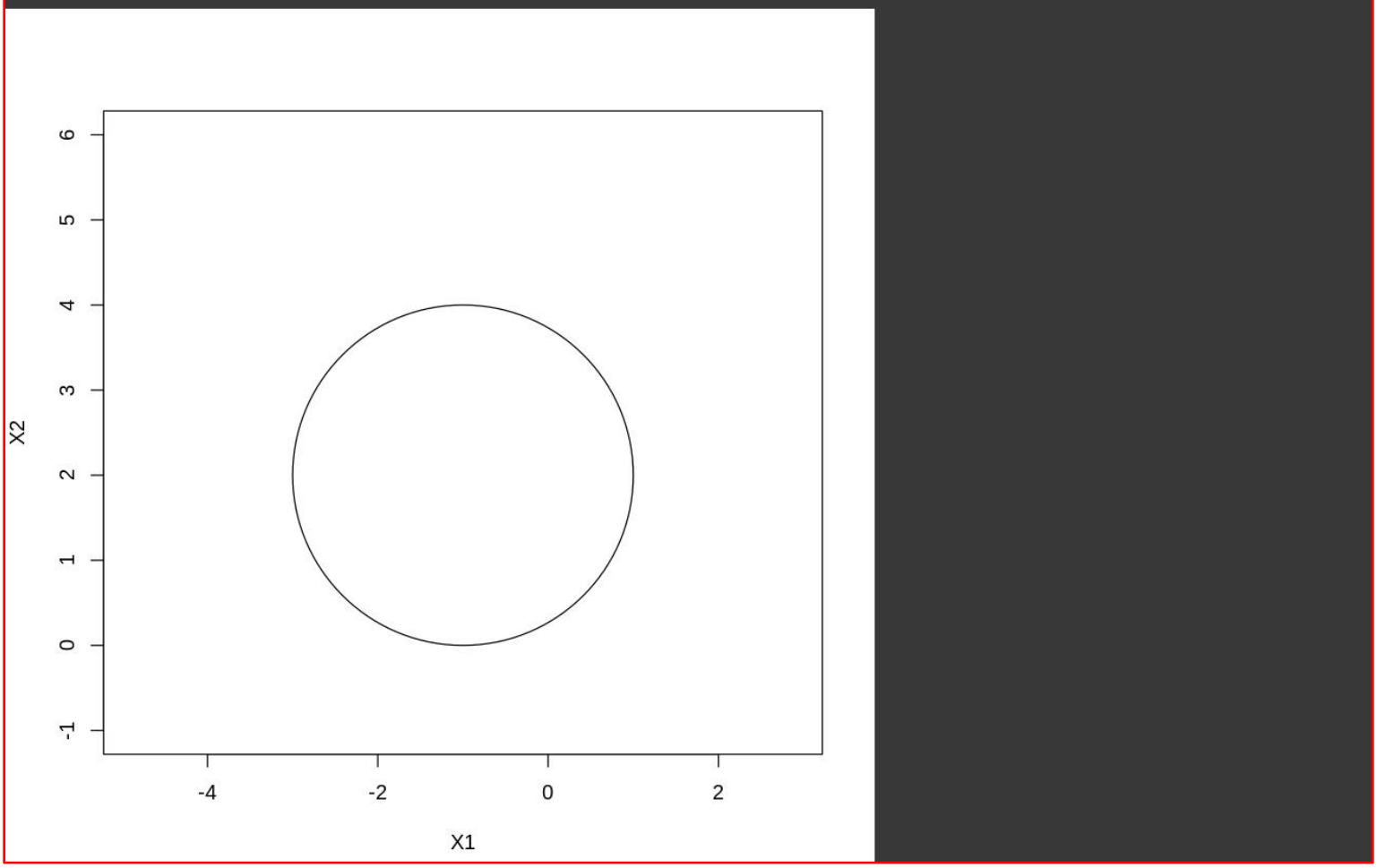
```
21.11
```

The pruned trees have a miss classification rate of 19.26% whereas the pruned trees have a higher classification rate in comparison (21.11%).

3. We have seen that in p = 2 dimensions, a linear decision boundary takes the form $\beta_0 + \beta_1 X_1 + \beta_2 X_2 = 0$. We now investigate a non-linear decision boundary.

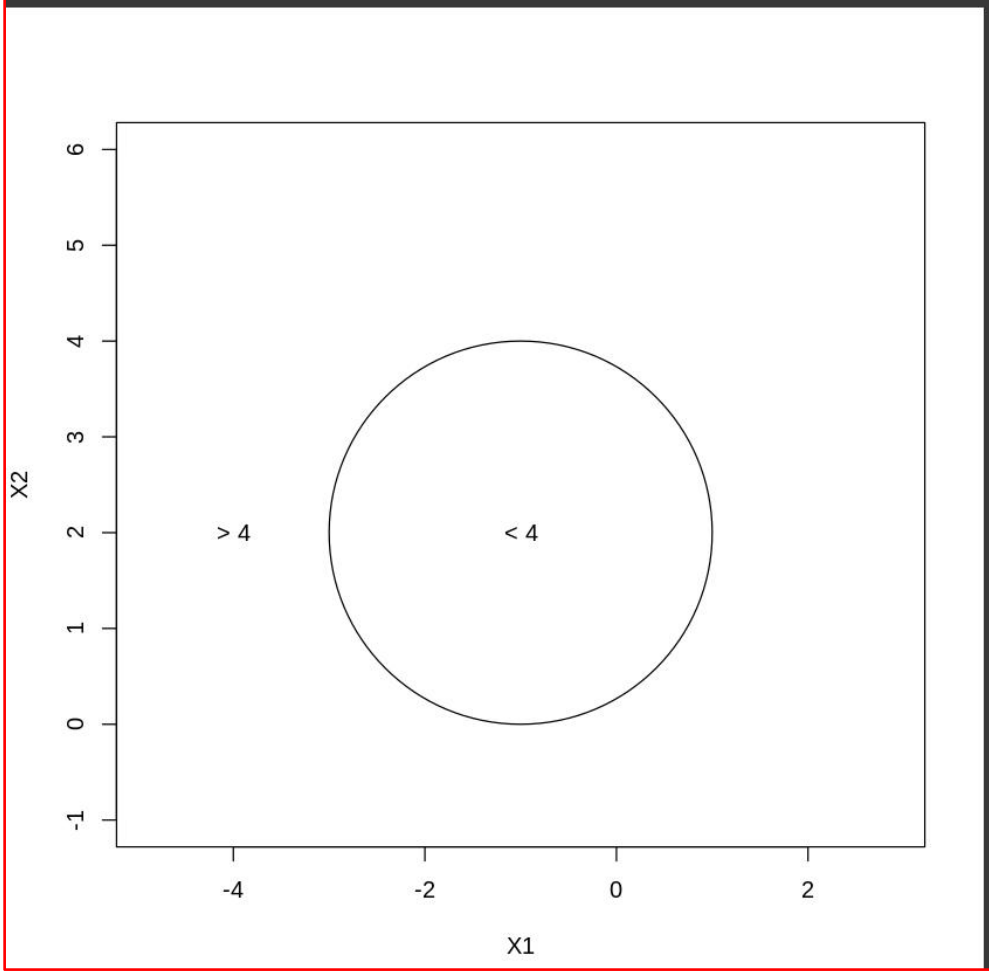
a. Sketch the curve: $(1 + X_1)^2 + (2 - X_2)^2 = 4$.

```
radius=2
plot(NA,NA,type="n",xlim=c(-4,2),ylim=c(-1,6),asp=1,xlab="X1",ylab="X2")
symbols(c(-1),c(2),circles=c(radius),add=TRUE,inches=FALSE)
```



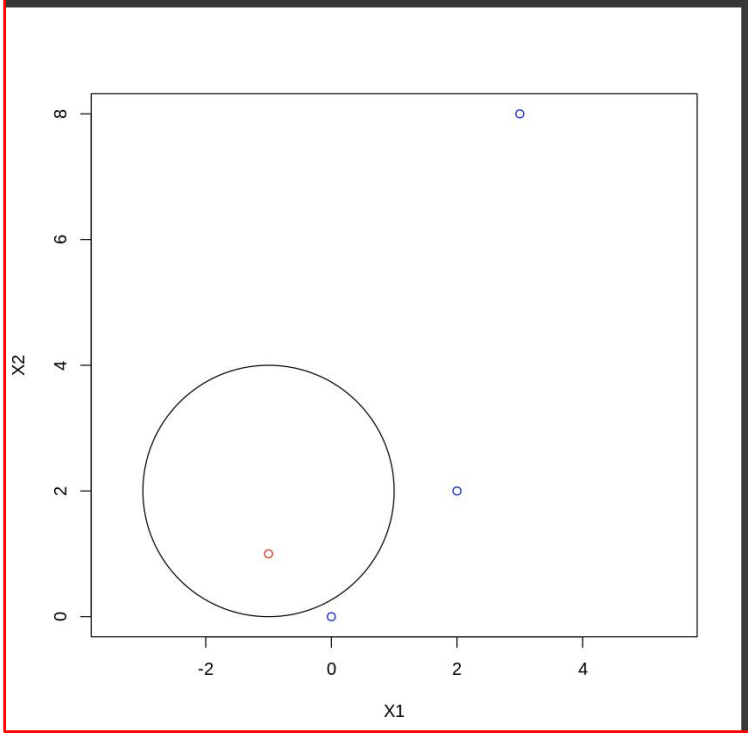
b. On your sketch, indicate the set of points for which: $(1 + X_1)^2 + (2 - X_2)^2 > 4$, as well as the set of points for which: $(1 + X_1)^2 + (2 - X_2)^2 \leq 4$.

```
radius=2
plot(NA,NA,type="n",xlim=c(-4,2),ylim=c(-1,6),asp=1,xlab="X1",ylab="X2")
symbols(c(-1),c(2),circles=c(radius),add=TRUE,inches=FALSE)
text(c(-1),c(2),"< 4")
text(c(-4),c(2),"> 4")
```



c. Suppose that a classifier assigns an observation to the blue class if $(1 + X_1)^2 + (2 - X_2)^2 > 4$, and to the red class otherwise. To what class is the observation (0, 0) classified? (−1, 1)? (2, 2)? (3, 8)?

```
radius=2
plot(c(0,-1,2,3),c(0,1,2,8),col=c("blue","red","blue","blue"),type="p",asp=1,xlab="X1",ylab="X2")
symbols(c(-1),c(2),circles=c(radius),add=TRUE,inches=FALSE)
```



d. Argue that while the decision boundary in (c) is not linear in terms of X_1 and X_2, it is linear in terms of X_1, X_1^2, X_2 and X_2^2.

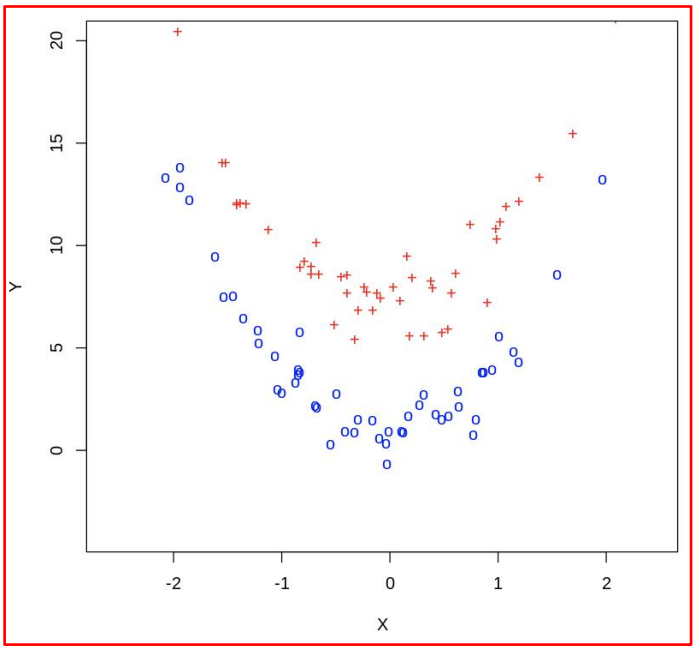
Expanding the decision boundary yields the equation
$(1 + 2X_1 + X_1^2) + (4 - 4X_2 + X_2^2) = 4$
which simplifies to
$X1^2 + X_2^2 + 2X_1 - 4X_2 + 1 = 0$
Consequently, the circular boundary transforms into a linear form within the feature space, namely (X_1^2, X_2^2, X_1, X_2).

4. Generate a simulated two-class data set with 100 observations and two features in which there is a visible but non-linear separation between the two classes. Show that in this setting, a support vector machine with a polynomial kernel (with degree greater than 1) or a radial kernel will outperform a support vector classifier on the training data. Which technique performs best on the test data? Make plots and report training and test error rates in order to back up your assertions.

