project2_harinris

Harin Rishabh

2023-12-02

```
library(tree)
library(ggplot2)
library(e1071)
library(boot)
library(caret)
```

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

```
##
## Attaching package: 'lattice'
```

```
## The following object is masked from 'package:boot':
##
##     melanoma
```

```
library(randomForest)
```

```
## randomForest 4.7-1.1
```

```
## Type rfNews() to see new features/changes/bug fixes.
```

```
##
## Attaching package: 'randomForest'
```

```
## The following object is masked from 'package:ggplot2':
##
##     margin
```

```
library(polycor)
```

First we read the .txt file into a table. The delimiter is a comma. Then we provide the columns with names for convenience.

```
set.seed(1)
data = read.table("project2.txt", sep= ",")
colnames(data)[1] = "x1"
colnames(data)[2] = "x2"
colnames(data)[3] = "x3"
colnames(data)[4] = "x4"
colnames(data)[5] = "y"
```

Checking for null values in data set. Looks like we have none.

```
colSums(is.na(data))
```

```
## x1 x2 x3 x4  y
##  0  0  0  0  0
```

```
dim(data)
```

```
## [1] 1372    5
```

Next, we encode the categorical response variable. Since we have only 0 and 1(binary) in the y column, it is not required as R libraries treat it as such. But we do it anyway to be sure. Now, we look at a summary of our data set. We can see that only y column is shown as frequency of occurrences of 0 and 1.

```
data$y <- as.factor(data$y)
summary(data)
```

```
##        x1                x2                x3                x4
##  Min.   :-7.0421   Min.   :-13.773   Min.   :-5.2861   Min.   :-8.5482
##  1st Qu.:-1.7730   1st Qu.: -1.708   1st Qu.:-1.5750   1st Qu.:-2.4135
##  Median : 0.4962   Median :  2.320   Median : 0.6166   Median :-0.5867
##  Mean   : 0.4337   Mean   :  1.922   Mean   : 1.3976   Mean   :-1.1917
##  3rd Qu.: 2.8215   3rd Qu.:  6.815   3rd Qu.: 3.1793   3rd Qu.: 0.3948
##  Max.   : 6.8248   Max.   : 12.952   Max.   :17.9274   Max.   : 2.4495
##  y
##  0:762
##  1:610
##
##
##
##
```

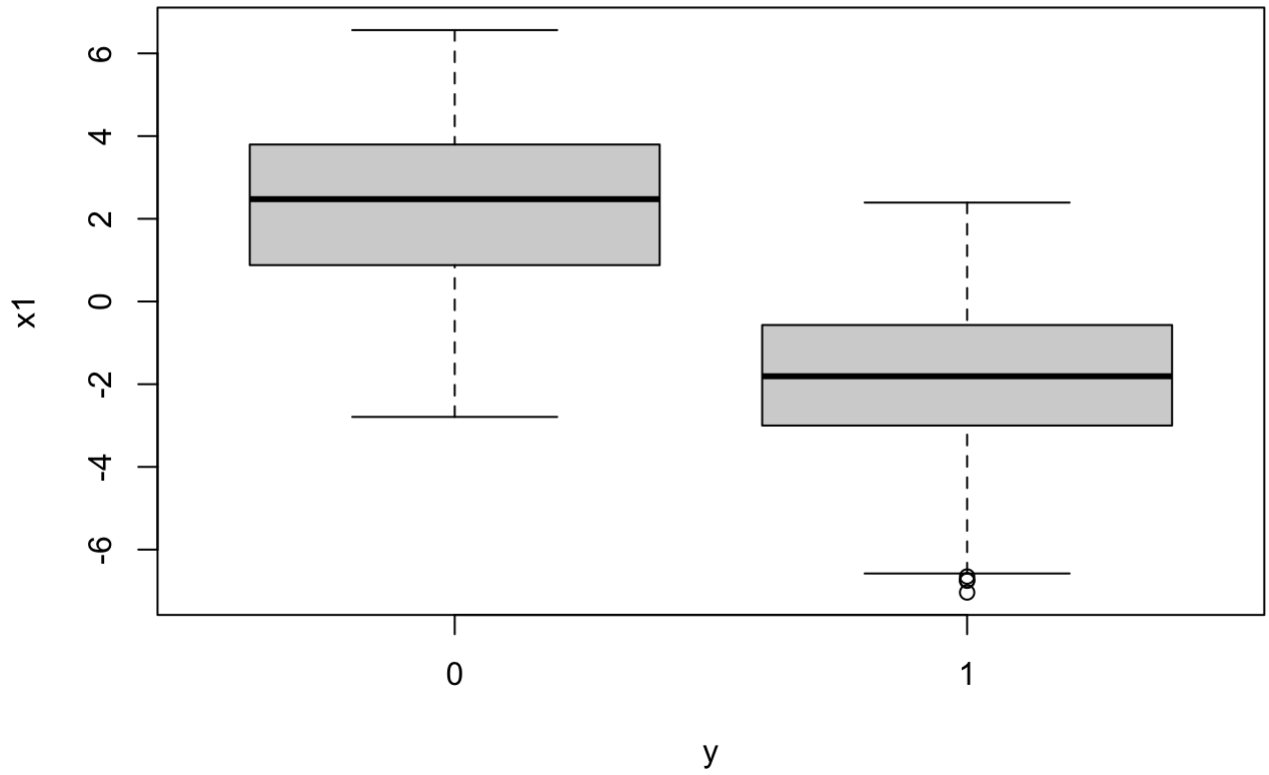
Splitting data set into training and test data sets.

```
indices = sample(1:nrow(data), size = 0.75 * nrow(data))
train_data <- data[indices, ]
test_data <- data[-indices, ]
```

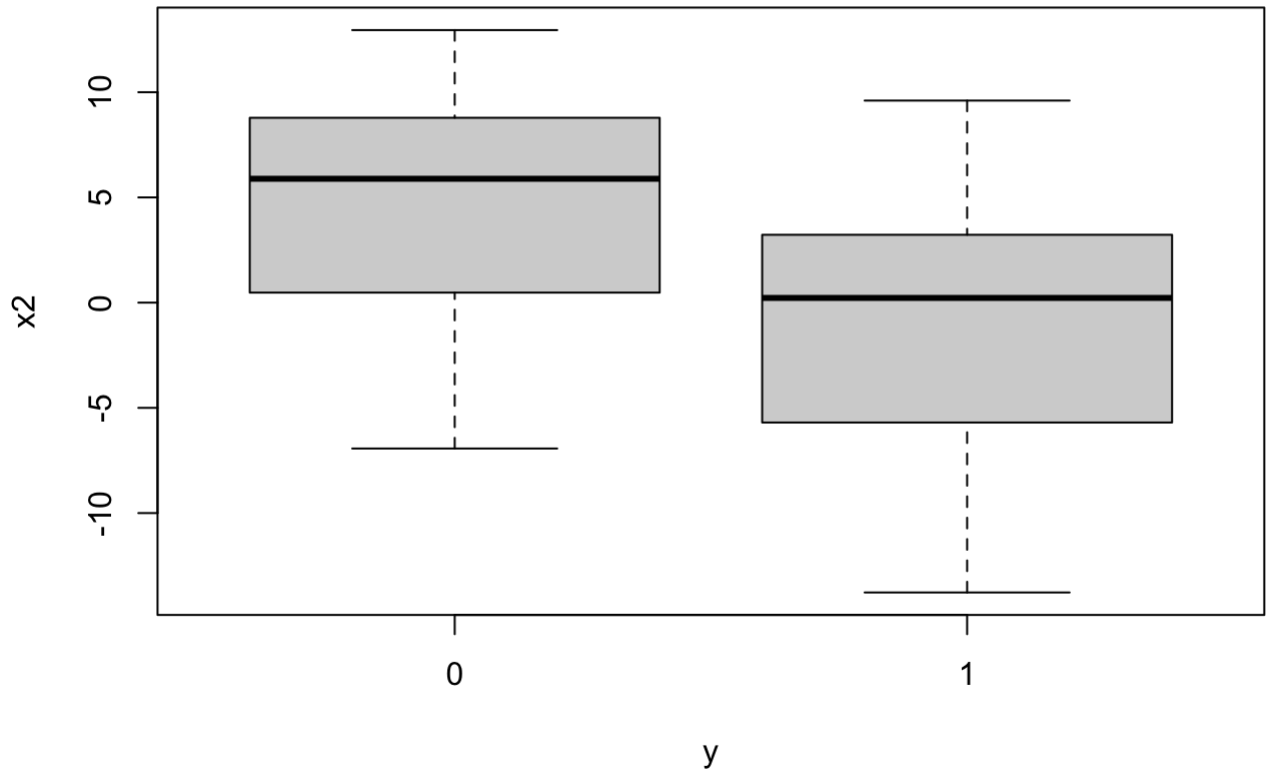
Visualizing relationships between variables