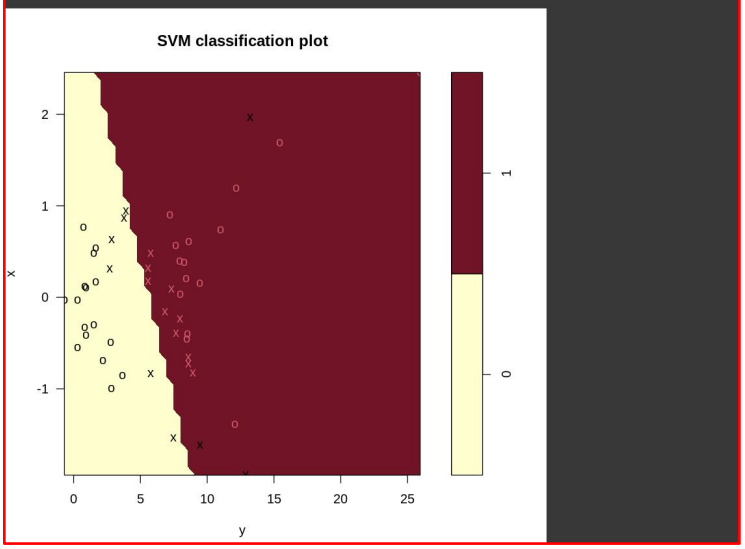
```r
install.packages("e1071")
library(e1071)
set.seed(123)
```

```
Installing package into '/usr/local/lib/R/site-library'
(as 'lib' is unspecified)

also installing the dependency 'proxy'
```

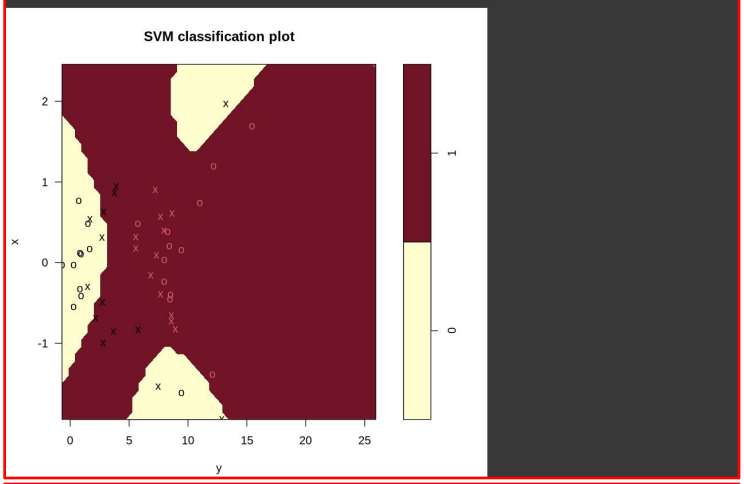
```r
# y=3(x^2) + 4
x=rnorm(100)
y=3 * (x^2) + 4 + rnorm(100)
z=rep(0,100)

temp_train_index=sample(100,50)
y[temp_train_index]=y[temp_train_index] + 3
y[-temp_train_index]=y[-temp_train_index] - 3
z[temp_train_index]=1

plot(x[temp_train_index],y[temp_train_index],pch="+",lwd=4,col="red",ylim=c(-4,20),xlab="X",ylab="Y")
points(x[-temp_train_index],y[-temp_train_index],pch="o",lwd=4,col="blue")

train_index=c(sample(temp_train_index,25),sample(setdiff(1:100,temp_train_index),25))
train_set=data.frame(x=x[train_index],y=y[train_index],z=as.factor(z[train_index]))
test_set=data.frame(x=x[-train_index],y=y[-train_index],z=as.factor(z[-train_index]))
```



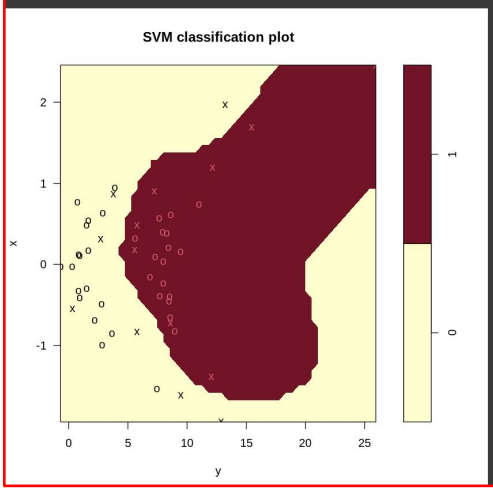

```r
# Linear SVM
svm_linear_fit=svm(z~.,data=train_set,kernel="linear",cost=10)
plot(svm_linear_fit,train_set)
```



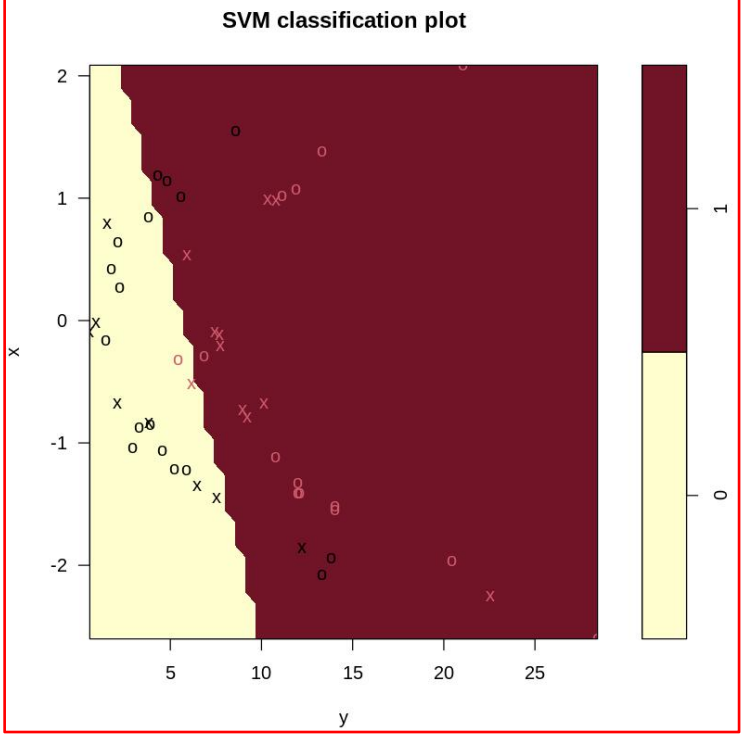
```r
# Polynomial SVM
svm_polynomial_fit=svm(z~.,data=train_set,kernel="polynomial",cost=10)
plot(svm_polynomial_fit,train_set)
```



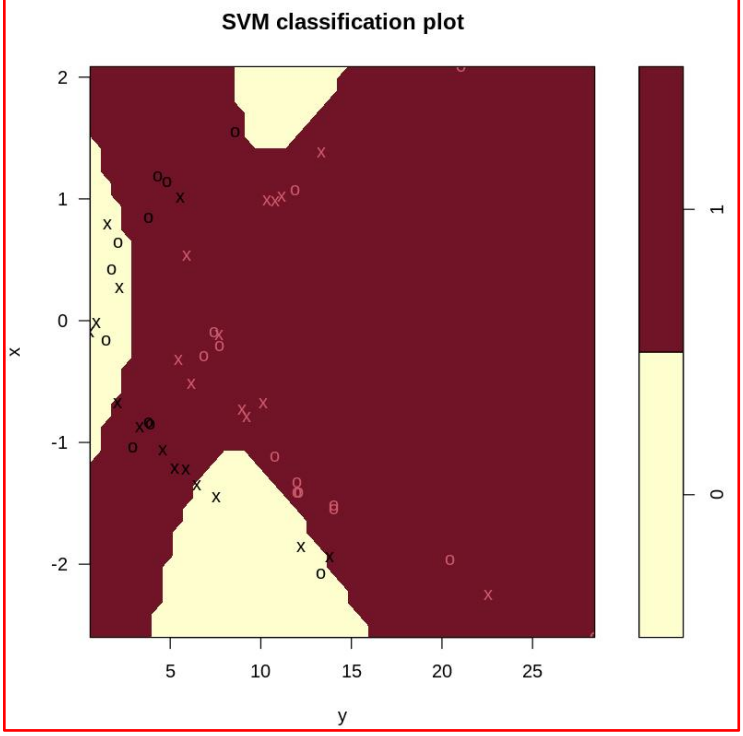

```r
# Radial SVM
svm_radial_fit=svm(z~.,data=train_set,kernel="radial",cost=10,gamma=1)
plot(svm_radial_fit,train_set)
```



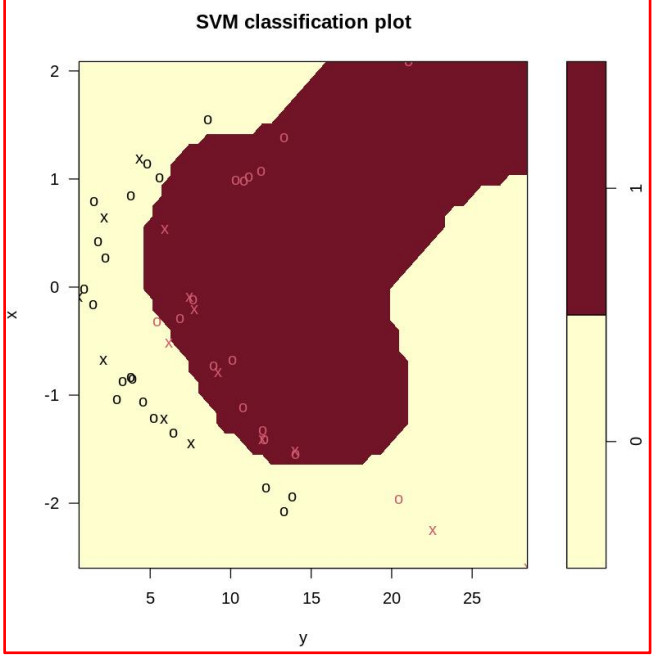

```r
plot(svm_linear_fit,test_set)
```




```r
plot(svm_polynomial_fit,test_set)
```



```
plot(svm_radial_fit,test_set)
```




```
# Linear SVM Confusion Matrix - Train
print(table(z[train_index],predict(svm_linear_fit,train_set)))
```

```
   0  1
0 21  4
1  0 25
```

```
# Linear SVM Miss Classification Rate - Train
round((1 - ( (21 + 25) / (21 + 4 + 25)))*100,2)
```

```
8
```

```
# Polynomial SVM Confusion Matrix - Train
print(table(z[train_index],predict(svm_polynomial_fit,train_set)))
```

```
   0  1
0 17  8
1  0 25
```

```
# Polynomial SVM Miss Classification Rate - Train
round((1 - ( (17 + 25) / (17 + 8 + 25)))*100,2)
```

```
16
```

```
# Radial SVM Confusion Matrix - Train
print(table(z[train_index],predict(svm_radial_fit,train_set)))
```

```
   0  1
0 25  0
1  0 25
```

```
# Radial SVM Miss Classification Rate - Train
round((1 - ( (25 + 25) / (25 + 25)))*100,2)
```

```
0
```

```
# Linear SVM Confusion Matrix - Test
print(table(z[-train_index],predict(svm_linear_fit,test_set)))
```

```
   0  1
0 18  7
1  2 23
```

```
# Linear SVM Miss Classification Rate - Test
round((1 - ( (18 + 23) / (18 + 7 + 2 + 23)))*100,2)
```

```
18
```

```
# Polynomial SVM Confusion Matrix - Test
print(table(z[-train_index],predict(svm_polynomial_fit,test_set)))
```

```
   0  1
0 10 15
1  0 25
```

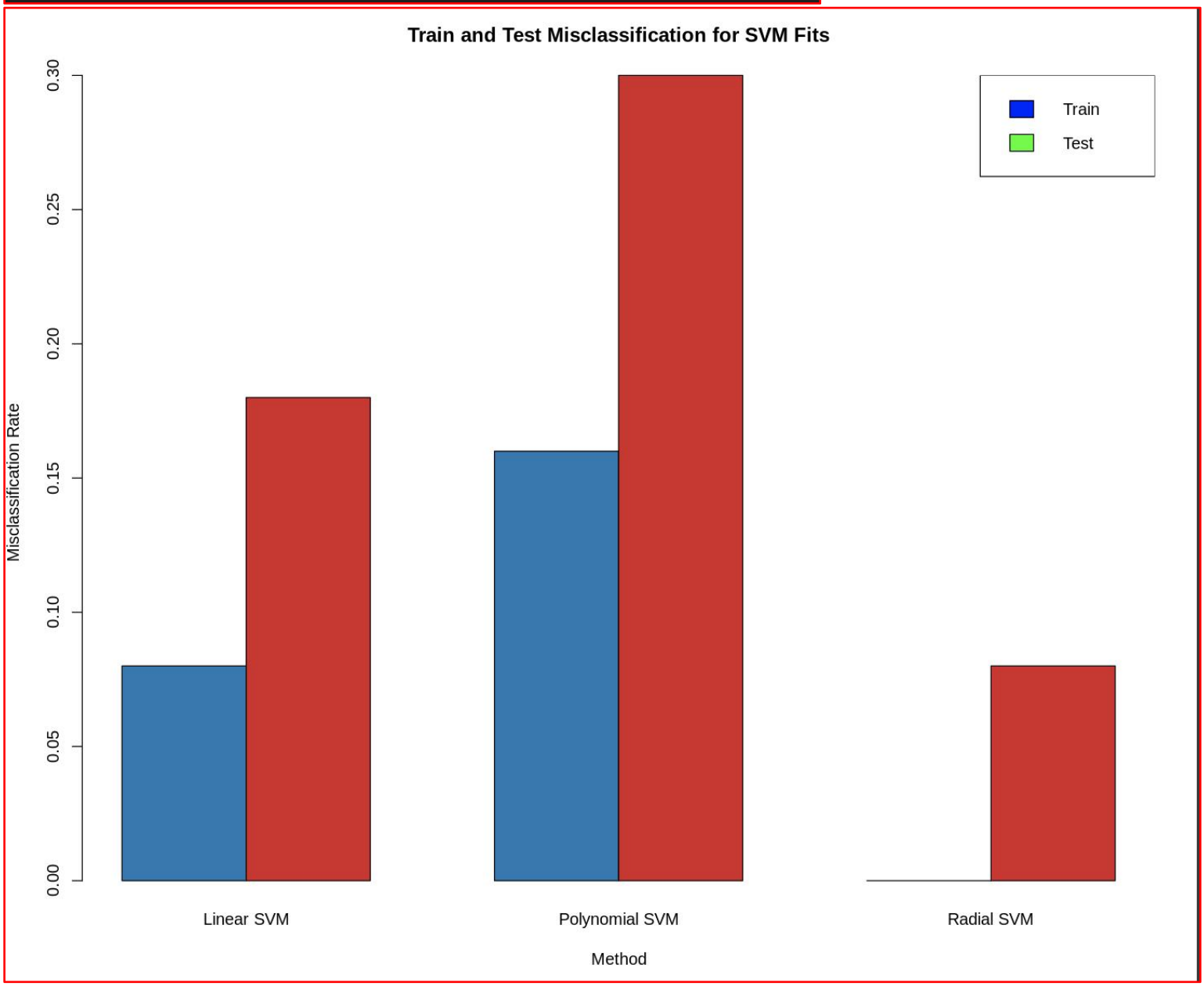
```
# Radial SVM Confusion Matrix - Test
print(table(z[-train_index],predict(svm_radial_fit,test_set)))
```

```
   0  1
0 25  0
1  4 21
```

```
# Radial SVM Miss Classification Rate - Test
round((1 - ( (25 + 21) / (25 + 4 + 21)))*100,2)
```

```
8
```

```r
options(repr.plot.width=12, repr.plot.height=10)

svm_result_df=data.frame(
  Method=c("Linear SVM","Polynomial SVM","Radial SVM"),
  Train_Misclassification=c(0.08,0.16,0.0),
  Test_Misclassification=c(0.18,0.30,0.08)
  )

barplot(
  t(as.matrix(svm_result_df[,-1])),
  beside=TRUE,
  # col=c("blue","green"),
  col=c("#1f77b4","#d62728"),
  # legend.text=svm_result_df$Method,
  names.arg = svm_result_df$Method,
  xlab="Method",
  ylab="Misclassification Rate",
  main="Train and Test Misclassification for SVM Fits"
  )
legend("topright",legend=c("Train","Test"),fill=c("blue","green"))
```



5. We have seen that we can fit an SVM with a non-linear kernel in order to perform classification using a non-linear decision boundary. We will now see that we can also obtain a non-linear decision boundary by performing logistic regression using non-linear transformations of the features.

a. Generate a data set with n = 500 and p = 2, such that the observations belong to two classes with a quadratic decision boundary between them.

```
install.packages(c('glmnet','e1071'))
library(glmnet)
library(e1071)
set.seed(123)
```

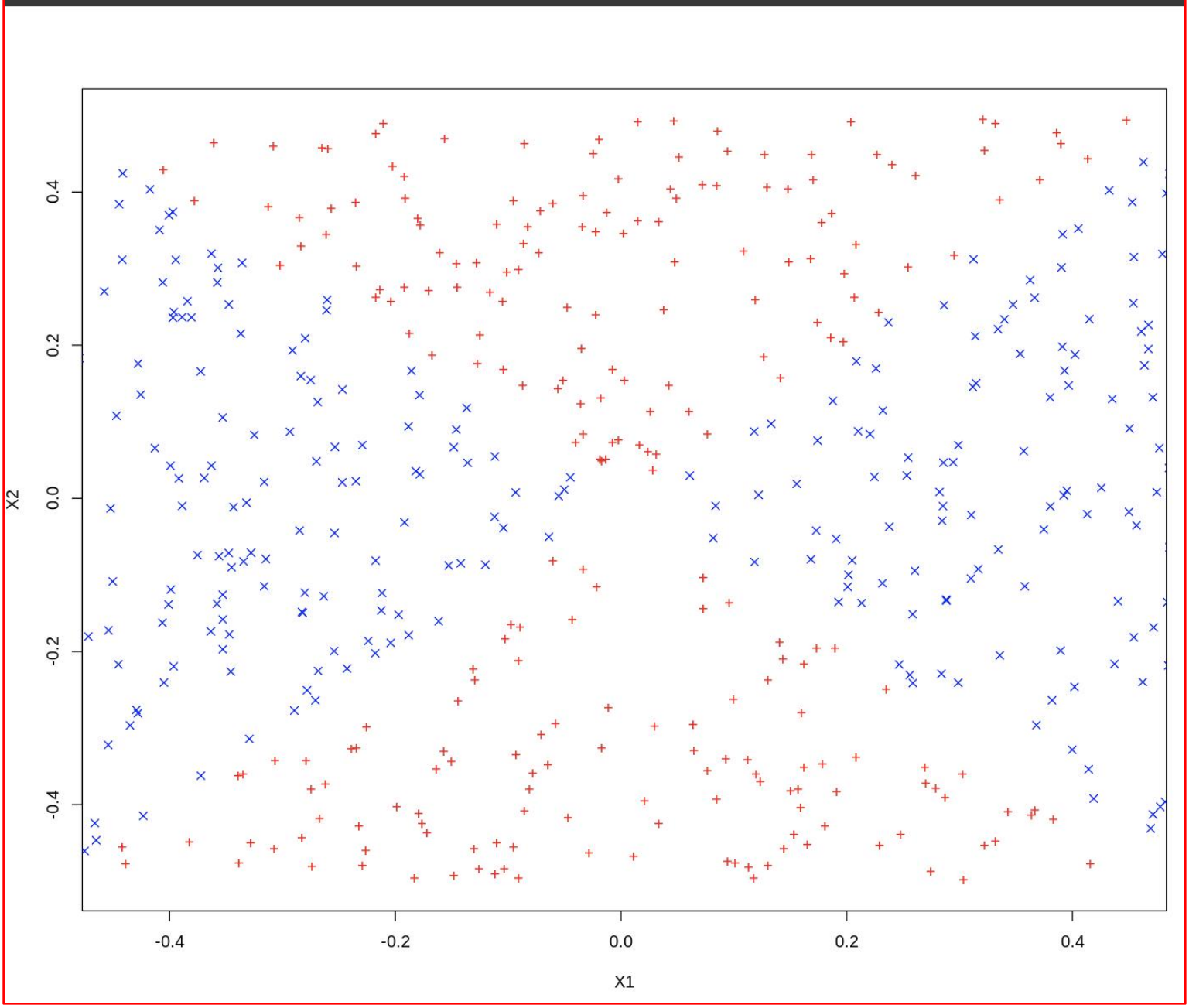
```
Installing packages into '/usr/local/lib/R/site-library'
(as 'lib' is unspecified)

also installing the dependencies 'iterators', 'foreach', 'shape', 'Rcpp', 'RcppEigen', 'proxy'


Loading required package: Matrix

Loaded glmnet 4.1-8
```

(a)

```
x1=runif(500) - 0.5
x2=runif(500) - 0.5
y=1 * (x1^2 - x2^2 > 0)
```

b. Plot the observations, coloured according to their class labels. Your plot should display X_1 on the x-axis, and X_2 on the y-axis.

```
options(repr.plot.width=12, repr.plot.height=10)

plot(x1[y == 0],x2[y == 0],col="red",xlab="X1",ylab="X2",pch="+")
points(x1[y == 1],x2[y == 1],col="blue",pch=4)
```



c. Fit a logistic regression model to the data, using X_1 and X_2 as predictors.

```
log_model_fit=glm(y ~ x1 + x2,family="binomial")
summary(log_model_fit)
```

```
Call:
glm(formula = y ~ x1 + x2, family = "binomial")

Coefficients:
            Estimate Std. Error z value Pr(>|z|)
(Intercept)  0.04792    0.08949   0.535    0.592
x1          -0.03999    0.31516  -0.127    0.899
x2           0.11509    0.30829   0.373    0.709

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 692.86  on 499  degrees of freedom
Residual deviance: 692.71  on 497  degrees of freedom
AIC: 698.71

Number of Fisher Scoring iterations: 3
```

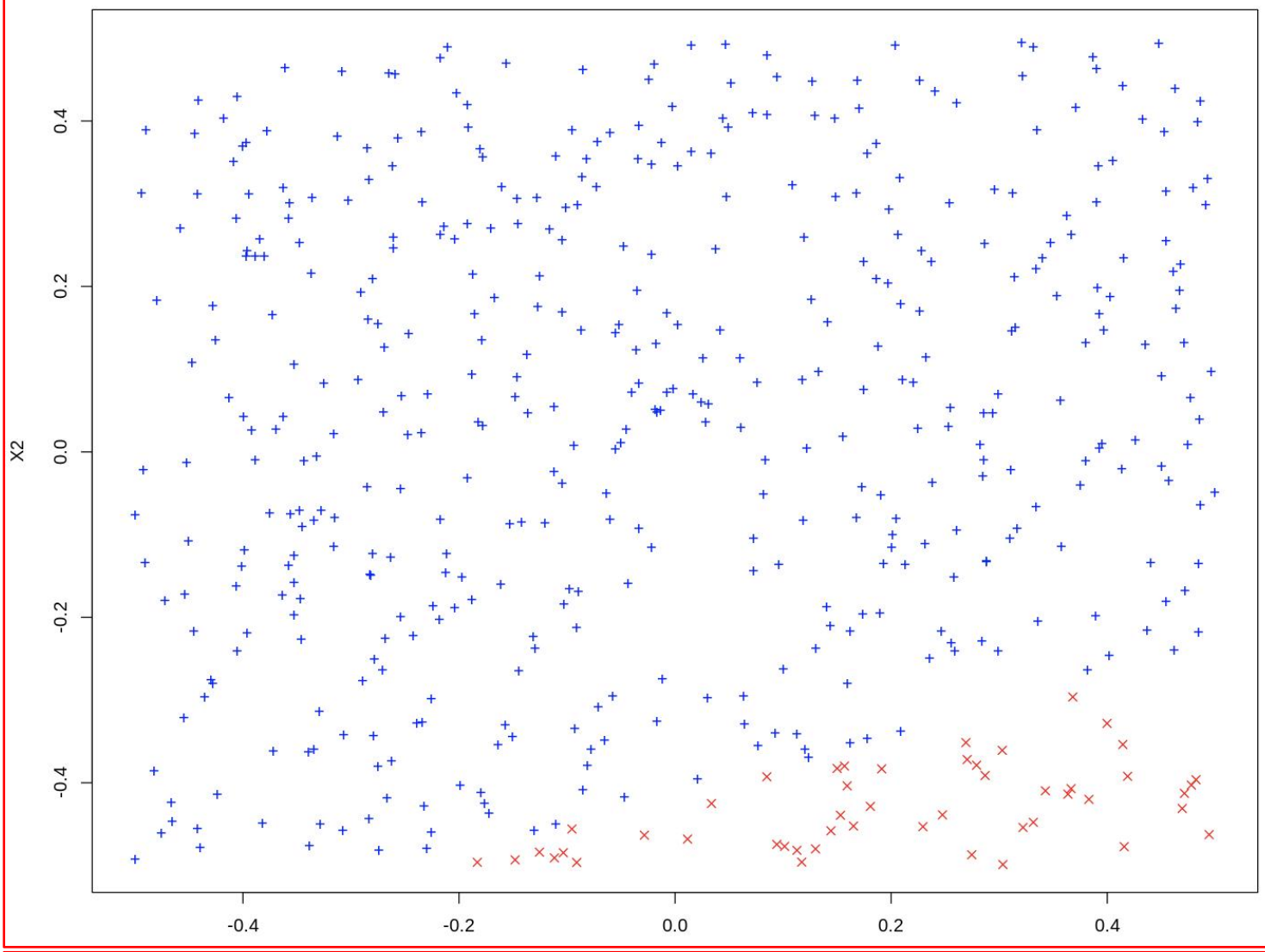
```
log_model_fit
```

```
Call:  glm(formula = y ~ x1 + x2, family = "binomial")

Coefficients:
(Intercept)           x1           x2
    0.04792     -0.03999      0.11509

Degrees of Freedom: 499 Total (i.e. Null);  497 Residual
Null Deviance:     692.9
Residual Deviance: 692.7  AIC: 698.7
```

d. Apply this model to the training data in order to obtain a predicted class label for each training observation. Plot the observations, coloured according to the predicted class labels. The decision boundary should be linear.

```
df=data.frame(x1=x1,x2=x2,y=y)
```

```
options(repr.plot.width=12, repr.plot.height=10)

log_model_prob=predict(log_model_fit,df,type="response")
log_model_pred=ifelse(log_model_prob > 0.5,1,0)
pos_data=df[log_model_pred == 1,]
neg_data=df[log_model_pred == 0,]
plot(pos_data$x1,pos_data$x2,col="blue",xlab="X1",ylab="X2",pch="+")
points(neg_data$x1,neg_data$x2,col="red",pch=4)
```




```
# Logistic Regression Confusion Matrix - Train
print(table(df[,"y"],log_model_pred))
```

```
  log_model_pred
     0   1
 0  40 204
 1   9 247
```

```
# Logistic Regression Miss Classification Rate - Train
round((1 - ( (40 + 247) / (40 + 204 + 9 + 247)))*100,2)
```

```
42.6
```

e. Now fit a logistic regression model to the data using non-linear functions of X_1 and X_2 as predictors (e.g. X_1^2, $X_1 \times X_2$, $\log(X_2)$, and so forth).

```
log_model_non_linear_fit=glm(y ~ poly(x1,2) + poly(x2,2) + log(x1 + 1e-10) + log(x2 + 1e-10) + I(x1 * x2),data=df,family="binomial",maxit=1000)
```

```
Warning message in log(x1 + 1e-10):
"NaNs produced"
Warning message in log(x2 + 1e-10):
"NaNs produced"
Warning message:
"glm.fit: fitted probabilities numerically 0 or 1 occurred"
```

```
log_model_non_linear_fit=glm(y ~ poly(x1,2) + poly(x2,2) + I(x1 * x2),data=df,family="binomial",maxit=1000)
```

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

```
log_model_non_linear_fit
```