We use box plots to visualize the distribution of the values of x1, x2, x3, x4 for each classification of y -
1. We can see from the first plot that for most cases if x1 is a positive value the classification is 0 and if x1 is negative the classification is 1.
2. In plot 2 it is clear that higher value of x2 results in a 0 classification and lower value results in a 1 classification. However, there exists a significant overlap for x2 values between 0 and 2.5.
3. The other two plots are inconclusive.

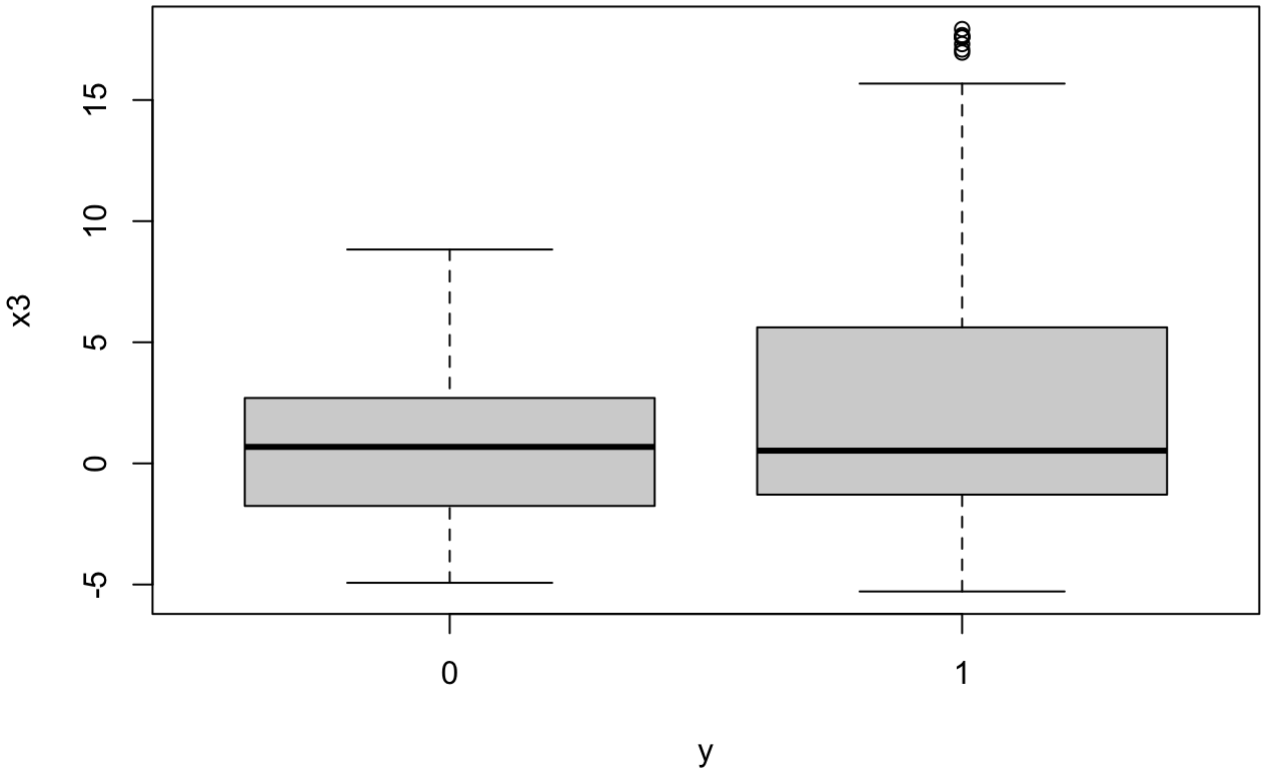
```
boxplot(x1~y, train_data)
```



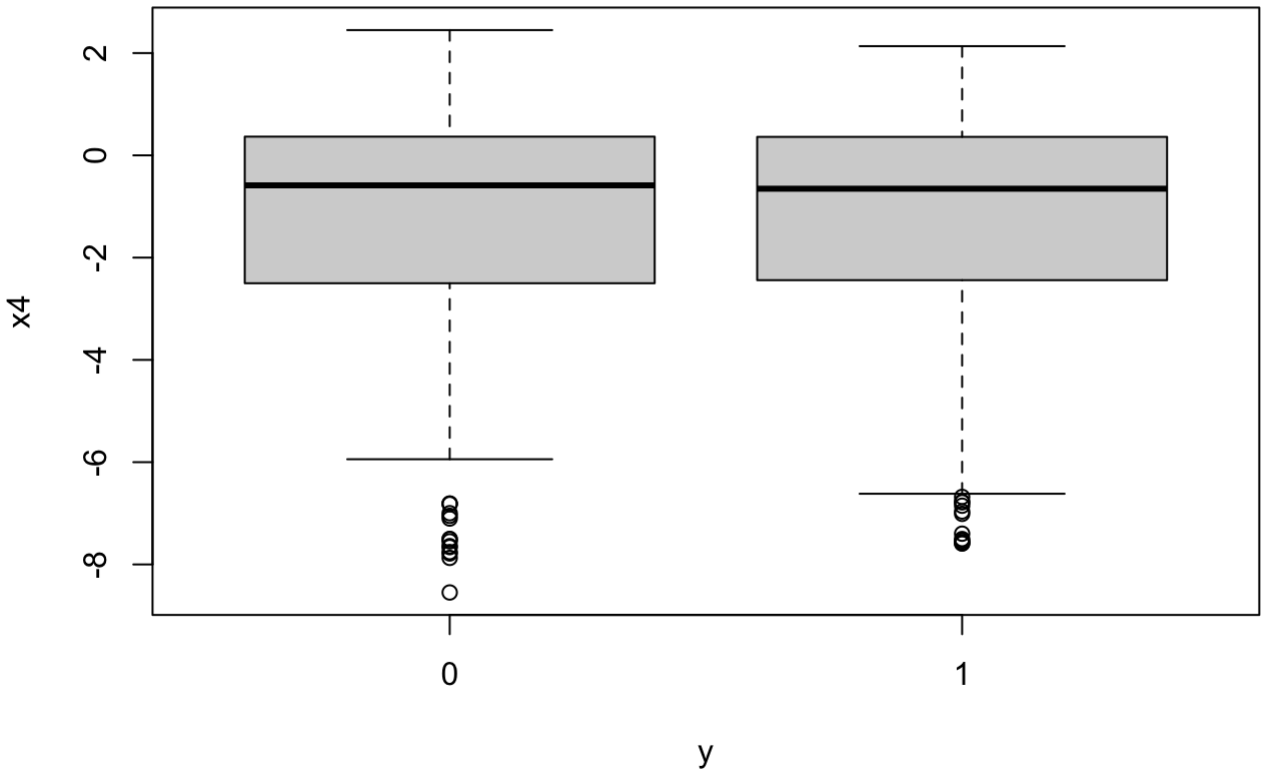

```
boxplot(x2~y, train_data)
```



```
boxplot(x3~y, train_data)
```