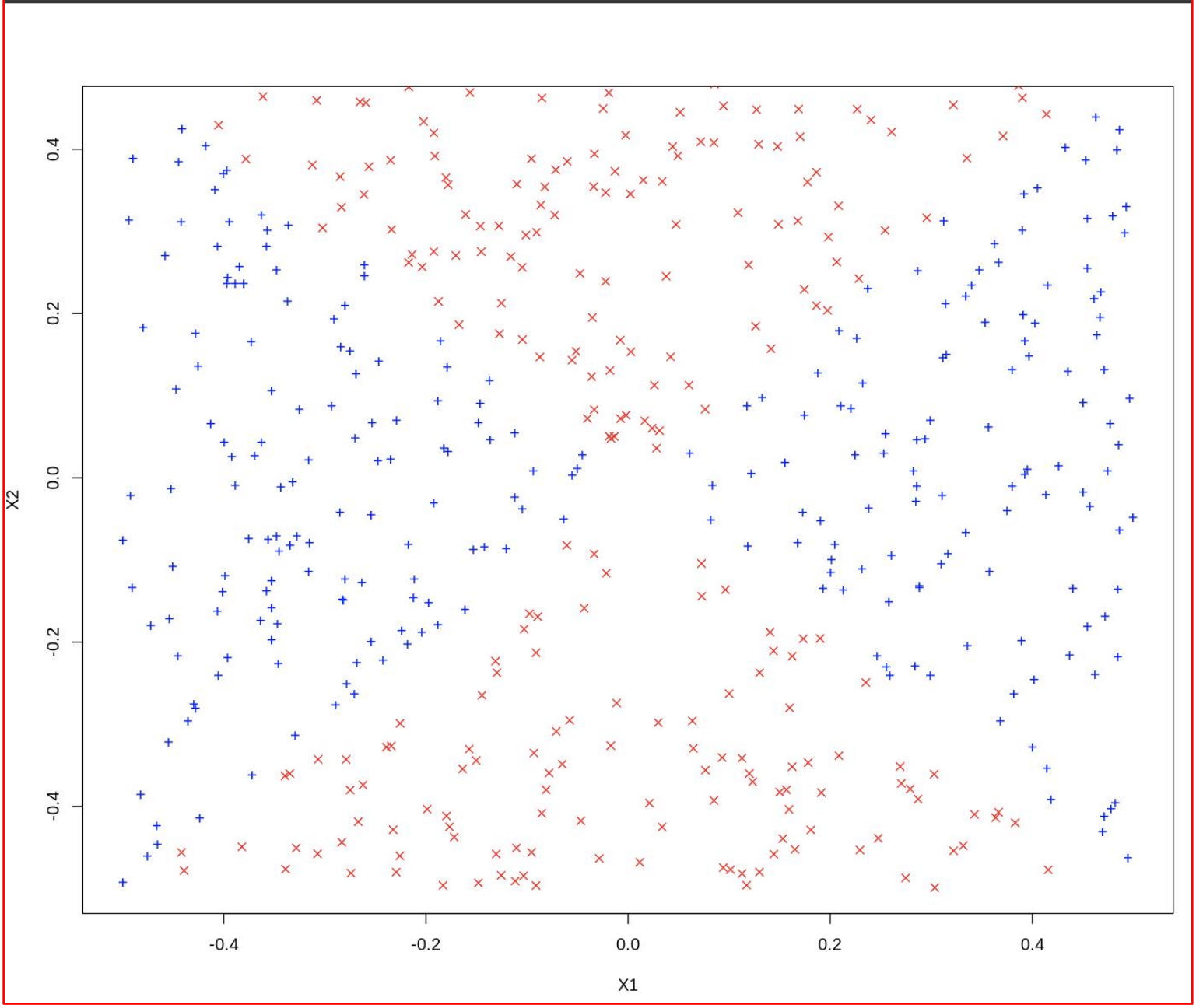
```
Call:  glm(formula = y ~ poly(x1, 2) + poly(x2, 2) + I(x1 * x2), family = "binomial",
    data = df, maxit = 1000)

Coefficients:
 (Intercept)  poly(x1, 2)1  poly(x1, 2)2  poly(x2, 2)1  poly(x2, 2)2
      -82.36        785.50      31563.15       3253.10     -32794.06
  I(x1 * x2)
      861.31

Degrees of Freedom: 499 Total (i.e. Null);  494 Residual
Null Deviance:     692.9
Residual Deviance: 1.05e-09   AIC: 12
```

```
summary(log_model_non_linear_fit)
```

```
Call:
glm(formula = y ~ poly(x1, 2) + poly(x2, 2) + I(x1 * x2), family = "binomial",
    data = df, maxit = 1000)

Coefficients:
               Estimate Std. Error z value Pr(>|z|)
(Intercept)  -8.236e+01  1.429e+05  -0.001    1.000
poly(x1, 2)1  7.855e+02  3.283e+06   0.000    1.000
poly(x1, 2)2  3.156e+04  3.296e+07   0.001    0.999
poly(x2, 2)1  3.253e+03  4.779e+06   0.001    0.999
poly(x2, 2)2 -3.279e+04  3.428e+07  -0.001    0.999
I(x1 * x2)    8.613e+02  1.667e+06   0.001    1.000

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 6.9286e+02  on 499  degrees of freedom
Residual deviance: 1.0502e-09  on 494  degrees of freedom
AIC: 12

Number of Fisher Scoring iterations: 34
```

f. Apply this model to the training data in order to obtain a predicted class label for each training observation. Plot the observations, coloured according to the predicted class labels. The decision boundary should be obviously non-linear. If it is not, then repeat (a)-(e) until you come up with an example in which the predicted class labels are obviously non-linear.

```
options(repr.plot.width=12, repr.plot.height=10)

log_model_non_linear_prob=predict(log_model_non_linear_fit,df,type="response")
log_model_non_linear_pred=ifelse(log_model_non_linear_prob > 0.5,1,0)
pos_data=df[log_model_non_linear_pred == 1,]
neg_data=df[log_model_non_linear_pred == 0,]
plot(pos_data$x1,pos_data$x2,col="blue",xlab="X1",ylab="X2",pch="+")
points(neg_data$x1,neg_data$x2,col="red",pch=4)
```



```r
# Logistic Regression Non-Linear Confusion Matrix - Train
print(table(df[,"y"],log_model_non_linear_pred))
```

```
  log_model_non_linear_pred
     0   1
0 244   0
1   0 256
```

```r
# Logistic Regression Non-Linear Miss Classification Rate - Train
round((1 - ( (244 + 256) / (244 + 256)))*100,2)
```

```
0
```

g. Fit a support vector classifier to the data with X_1 and X_2 as predictors. Obtain a class prediction for each training observation. Plot the observations, coloured according to the predicted class labels.

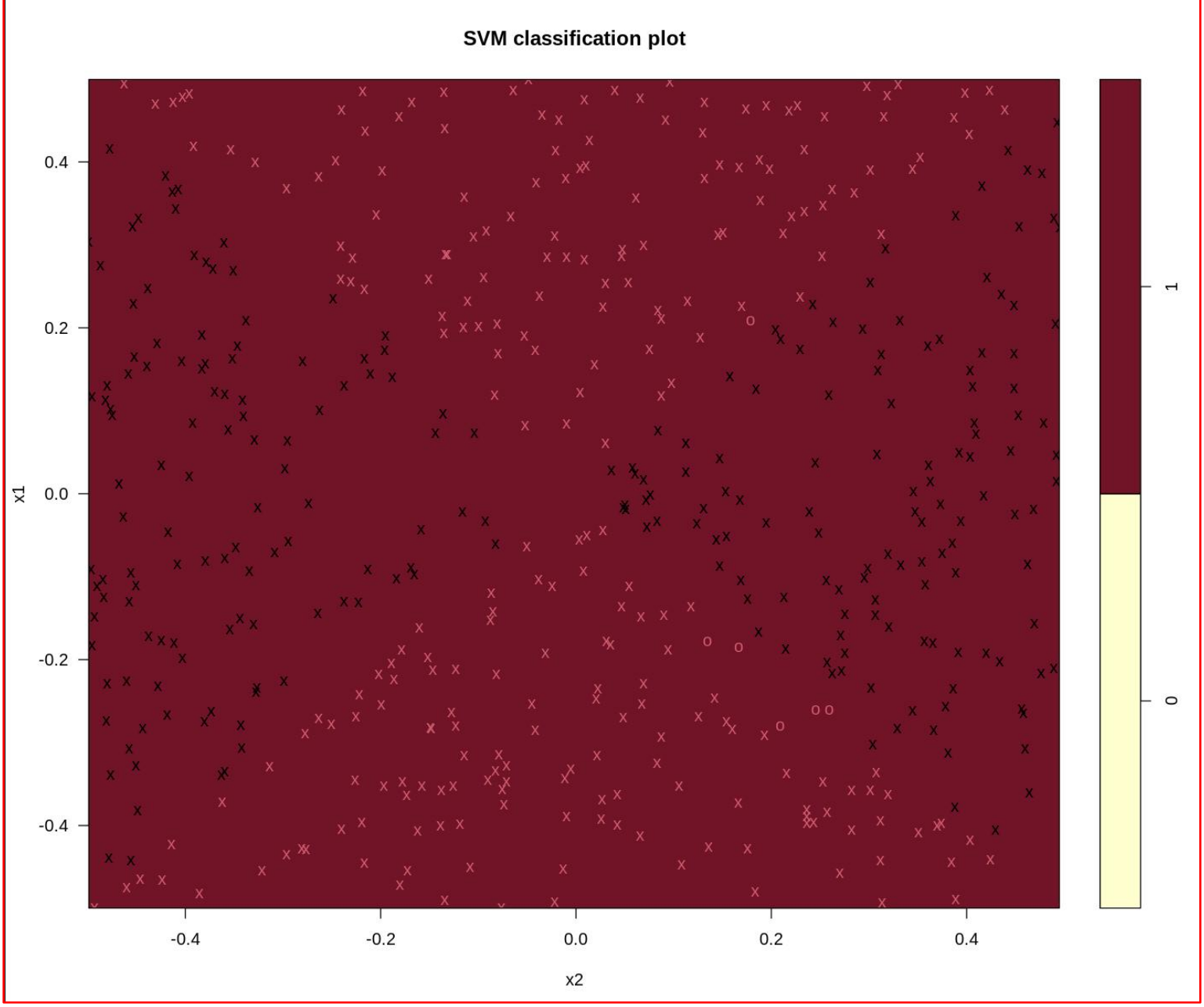
```r
# Linear SVM
svm_linear_fit=svm(as.factor(y) ~ x1 + x2,data=df,kernel="linear",cost=10)
summary(svm_linear_fit)
```

```
Call:
svm(formula = as.factor(y) ~ x1 + x2, data = df, kernel = "linear",
    cost = 10)


Parameters:
   SVM-Type:  C-classification
 SVM-Kernel:  linear
       cost:  10

Number of Support Vectors:  494

 ( 250 244 )


Number of Classes:  2

Levels:
 0 1
```

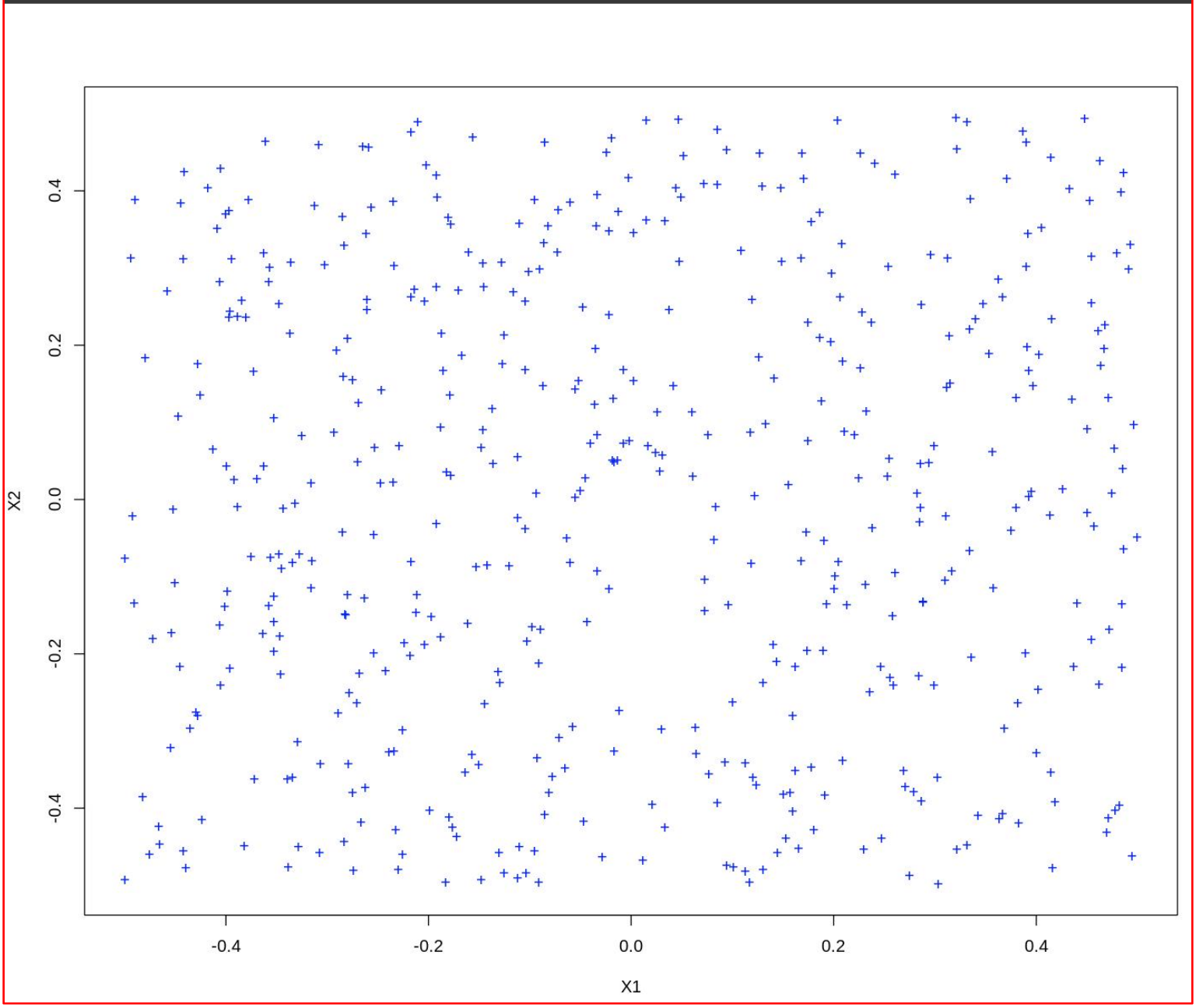
```r
[ ]  svm_linear_fit
```

```
Call:
svm(formula = as.factor(y) ~ x1 + x2, data = df, kernel = "linear",
    cost = 10)


Parameters:
   SVM-Type:  C-classification
 SVM-Kernel:  linear
       cost:  10

Number of Support Vectors:  494
```

```r
options(repr.plot.width=12, repr.plot.height=10)

plot(svm_linear_fit,df)
```



```r
options(repr.plot.width=12, repr.plot.height=10)

svm_linear_pred=predict(svm_linear_fit,df)
pos_data=df[svm_linear_pred == 1,]
neg_data=df[svm_linear_pred == 0,]
plot(pos_data$x1,pos_data$x2,col="blue",xlab="X1",ylab="X2",pch="+")
points(neg_data$x1,neg_data$x2,col="red",pch=4)
```



```r
# Linear SVM Confusion Matrix - Train
print(table(df[,"y"],predict(svm_linear_fit,df)))
```

```
   0   1
0  0 244
1  0 256
```

```r
# Linear SVM Miss Classification Rate - Train
round((1 - ( (256) / (244 + 256)))*100,2)
```

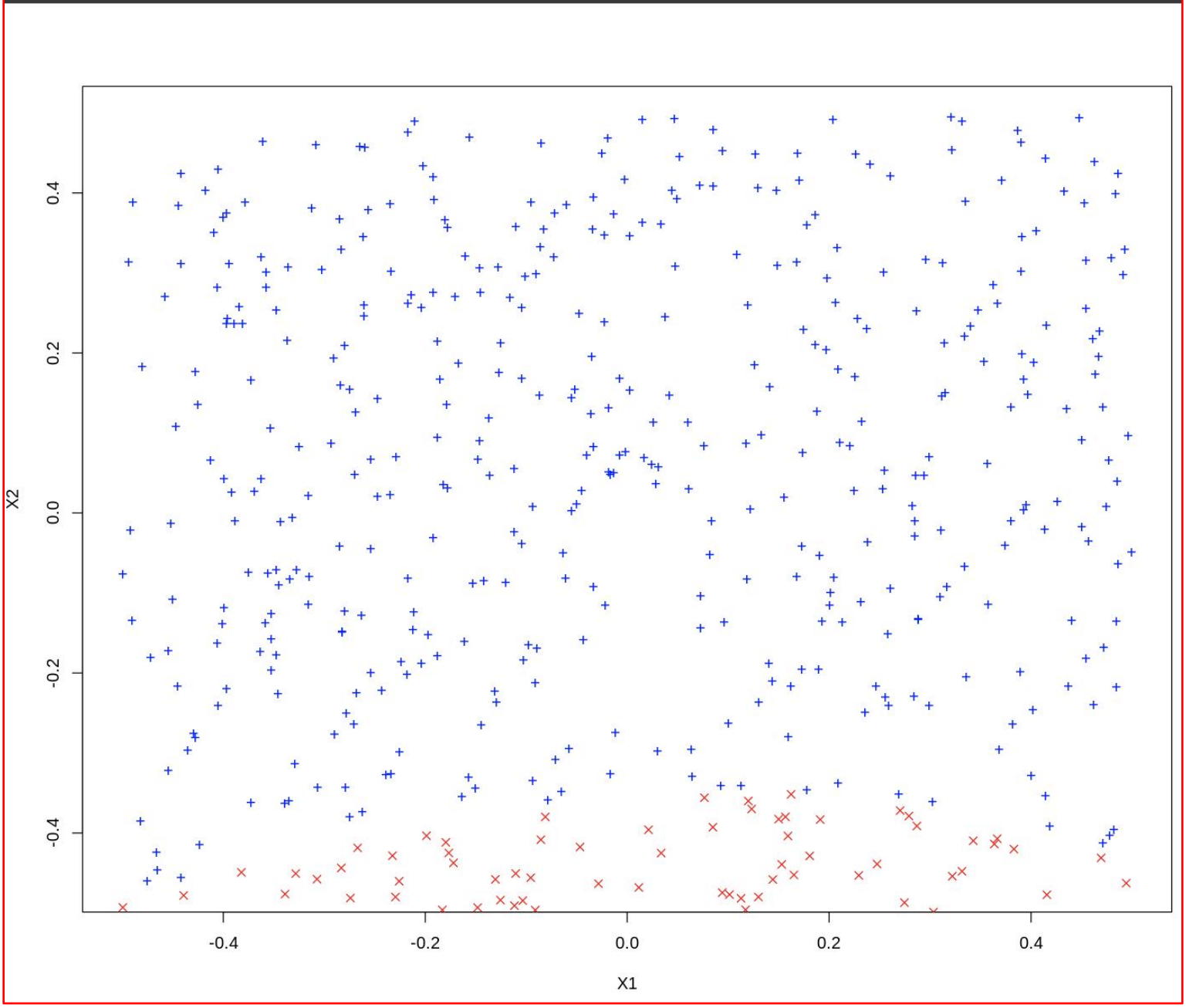
```
48.8
```

h. Fit a SVM using a non-linear kernel to the data. Obtain a class prediction for each training observation. Plot the observations, coloured according to the predicted class labels.

```r
# Polynomial SVM
svm_polynomial_fit=svm(as.factor(y) ~ x1 + x2,data=df,kernel="polynomial",cost=10)
summary(svm_polynomial_fit)
```

```
Call:
svm(formula = as.factor(y) ~ x1 + x2, data = df, kernel = "polynomial",
    cost = 10)


Parameters:
   SVM-Type:  C-classification
 SVM-Kernel:  polynomial
       cost:  10
     degree:  3
     coef.0:  0

Number of Support Vectors:  484

 ( 243 241 )


Number of Classes:  2

Levels:
 0 1
```

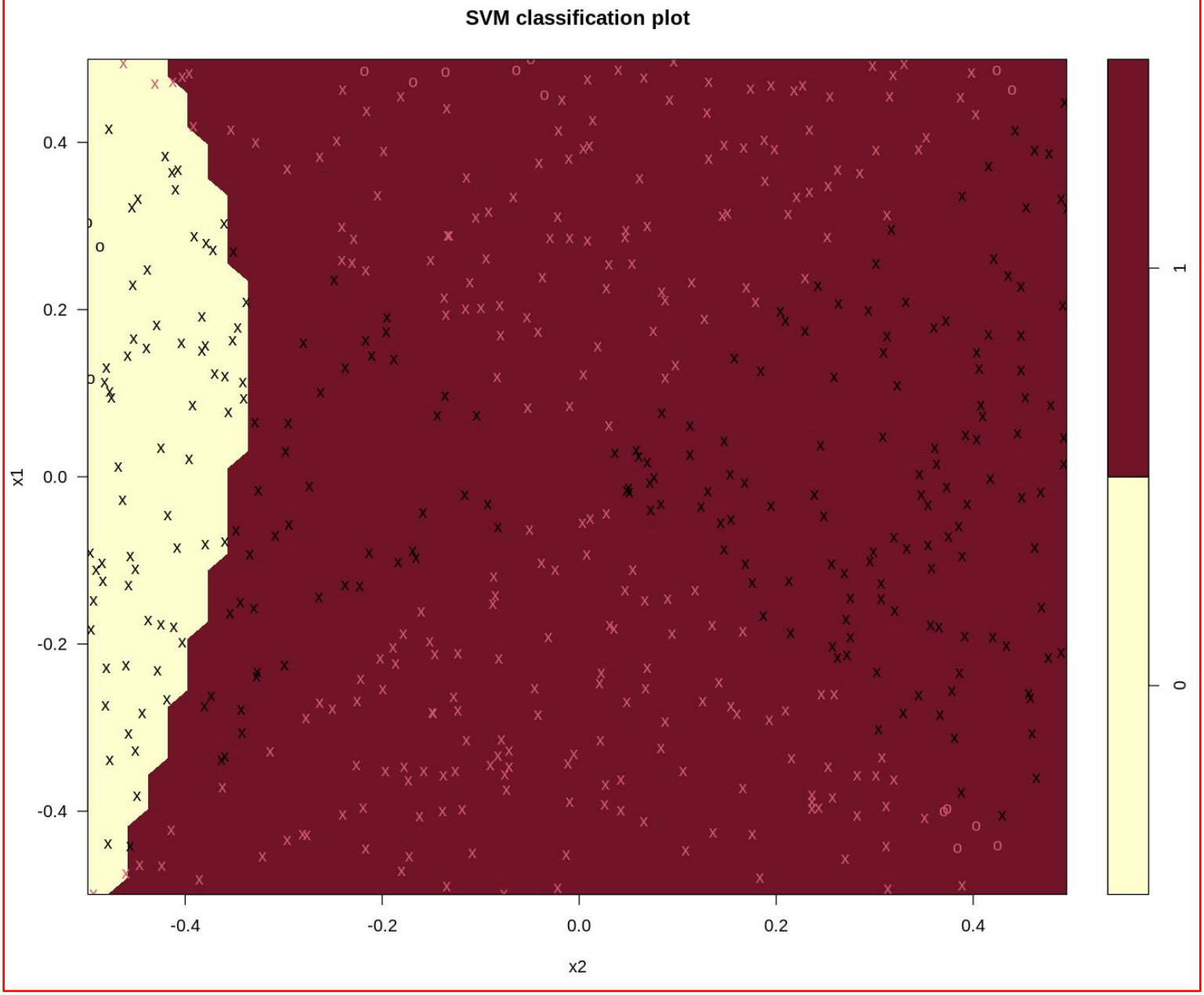
```r
options(repr.plot.width=12, repr.plot.height=10)

svm_polynomial_pred=predict(svm_polynomial_fit,df)
pos_data=df[svm_polynomial_pred == 1,]
neg_data=df[svm_polynomial_pred == 0,]
plot(pos_data$x1,pos_data$x2,col="blue",xlab="X1",ylab="X2",pch="+")
points(neg_data$x1,neg_data$x2,col="red",pch=4)
```



```
 1   svm_polynomial_fit
```

```
Call:
svm(formula = as.factor(y) ~ x1 + x2, data = df, kernel = "polynomial",
    cost = 10)


Parameters:
   SVM-Type:  C-classification
 SVM-Kernel:  polynomial
       cost:  10
     degree:  3
     coef.0:  0

Number of Support Vectors:  484
```

```r
options(repr.plot.width=12, repr.plot.height=10)

plot(svm_polynomial_fit,df)
```



```r
# Polynomial SVM Confusion Matrix - Train
print(table(df[,"y"],predict(svm_polynomial_fit,df)))
```

```
    0   1
0  63 181
1   3 253
```

```r
# Polynomial SVM Miss Classification Rate - Train
round((1 - ( (63 + 253) / (63 + 181 + 3 + 253)))*100,2)
```

```
36.8
```

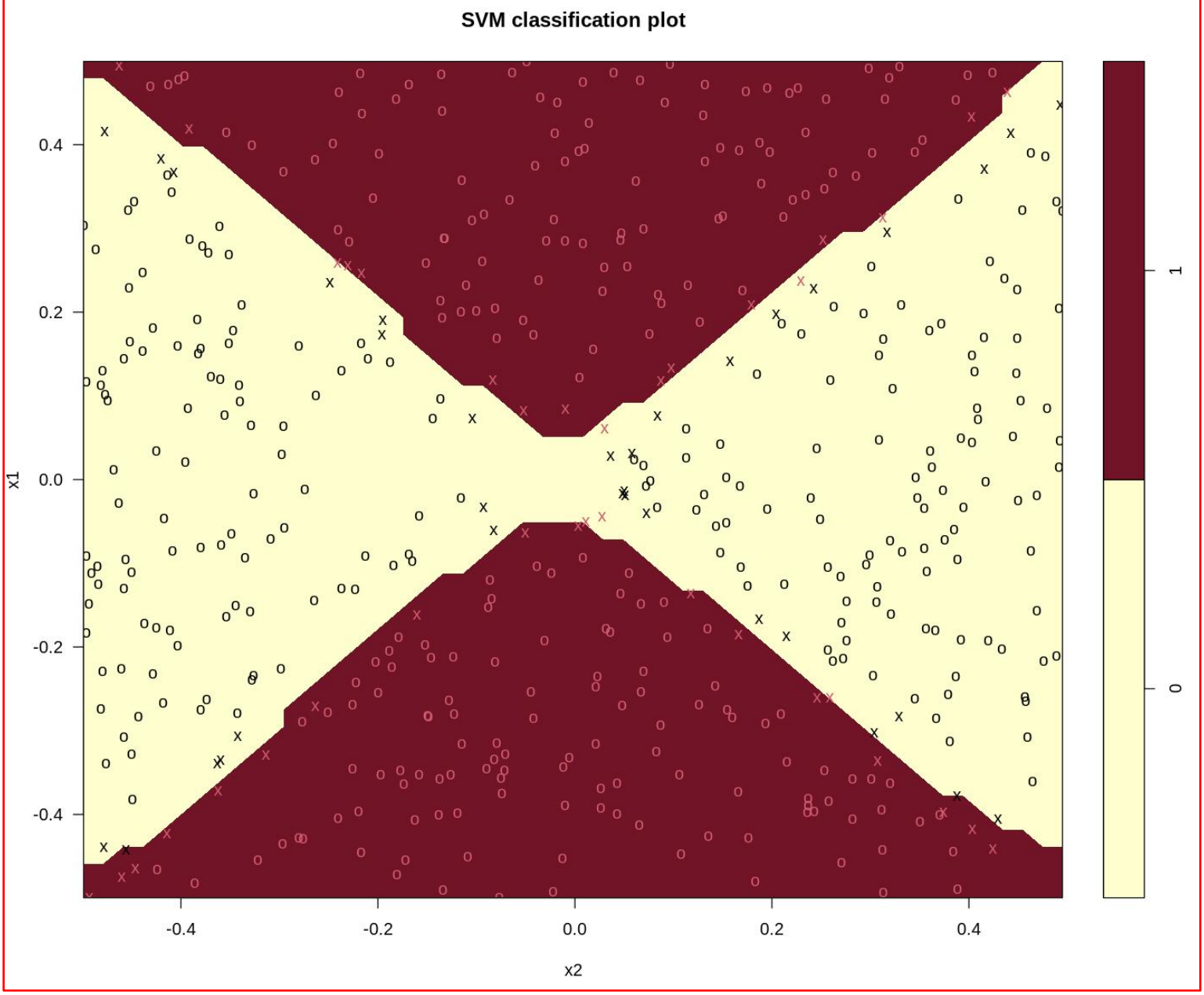
```r
# Radial SVM
svm_radial_fit=svm(as.factor(y) ~ x1 + x2,data=df,kernel="radial",cost=10,gamma=1)
summary(svm_radial_fit)
```

```
Call:
svm(formula = as.factor(y) ~ x1 + x2, data = df, kernel = "radial",
    cost = 10, gamma = 1)


Parameters:
   SVM-Type:  C-classification
 SVM-Kernel:  radial
       cost:  10

Number of Support Vectors:  71

 ( 37 34 )


Number of Classes:  2

Levels:
 0 1
```