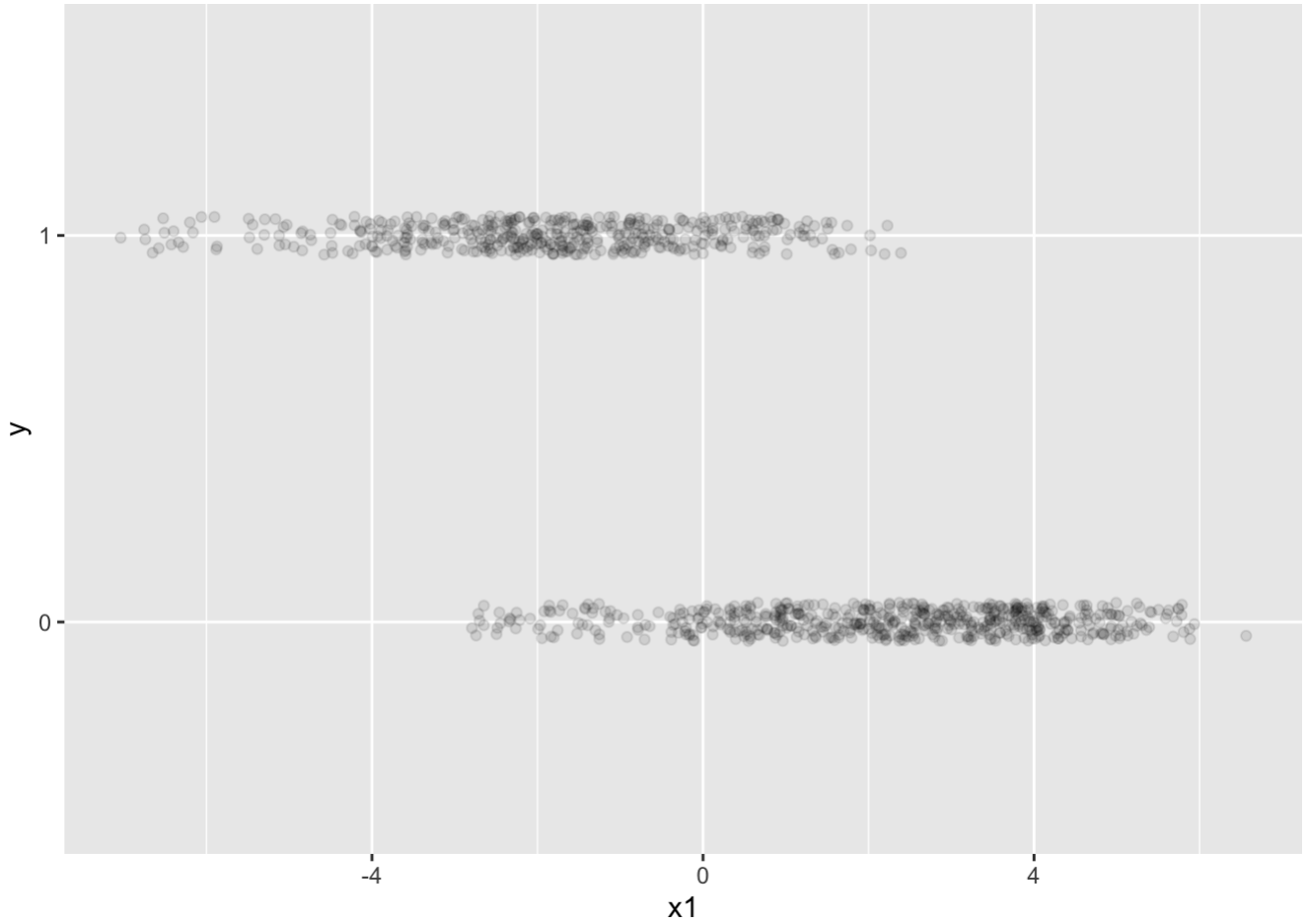



```
boxplot(x4~y, train_data)
```

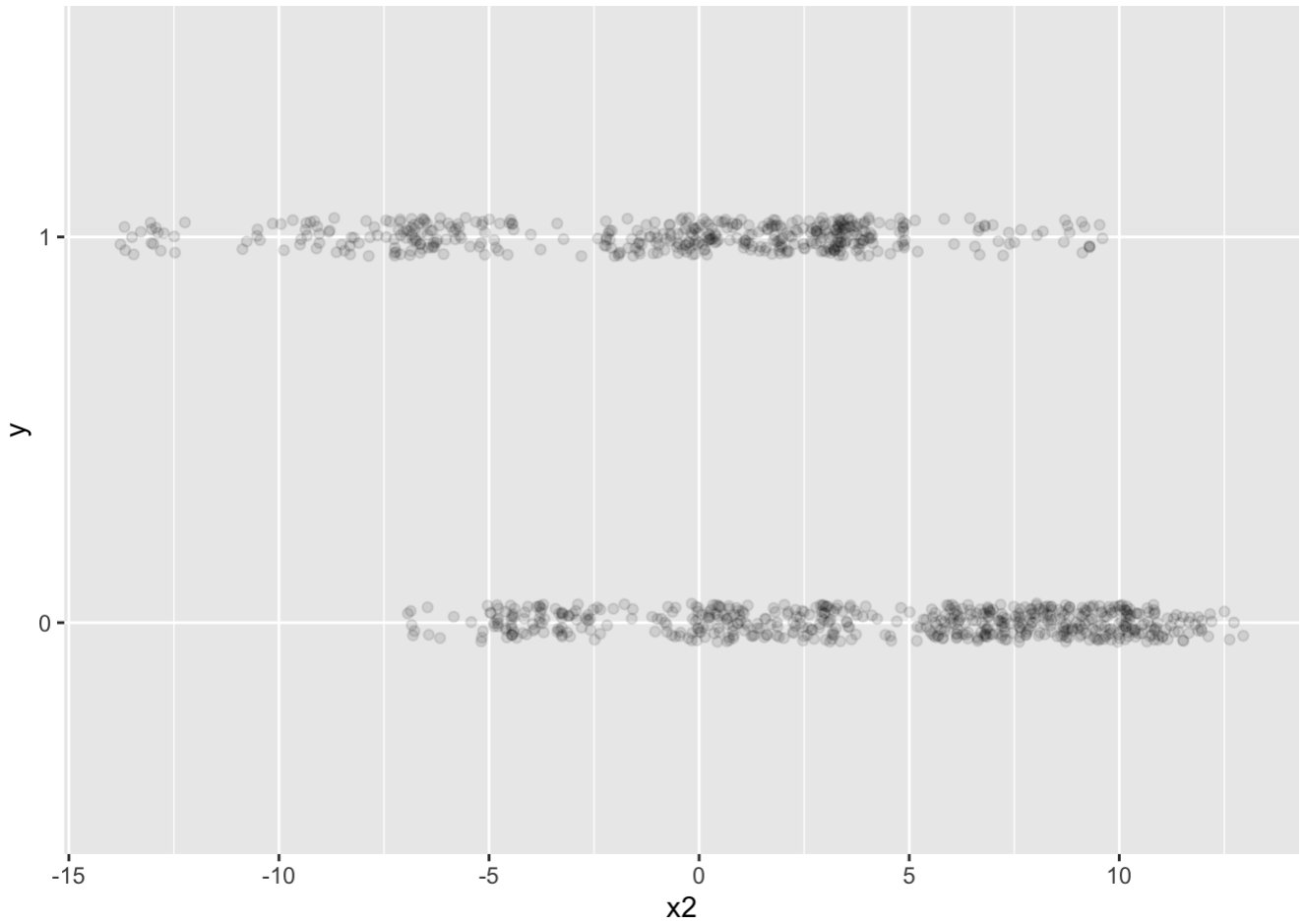


Next we use scatterplots to see if we can derive any more insights from the data.
We can see that the spread of data confirms the insights we derived from the box plots. Again, we see x4 seems to be not very useful for our classification problem.