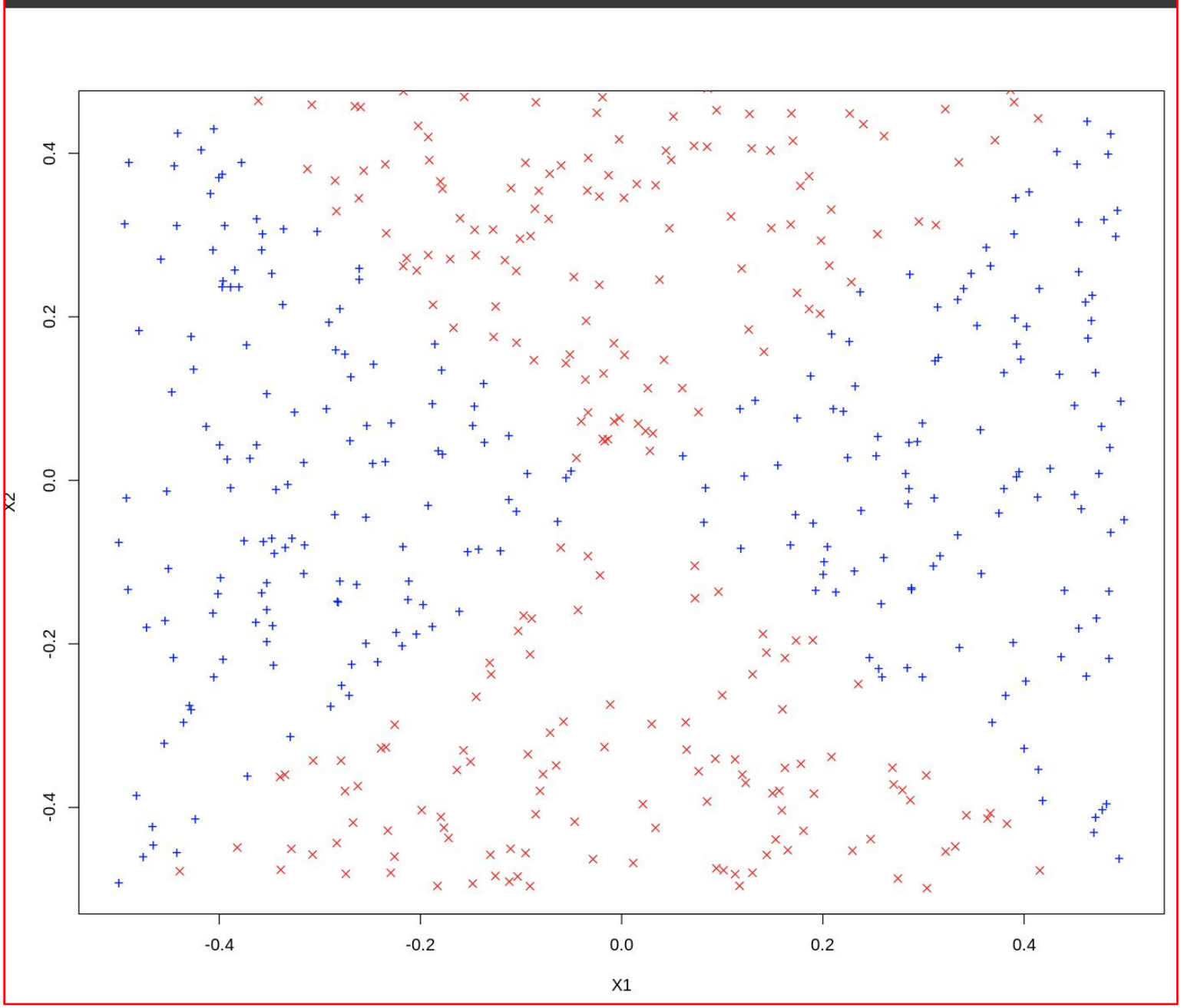
```
[ ]   1  svm_radial_fit
```

```
Call:
svm(formula = as.factor(y) ~ x1 + x2, data = df, kernel = "radial",
    cost = 10, gamma = 1)


Parameters:
   SVM-Type:  C-classification
 SVM-Kernel:  radial
       cost:  10

Number of Support Vectors:  71
```

```
1  options(repr.plot.width=12, repr.plot.height=10)
2
3  plot(svm_radial_fit,df)
```



```r
options(repr.plot.width=12, repr.plot.height=10)

svm_radial_pred=predict(svm_radial_fit,df)
pos_data=df[svm_radial_pred == 1,]
neg_data=df[svm_radial_pred == 0,]
plot(pos_data$x1,pos_data$x2,col="blue",xlab="X1",ylab="X2",pch="+")
points(neg_data$x1,neg_data$x2,col="red",pch=4)
```



```r
# Radial SVM Confusion Matrix - Train
print(table(df[,"y"],predict(svm_radial_fit,df)))
```

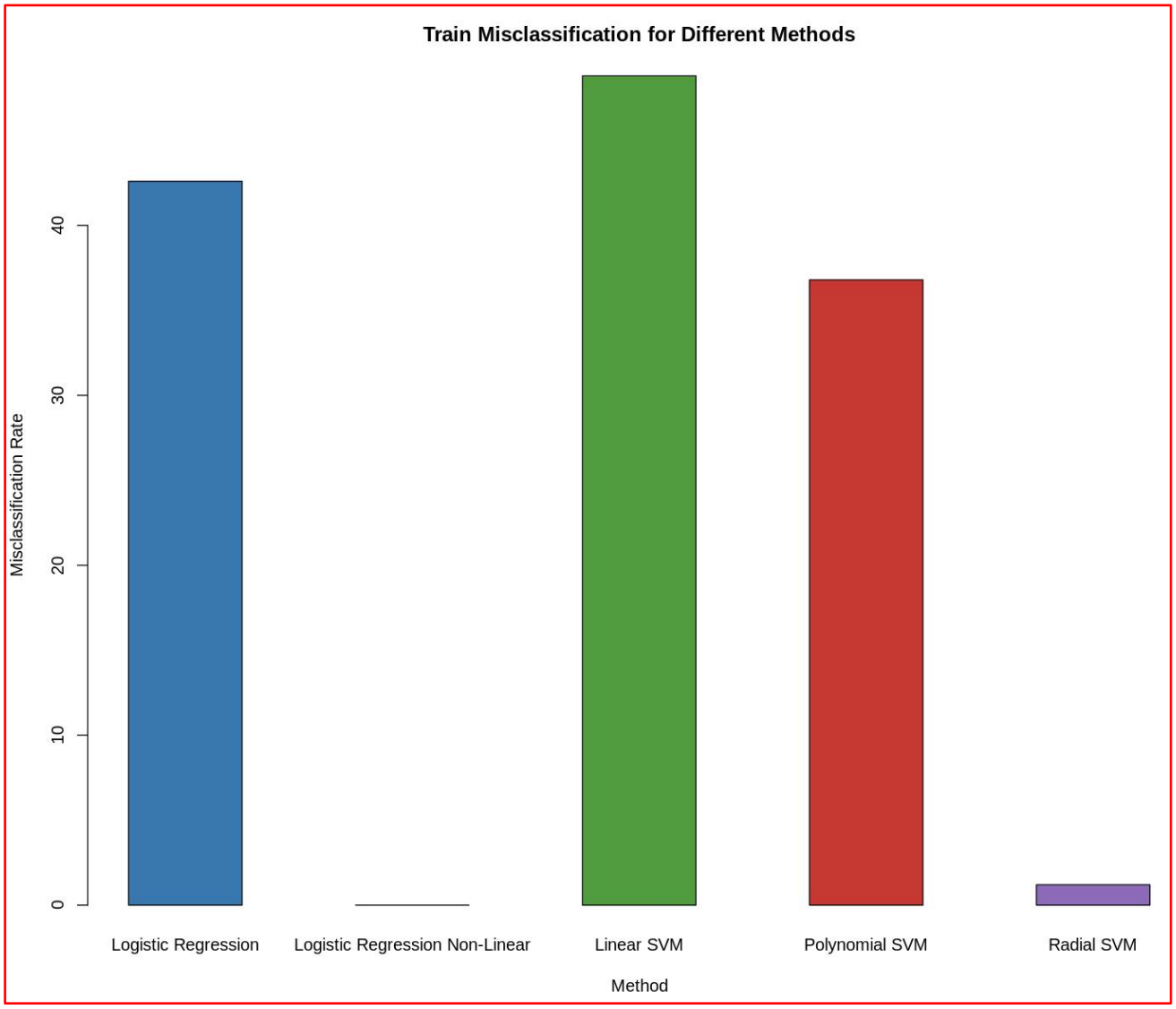
```
   0   1
0 240   4
1   2 254
```

```r
# Radial SVM Miss Classification Rate - Train
round((1 - ( (240 + 254) / (240 + 4 + 2 + 254)))*100,2)
```

```
1.2
```

i. Comment on your results.

```r
options(repr.plot.width=12, repr.plot.height=10)

svm_result_df=data.frame(
  Method = c("Logistic Regression", "Logistic Regression Non-Linear", "Linear SVM", "Polynomial SVM", "Radial SVM"),
  Train_Misclassification = c(42.6, 0, 48.8, 36.8, 1.2)
)

barplot(
  t(as.matrix(svm_result_df[,-1])),
  beside = TRUE,
  col = c("#1f77b4", "#ff7f0e", "#2ca02c", "#d62728", "#9467bd"),
  names.arg = svm_result_df$Method,
  xlab = "Method",
  ylab = "Misclassification Rate",
  main = "Train Misclassification for Different Methods",
)
```



The results demonstrate significant disparities in the performance of various classification methods. While logistic regression yields a relatively high training misclassification rate of 42.6%, the introduction of non-linear capabilities results in a dramatic improvement, achieving a perfect classification. Conversely, linear SVM exhibits a substantial misclassification rate of 48.8%, indicating its limitation in handling non-linear data. Polynomial SVM exhibits a comparatively lower misclassification rate of 36.8%, suggesting a partial success in capturing the underlying complexity. However, radial SVM emerges as the standout performer, boasting an impressively low misclassification rate of only 1.2%, indicating its robustness in effectively capturing non-linear patterns in the data. These findings underscore the importance of selecting appropriate classification methods, with radial SVM demonstrating superior efficacy in handling non-linear classification tasks.

6. Suppose that for a particular data set, we perform hierarchical clustering using single linkage and using complete linkage. We obtain two dendrograms.

a. At a certain point on the single linkage dendrogram, the clusters {1,2,3} and {4,5} fuse. On the complete linkage dendrogram, the clusters {1, 2, 3} and {4, 5} also fuse at a certain point. Which fusion will occur higher on the tree, or will they fuse at the same height, or is there not enough information to tell?

The comparison between single and complete linkage in hierarchical clustering hinges on the dissimilarity measures between clusters. If the maximal and minimal inter-cluster dissimilarities are equal, they would fuse at the same height in the dendrogram. Conversely, if they are unequal, single linkage dendrograms would fuse at a lower height. Generally, complete linkage fusion occurs higher on the dendrogram, as it utilizes the highest inter-cluster dissimilarity, while single linkage uses the lowest. For instance, in scenarios where inter-cluster dissimilarities differ, single linkage fusion would occur at a lower height than complete linkage fusion. However, in rare cases where all inter-cluster distances are identical, fusion would occur at the same height for both linkage methods. Thus we don't have enough information to conclude.

b. At a certain point on the single linkage dendrogram, the clusters {5} and {6} fuse. On the complete linkage dendrogram, the clusters {5} and {6} also fuse at a certain point. Which fusion will occur higher on the tree, or will they fuse at the same height, or is there not enough information to tell?

The fusion height remains consistent regardless of the linkage type, as it does not influence leaf-to-leaf fusion. For instance, if the dissimilarity between elements 5 and 6 is 2, both single and complete linkage methods would yield a dissimilarity of 2 between clusters {5} and {6}. Consequently, the fusion height for both single and complete linkage methods would be 2.

7. In this problem, you will generate simulated data, and then perform PCA and K-means clustering on the data.

(a) Generate a simulated data set with 20 observations in each of three classes (i.e. 60 observations total), and 50 variables. Hint: There are a number of functions in R that you can use to generate data. One example is the rnorm() function; runif() is another option. Be sure to add a mean shift to the observations in each class so that there are three distinct classes.

```r
install.packages("stats")
library(stats)
set.seed(123)
```

```
Installing package into '/usr/local/lib/R/site-library'
(as 'lib' is unspecified)

Warning message:
"package 'stats' is a base package, and should not be updated"
```

```r
x=matrix(rnorm(20*3*50,mean=0,sd=0.001),ncol=50)
x[1:20,2]=1
x[21:40,1]=2
x[21:40,2]=2
x[41:60,1]=1
```

(b) Perform PCA on the 60 observations and plot the first two principal component score vectors. Use a different color to indicate the observations in each of the three classes. If the three classes appear separated in this plot, then continue on to part (c). If not, then return to part (a) and modify the simulation so that there is greater separation between the three classes. Do not continue to part (c) until the three classes show at least some separation in the first two principal component score vectors.

```
1  pca_output=prcomp(x)
2  summary(pca_output)
```

```
Importance of components:
                          PC1    PC2      PC3      PC4      PC5      PC6
Standard deviation     1.008 0.5823 0.001858 0.001718 0.001669 0.001658
Proportion of Variance 0.750 0.2500 0.000000 0.000000 0.000000 0.000000
Cumulative Proportion  0.750 1.0000 0.999970 0.999970 0.999970 0.999970
                          PC7      PC8      PC9     PC10     PC11     PC12
Standard deviation     0.001627 0.001542 0.001463 0.001408 0.001356 0.001313
Proportion of Variance 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000
Cumulative Proportion  0.999980 0.999980 0.999980 0.999980 0.999980 0.999980
                          PC13     PC14     PC15     PC16     PC17     PC18
Standard deviation     0.001284 0.001257 0.001202 0.001139 0.001129 0.001104
Proportion of Variance 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000
Cumulative Proportion  0.999990 0.999990 0.999990 0.999990 0.999990 0.999990
                          PC19     PC20     PC21      PC22      PC23      PC24
Standard deviation     0.001079 0.001034 0.001004 0.0009466 0.0009253 0.0009016
Proportion of Variance 0.000000 0.000000 0.000000 0.0000000 0.0000000 0.0000000
Cumulative Proportion  0.999990 0.999990 0.999990 0.9999900 0.9999900 0.9999900
                          PC25      PC26      PC27     PC28      PC29  PC30
Standard deviation     0.000888 0.0008498 0.0007979 0.000785 0.0007489 7e-04
Proportion of Variance 0.000000 0.0000000 0.0000000 0.000000 0.0000000 0e+00
Cumulative Proportion  1.000000 1.0000000 1.0000000 1.000000 1.0000000 1e+00
                          PC31      PC32      PC33     PC34      PC35
Standard deviation     0.0006661 0.0006375 0.0006101 0.000573 0.0005366
Proportion of Variance 0.0000000 0.0000000 0.0000000 0.000000 0.0000000
Cumulative Proportion  1.0000000 1.0000000 1.0000000 1.000000 1.0000000
                          PC36      PC37      PC38      PC39      PC40
Standard deviation     0.000514 0.0005004 0.0004684 0.0004261 0.0003927
Proportion of Variance 0.000000 0.0000000 0.0000000 0.0000000 0.0000000
Cumulative Proportion  1.000000 1.0000000 1.0000000 1.0000000 1.0000000
                          PC41      PC42      PC43      PC44     PC45
Standard deviation     0.0003856 0.0003346 0.0003208 0.0002896 0.000284
Proportion of Variance 0.0000000 0.0000000 0.0000000 0.0000000 0.000000
Cumulative Proportion  1.0000000 1.0000000 1.0000000 1.0000000 1.000000
```

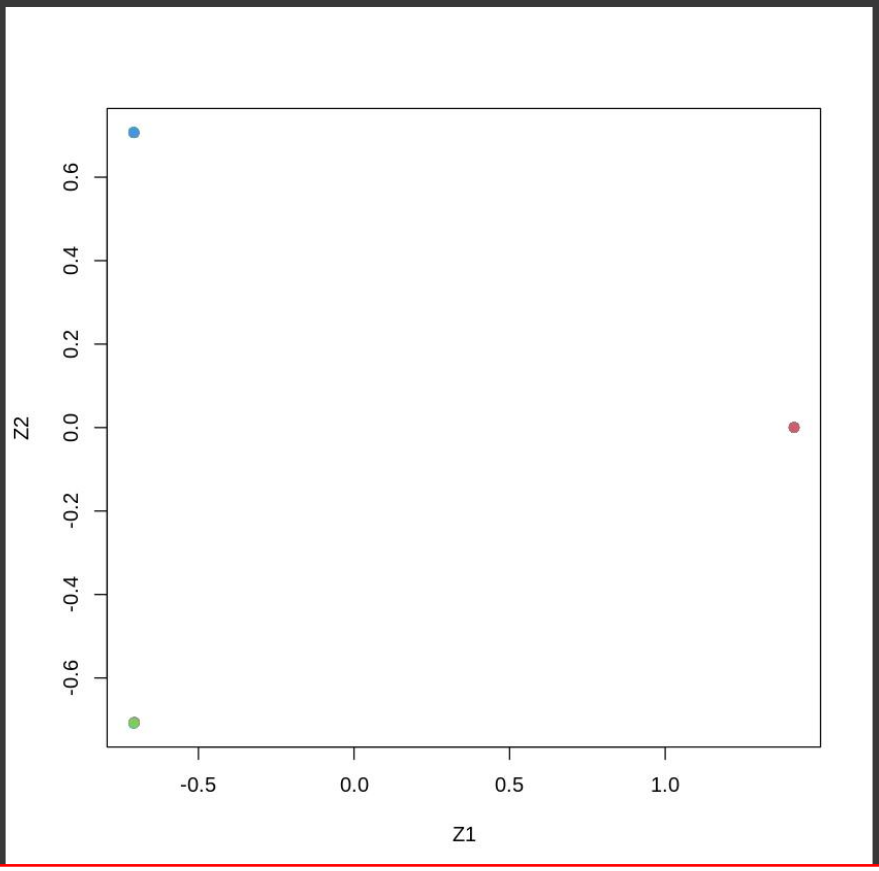
```
1  pca_output
```

```
[49,]  1.472843e-01 -2.181345e-01  4.162384e-01 -1.393111e-02  0.0955935527
[50,]  2.416638e-01  2.861609e-01  7.344346e-03 -1.657490e-02  0.0301805105
             PC41          PC42          PC43          PC44          PC45
 [1,] -0.0001273965  1.603868e-04  9.181062e-05  5.522836e-05  1.088094e-05
 [2,] -0.0001652326 -4.802871e-05  5.357907e-05 -1.881206e-04 -3.816625e-05
 [3,]  0.0035544749 -2.108886e-02  2.387166e-01 -2.679722e-01  2.797682e-01
 [4,]  0.0828550421  2.288922e-02 -1.776536e-01 -2.507106e-01  7.852908e-03
 [5,] -0.2904395669  1.329429e-01  1.633542e-01 -1.429205e-01  1.166329e-02
 [6,] -0.1749218255 -1.027397e-01  1.967315e-01  8.412620e-02  1.583300e-01
 [7,]  0.0538759984 -1.550940e-01 -2.947205e-02 -5.658009e-02  1.947613e-01
 [8,]  0.0553057955  1.864413e-01  7.849935e-02 -1.286133e-01  1.680161e-01
 [9,] -0.1224706695 -1.972228e-01  1.220613e-01  1.820397e-01  4.536159e-03
[10,]  0.1758603234  9.295717e-02 -1.641387e-01  9.206069e-02  2.168567e-01
[11,] -0.0728290317 -7.964263e-02  1.228486e-01 -1.422051e-01  4.785233e-02
[12,]  0.1170742067 -1.510748e-01  1.167342e-01  8.185501e-02 -1.082205e-03
[13,]  0.1316392525 -1.755286e-01  1.000752e-01  2.010658e-02 -3.276927e-02
[14,]  0.0011510455  9.320757e-02  1.610626e-01  1.029167e-01  1.602099e-01
[15,] -0.1657651571 -4.723072e-03  1.478264e-01 -1.857126e-01 -1.259986e-02
[16,]  0.0132093597  1.470681e-02 -1.665543e-02 -5.598501e-02 -1.648666e-02
[17,]  0.1255832295 -7.294007e-02  2.189627e-01 -8.260341e-03 -1.718693e-01
[18,]  0.0170132737  7.022441e-02  7.871280e-03  2.451584e-01  9.200838e-02
[19,]  0.0017596494  1.242559e-01 -1.459824e-01  2.400010e-01 -5.383550e-02
[20,] -0.0570191586  6.514504e-02  2.593203e-01 -2.192166e-01  1.681058e-01
[21,]  0.0197445672  1.946741e-02 -2.003759e-01  1.017992e-01  2.647741e-02
[22,] -0.3337980998  1.141689e-01  3.543708e-02  5.411823e-02 -1.173668e-01
[23,] -0.2805558702  1.077089e-02  4.994445e-02  4.652616e-02 -2.697060e-02
[24,]  0.2247152839 -2.367033e-01  1.400308e-01 -1.305303e-01 -8.429746e-02
[25,] -0.2601736380  1.409338e-01 -1.544106e-01 -5.546437e-02 -1.227135e-01
[26,]  0.0472872470  3.322513e-01  2.280773e-01  2.092134e-01 -1.669842e-01
[27,] -0.0710550209  1.171216e-02  1.913981e-01  1.662559e-01 -1.007583e-01
[28,]  0.0891371505  1.754111e-01  6.602707e-03 -1.950575e-01  2.011934e-01
[29,]  0.0800557049 -8.808969e-02 -4.394050e-02  4.154848e-02  2.522853e-02
[30,] -0.2680329799 -1.487933e-01  4.207415e-02  1.307147e-01  2.799342e-01
[31,]  0.2386478162 -3.124306e-02  1.236844e-01  1.230003e-01 -2.132352e-01
[32,]  0.2214951060  9.874590e-02  1.461337e-01 -1.073191e-01 -1.535480e-01
[33,]  0.1243398295 -1.248308e-01 -1.928614e-01  1.696619e-01  1.696813e-01
[34,]  0.0881634116  3.685416e-02  2.903489e-01  2.140226e-01 -1.198671e-01
[35,] -0.0905470286 -3.010656e-01  1.411331e-01  3.144756e-01  1.677008e-01
[36,] -0.0147207419 -4.307095e-02 -1.169742e-01  7.138123e-02 -3.341744e-01
```

```
1  pca_output$x[,1:2]
```

1.4142644	0.0002359989
1.4142662	0.0002380780
1.4142662	0.0002346925
1.4142647	0.0002384790
1.4142639	0.0002370604
1.4142658	0.0002348066
1.4142654	0.0002359457
1.4142664	0.0002353647
1.4142648	0.0002379650
1.4142661	0.0002362209
1.4142654	0.0002357503
1.4142647	0.0002343127
-0.7077367	0.7073119184
-0.7070506	0.7066309591
-0.7074081	0.7069864577
-0.7074770	0.7070569866
-0.7079057	0.7074812089
-0.7072647	0.7068465757
-0.7077880	0.7073689812
-0.7084134	0.7079909255
-0.7075026	0.7070842706
-0.7065831	0.7061632066
-0.7076392	0.7072183339

```
1  plot(pca_output$x[,1:2],col=2:4,xlab="Z1",ylab="Z2",pch=19)
```



(c) Perform K-means clustering of the observations with K = 3. How well do the clusters that you obtained in K-means clustering compare to the true class labels? Hint: You can use the table() function in R to compare the true class labels to the class labels obtained by clustering. Be careful how you interpret the results: K-means clustering will arbitrarily number the clusters, so you cannot simply check whether the true class labels and clustering labels are the same.

```
1  km_3_output=kmeans(x,3,nstart=20)
2  summary(km_3_output)
```

```
            Length Class  Mode
cluster      60    -none- numeric
centers     150    -none- numeric
totss         1    -none- numeric
withinss      3    -none- numeric
tot.withinss  1    -none- numeric
betweenss     1    -none- numeric
size          3    -none- numeric
iter          1    -none- numeric
ifault        1    -none- numeric
```

```
1  km_3_output
```

```
K-means clustering with 3 clusters of sizes 20, 20, 20

Cluster means:
        [,1]          [,2]          [,3]          [,4]          [,5]
1 2.0000000000  2.0000000000 -0.0001733363  5.948306e-05  2.094816e-04
2 0.0001416238  1.0000000000 -0.0000383492 -1.335579e-04 -2.324115e-05
3 1.0000000000 -0.0003593805  0.0001668898 -3.834048e-05  3.949425e-04
          [,6]          [,7]          [,8]          [,9]         [,10]
1  5.021726e-05 -1.466458e-04 -3.640754e-05 -0.0002754007 -0.0000406936
2 -3.901232e-05  2.858593e-05  2.065840e-04  0.0002698108  0.0001010186
3 -7.425957e-05 -2.119631e-04  3.012304e-04  0.0002521723 -0.0002485964
         [,11]         [,12]         [,13]         [,14]         [,15]
1 -0.0002010846 -1.507986e-04 4.064904e-04  1.874406e-05 3.797888e-04
2 -0.0002707735 -1.617905e-04 8.075581e-05 -1.449059e-04 3.072856e-04
3  0.0000362264  6.499619e-05 1.220312e-04 -2.415699e-04 3.700029e-06
         [,16]         [,17]         [,18]         [,19]         [,20]
1  0.0001246399  0.0000708230 3.012403e-05 -1.261826e-06 -1.453205e-04
2  0.0002428251 -0.0003194841 2.410900e-04  3.765989e-05 -8.410132e-05
3 -0.0002147676  0.0002532857 3.754321e-05  6.940426e-06  1.278654e-04
         [,21]         [,22]         [,23]         [,24]         [,25]
1 -0.0002079398 -0.0001931771 -2.101255e-04  1.416406e-04 -0.0003238265
```

```
          [,36]         [,37]         [,38]         [,39]         [,40]
1 -1.055675e-04  1.095263e-05 -5.023213e-04  1.540143e-04  4.078055e-04
2 -7.120729e-05 -3.062049e-04 -2.855283e-05 -8.749326e-05 -2.487604e-04
3 -1.923419e-04  2.756327e-04 -1.766454e-04  4.525575e-05  6.836571e-06
          [,41]         [,42]         [,43]         [,44]         [,45]
1 -7.353657e-05 2.782470e-04 -1.283480e-04 -0.0001951666 -1.851377e-04
2 -5.217887e-04 1.072581e-05  3.434313e-05  0.0001092817  5.919534e-06
3  1.160558e-04 1.996153e-04  3.155459e-04  0.0002854219 -4.860297e-04
          [,46]         [,47]         [,48]         [,49]         [,50]
1  0.0002714904  1.523068e-04  0.0001627394 -1.613143e-04 0.0002974274
2 -0.0003487172  2.042875e-05  0.0001321823 -4.342839e-06 0.0005034433
3 -0.0001714830 -5.160566e-06 -0.0000476817 -3.795508e-04 0.0002181973

Clustering vector:
 [1] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
[39] 1 1 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3

Within cluster sum of squares by cluster:
[1] 0.0009679969 0.0008927647 0.0009002234
 (between_SS / total_SS = 100.0 %)

Available components:

[1] "cluster"      "centers"      "totss"        "withinss"     "tot.withinss"
[6] "betweenss"    "size"         "iter"         "ifault"
```

```
1  table(km_3_output$cluster,c(rep(1,20),rep(2,20),rep(3,20)))
```

```
   1  2  3
1  0 20  0
2 20  0  0
3  0  0 20
```

(d) Perform K-means clustering with K = 2. Describe your results.

```
1  km_2_output=kmeans(x,2,nstart=20)
2  summary(km_2_output)
```

```
            Length Class  Mode
cluster      60    -none- numeric
centers     100    -none- numeric
totss         1    -none- numeric
withinss      2    -none- numeric
tot.withinss  1    -none- numeric
betweenss     1    -none- numeric
size          2    -none- numeric
iter          1    -none- numeric
ifault        1    -none- numeric
```

```
1  km_2_output
```

```
K-means clustering with 2 clusters of sizes 20, 40

Cluster means:
       [,1]      [,2]          [,3]          [,4]         [,5]          [,6]
1 2.0000000 2.0000000 -1.733363e-04  5.948306e-05 0.0002094816  5.021726e-05
2 0.5000708 0.4998203  6.427032e-05 -8.594918e-05 0.0001858507 -5.663594e-05
          [,7]          [,8]          [,9]         [,10]         [,11]
1 -1.466458e-04 -3.640754e-05 -0.0002754007 -4.069360e-05 -0.0002010846
2 -9.168858e-05  2.539072e-04  0.0002609916 -7.378892e-05 -0.0001172735
         [,12]         [,13]         [,14]         [,15]         [,16]
1 -1.507986e-04 0.0004064904  1.874406e-05 0.0003797888 1.246399e-04
2 -4.839714e-05 0.0001013935 -1.932379e-04 0.0001554928 1.402874e-05
         [,17]         [,18]         [,19]         [,20]         [,21]
1  7.082300e-05 3.012403e-05 -1.261826e-06 -1.453205e-04 -2.079398e-04
2 -3.309919e-05 1.393166e-04  2.230016e-05  2.188203e-05  4.317470e-06
         [,22]         [,23]         [,24]         [,25]         [,26]
1 -0.0001931771 -2.101255e-04  1.416406e-04 -0.0003238265 0.0001961571
2  0.0001189212 -1.379425e-05 -9.701470e-07  0.0003638367 0.0001105868
         [,27]         [,28]         [,29]         [,30]         [,31]
1 -1.663790e-04 -0.0002497265  0.0001767372 8.472923e-05 -3.255986e-05
2  9.538168e-05  0.0003047045 -0.0002223341 7.984142e-05 -7.498187e-06
```

```
2  0.0001189212 -1.379425e-05 -9.701470e-07  0.0003638367 0.0001105868
          [,27]         [,28]         [,29]         [,30]         [,31]
1 -1.663790e-04 -0.0002497265  0.0001767372 8.472923e-05 -3.255986e-05
2  9.538168e-05  0.0003047045 -0.0002223341 7.984142e-05 -7.498187e-06
          [,32]         [,33]         [,34]         [,35]         [,36]
1 4.255289e-05  0.0003614494 -1.968976e-05 -0.0003022201 -0.0001055675
2 2.427007e-04 -0.0001177294  1.336535e-04 -0.0002267818 -0.0001317746
          [,37]         [,38]         [,39]         [,40]         [,41]
1  1.095263e-05 -0.0005023213  1.540143e-04  0.0004078055 -7.353657e-05
2 -1.528607e-05 -0.0001025991 -2.111875e-05 -0.0001209619 -2.028664e-04
          [,42]         [,43]         [,44]         [,45]         [,46]
1 0.0002782470 -0.0001283480 -0.0001951666 -0.0001851377  0.0002714904
2 0.0001051706  0.0001749445  0.0001973518 -0.0002400551 -0.0002601001
          [,47]         [,48]         [,49]         [,50]
1 1.523068e-04 0.0001627394 -0.0001613143 0.0002974274
2 7.634092e-06 0.0000422503 -0.0001919468 0.0003608203

Clustering vector:
 [1] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
[39] 1 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2

Within cluster sum of squares by cluster:
[1] 9.679969e-04 2.000619e+01
 (between_SS / total_SS =  75.0 %)

Available components:

[1] "cluster"      "centers"      "totss"        "withinss"     "tot.withinss"
[6] "betweenss"    "size"         "iter"         "ifault"
```

```
[ ]  1  table(km_2_output$cluster,c(rep(1,20),rep(2,20),rep(3,20)))
```

```
   1  2  3
1  0 20  0
2 20  0 20
```

The final results indicate the distribution of observations from three groups among the two clusters generated by k-means clustering. Specifically, for the first group of 20 observations, all are assigned to Cluster 2; for the second group, all 20 observations are assigned to Cluster 1; and for the third group, all 20 observations are again assigned to Cluster 2.

(e) Now perform K-means clustering with K = 4, and describe your results.

```
km_4_output=kmeans(x,4,nstart=20)
summary(km_4_output)
```

```
            Length Class  Mode
cluster      60    -none- numeric
centers     200    -none- numeric
totss         1    -none- numeric
withinss      4    -none- numeric
tot.withinss  1    -none- numeric
betweenss     1    -none- numeric
size          4    -none- numeric
iter          1    -none- numeric
ifault        1    -none- numeric
```

```
km_4_output
```

```
K-means clustering with 4 clusters of sizes 8, 20, 12, 20

Cluster means:
         [,1]          [,2]          [,3]          [,4]          [,5]
1 1.0000000000 -0.0004160859  0.0008703514  2.226209e-04 -7.325791e-06
2 2.0000000000  2.0000000000 -0.0001733363  5.948306e-05  2.094816e-04
3 1.0000000000 -0.0003215768 -0.0003020846 -2.123147e-04  6.631214e-04
4 0.0001416238  1.0000000000 -0.0000383492 -1.335579e-04 -2.324115e-05
          [,6]          [,7]          [,8]          [,9]         [,10]
1 -9.341944e-04 -5.676621e-05 -1.140556e-04 -0.0001705292  0.0001393440
2  5.021726e-05 -1.466458e-04 -3.640754e-05 -0.0002754007 -0.0000406936
3  4.990303e-04 -3.154277e-04  5.780878e-04  0.0005339734 -0.0005072234
4 -3.901232e-05  2.858593e-05  2.065840e-04  0.0002698108  0.0001010186
         [,11]         [,12]         [,13]         [,14]         [,15]
1  0.0003553200  7.848231e-05 -3.058524e-04 -3.190501e-06  0.0002758207
2 -0.0002010846 -1.507986e-04  4.064904e-04  1.874406e-05  0.0003797888
3 -0.0001765026  5.600545e-05  4.072869e-04 -4.004895e-04 -0.0001777137
4 -0.0002707735 -1.617905e-04  8.075581e-05 -1.449059e-04  0.0003072856
         [,16]         [,17]         [,18]         [,19]         [,20]
1 -0.0003424324  0.0003374790 -3.037865e-04  6.951741e-04  9.001935e-04
2  0.0001246399  0.0000708230  3.012403e-05 -1.261826e-06 -1.453205e-04
3 -0.0001296577  0.0001971569  2.650964e-04 -4.518820e-04 -3.870200e-04
4  0.0002428251 -0.0003194841  2.410900e-04  3.765989e-05 -8.410132e-05
1  5.581364e-04 -4.238123e-04  4.405196e-05  5.742555e-04 -4.957668e-04
2 -3.255986e-05  4.255289e-05  3.614494e-04 -1.968976e-05 -3.022201e-04
3 -4.117185e-04  5.400034e-04  1.892759e-05 -7.728613e-05 -4.402116e-04
4  8.780186e-06  3.309242e-04 -2.644361e-04  8.397655e-05  8.870102e-06
          [,36]         [,37]         [,38]         [,39]         [,40]
1 -2.368107e-05 -6.085326e-05 -2.492186e-04  7.377953e-05  0.0006076137
2 -1.055675e-04  1.095263e-05 -5.023213e-04  1.540143e-04  0.0004078055
3 -3.047825e-04  4.999567e-04 -1.282633e-04  2.623991e-05 -0.0003936815
4 -7.120729e-05 -3.062049e-04 -2.855283e-05 -8.749326e-05 -0.0002487604
          [,41]         [,42]         [,43]         [,44]         [,45]
1  3.157409e-04 -1.966772e-04  5.322304e-04 -0.0002063019 -6.366741e-04
2 -7.353657e-05  2.782470e-04 -1.283480e-04 -0.0001951666 -1.851377e-04
3 -1.706756e-05  4.638103e-04  1.710895e-04  0.0006132377 -3.856001e-04
4 -5.217887e-04  1.072581e-05  3.434313e-05  0.0001092817  5.919534e-06
          [,46]         [,47]         [,48]         [,49]         [,50]
1  0.0007536535  3.038796e-05 -3.958519e-05 -7.548785e-04 -0.0003595711
2  0.0002714904  1.523068e-04  1.627394e-04 -1.613143e-04  0.0002974274
3 -0.0007882407 -2.885959e-05 -5.307937e-05 -1.293324e-04  0.0006033762
4 -0.0003487172  2.042875e-05  1.321823e-04 -4.342839e-06  0.0005034433

Clustering vector:
 [1] 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
[39] 2 2 3 3 3 1 3 1 3 3 1 3 3 1 1 1 3 3 3 1 1 3

Within cluster sum of squares by cluster:
[1] 0.0003119594 0.0009679969 0.0004849360 0.0008927647
 (between_SS / total_SS = 100.0 %)

Available components:

[1] "cluster"      "centers"      "totss"        "withinss"     "tot.withinss"
[6] "betweenss"    "size"         "iter"         "ifault"
```

```
table(km_4_output$cluster,c(rep(1,20),rep(2,20),rep(3,20)))
```

```
    1  2  3
 1  0  0  8
 2  0 20  0
 3  0  0 12
 4 20  0  0
```

All 20 observations belonging to Group 1 are assigned to Cluster 4, indicating a clear separation from the other groups. Similarly, all 20 observations from Group 2 are clustered together in Cluster 2, demonstrating a distinct grouping. Group 3 exhibits a split, with eight observations assigned to Cluster 1 and the remaining 12 assigned to Cluster 3. This distribution suggests that while Group 3 shares similarities with both Clusters 1 and 3, the majority aligns more closely with Cluster 3.

(f) Now perform K-means clustering with K = 3 on the first two principal component score vectors, rather than on the raw data. That is, perform K-means clustering on the 60 × 2 matrix of which the first column is the first principal component score vector, and the second column is the second principal component score vector. Comment on the results.

```
1  km_3_pca_output=kmeans(pca_output$x[,1:2],3,nstart=20)
2  summary(km_3_pca_output)
```

```
            Length Class  Mode
cluster     60     -none- numeric
centers      6     -none- numeric
totss        1     -none- numeric
withinss     3     -none- numeric
tot.withinss 1     -none- numeric
betweenss    1     -none- numeric
size         3     -none- numeric
iter         1     -none- numeric
ifault       1     -none- numeric
```

```
1  km_3_pca_output
```

```
K-means clustering with 3 clusters of sizes 20, 20, 20

Cluster means:
         PC1          PC2
1 -0.7067785 -0.707302439
2  1.4142654  0.000236067
3 -0.7074869  0.707066372

Clustering vector:
 [1] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
[39] 2 2 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3

Within cluster sum of squares by cluster:
[1] 1.797711e-05 4.571144e-11 9.323590e-06
 (between_SS / total_SS = 100.0 %)

Available components:

[1] "cluster"      "centers"      "totss"        "withinss"     "tot.withinss"
[6] "betweenss"    "size"         "iter"         "ifault"
```

```
1  table(km_3_pca_output$cluster,c(rep(1,20),rep(2,20),rep(3,20)))
```

```
   1  2  3
1 20  0  0
2  0 20  0
3  0  0 20
```

Each group, consisting of 20 observations, is perfectly partitioned into separate clusters, with all observations from Group 1 allocated to Cluster 1, those from Group 2 to Cluster 2, and those from Group 3 to Cluster 3. This clear separation indicates distinct clustering patterns among the groups, suggesting that the principal components successfully capture the underlying structure of the data, allowing for effective discrimination between the groups in the reduced-dimensional space. Overall, the results demonstrate the efficacy of k-means clustering in identifying meaningful patterns within the dataset, particularly when applied to principal components.

(g) Using the scale() function, perform K-means clustering with K = 3 on the data after scaling each variable to have standard deviation one. How do these results compare to those obtained in (b)? Explain.

```
km_scaled_3_output=kmeans(scale(x),3,nstart=20)
summary(km_scaled_3_output)
```

```
            Length Class  Mode
cluster      60    -none- numeric
centers     150    -none- numeric
totss         1    -none- numeric
withinss      3    -none- numeric
tot.withinss  1    -none- numeric
betweenss     1    -none- numeric
size          3    -none- numeric
iter          1    -none- numeric
ifault        1    -none- numeric
```

```
km_scaled_3_output
```

```
K-means clustering with 3 clusters of sizes 13, 24, 23

Cluster means:
        [,1]       [,2]       [,3]        [,4]        [,5]       [,6]
1 -0.1869073 -0.4670342 -0.0285322  0.20244350 -0.18394432 -0.3212735
2 -0.1516965 -0.1518502 -0.1369082 -0.04466254  0.15708110  0.3330718
3  0.2639352  0.4224282  0.1589876 -0.06782020 -0.05994219 -0.1659638
         [,7]       [,8]       [,9]      [,10]      [,11]      [,12]      [,13]
1 -0.67316523  0.2439458  0.2657201 -0.3041286  0.1158625  0.4382480  0.1683716
2 -0.01639648  0.1911209  0.3948557 -0.1044346 -0.2412934 -0.1295240  0.3053698
3  0.39759407 -0.3373130 -0.5622130  0.2808740  0.1862969 -0.1125499 -0.4138133
       [,14]      [,15]      [,16]       [,17]       [,18]      [,19]
1 -0.2930064 -0.4564940 -0.6786133 -0.23562965  0.04055868  0.6504871
2 -0.2472242 -0.4282816  0.5007142 -0.09925188  0.31503058 -0.6304167
3  0.4235853  0.7049209 -0.1389204  0.23674915 -0.35165204  0.2901595
       [,20]      [,21]       [,22]       [,23]      [,24]       [,25]
1  0.2894780  0.6753737  0.20820989  0.03343117  0.1762375  0.02069088
2 -0.3978353 -0.2409234 -0.09480842  0.06760991 -0.4230172 -0.01441091
3  0.2515145 -0.1303347 -0.01875333 -0.08944535  0.3417967  0.00334262
       [,26]       [,27]      [,28]       [,29]      [,30]      [,31]
1 -0.80457825  0.03574639  0.1357881 -0.37355760  0.1102672 -0.3676296
2  0.39633499  0.09311017  0.1747341  0.02286465  0.1556669 -0.2405551
       [,32]      [,33]      [,34]       [,35]       [,36]       [,37]
1 -0.2301085 -0.1395561  0.7511026 -0.349842446 -0.59228214 -0.80223443
2  0.3943399 -0.2221726 -0.2666044 -0.009351003 -0.01330933  0.39104351
3 -0.2814237  0.3107118 -0.1463404  0.207494603  0.34865616  0.04539145
       [,38]       [,39]       [,40]       [,41]      [,42]       [,43]
1 -0.28648658 -0.22941585 -0.40136716  0.31249720 -0.4402693  0.42587534
2  0.21609400  0.16222905  0.06679687 -0.23333816  0.3409348  0.07942395
3 -0.06356219 -0.03961266  0.15715862  0.06685444 -0.1069102 -0.32358931
       [,44]      [,45]      [,46]       [,47]       [,48]      [,49]
1  0.15008746 -0.6695473 -0.0337734 -0.86271451 -0.01692805 -0.3561713
2 -0.06323097  0.1392049 -0.2548993  0.45087106  0.21374870  0.5810751
3 -0.01885190  0.2331825  0.2850712  0.01714709 -0.21347409 -0.4050250
       [,50]
1  0.5141528
2  0.1267568
3 -0.4228760

Clustering vector:
 [1] 1 3 1 2 3 2 2 1 2 2 3 1 3 3 3 2 2 3 2 2 3 3 2 2 3 3 2 3 2 3 2 1 1 3 3 3 3 3
[39] 3 2 2 1 1 3 2 1 2 2 1 2 2 3 3 3 2 1 2 1 1 2

Within cluster sum of squares by cluster:
[1]  527.1295 1050.4685 1090.5983
 (between_SS / total_SS =   9.6 %)

Available components:

[1] "cluster"      "centers"      "totss"        "withinss"     "tot.withinss"
[6] "betweenss"    "size"         "iter"         "ifault"
```

```
table(km_scaled_3_output$cluster,c(rep(1,20),rep(2,20),rep(3,20)))
```

```
   1  2  3
1  4  2  7
2  9  6  9
3  7 12  4
```

Each row represents one of the three clusters generated by the algorithm, while the columns correspond to the three groups of observations in the original dataset. The numbers in the table indicate the count of observations from each group assigned to each cluster. Cluster 1 predominantly contains observations from Group 3, with a few from Group 1 and Group 2. Cluster 2 consists a few of observations from Group 2, with a balanced representation from Groups 1 and 3. In contrast, Cluster 3 is dominated by observations from Group 2, with a substantial number from Group 1 and a smaller representation from Group 3. These findings suggest that the clustering algorithm has effectively partitioned the data into distinct groups based on similarities in the scaled features, with each cluster exhibiting varying degrees of association with the original groups.