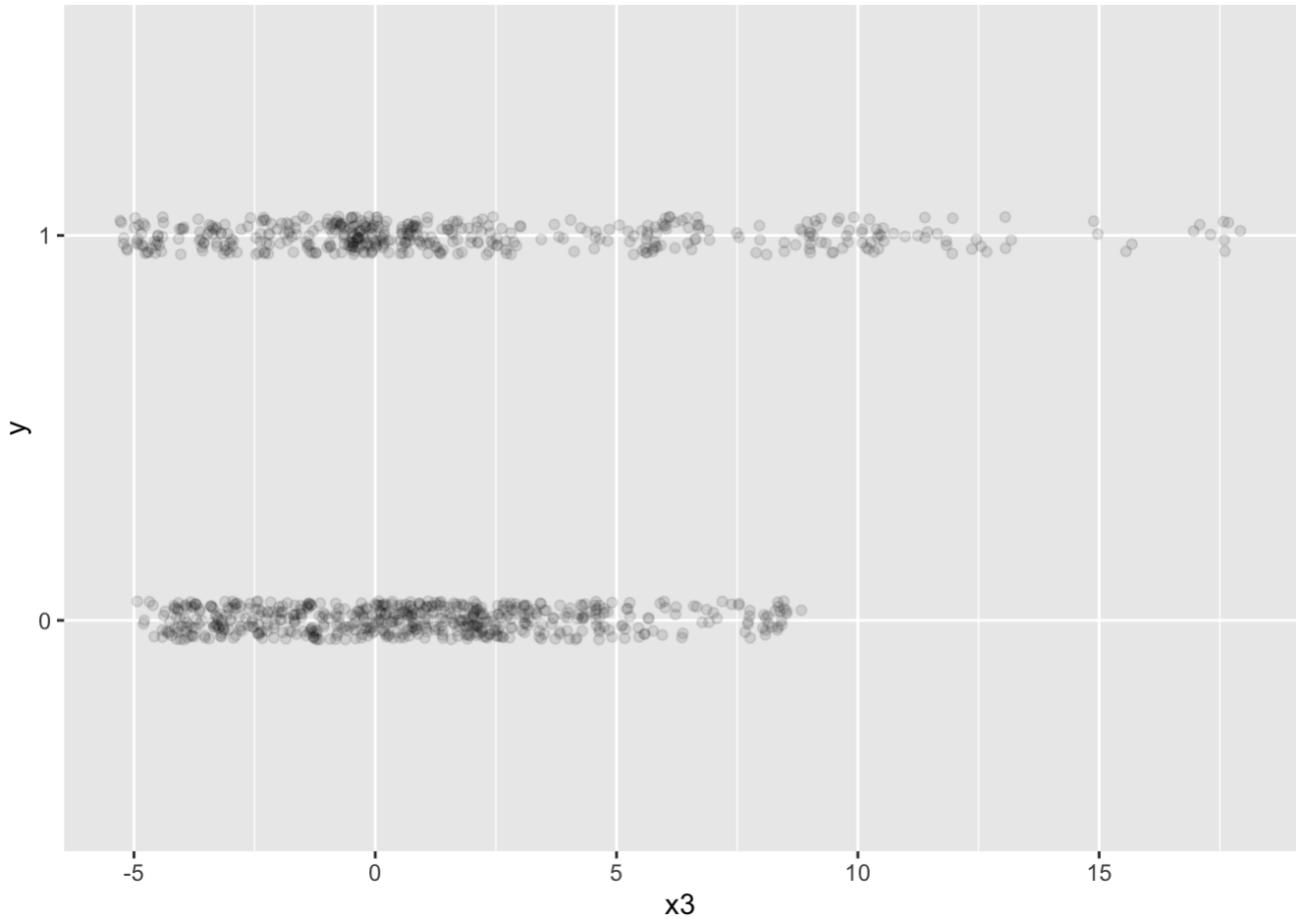
```
ggplot(train_data, aes(x=x1, y=y)) + geom_jitter(height = 0.05, alpha = 0.1)
```



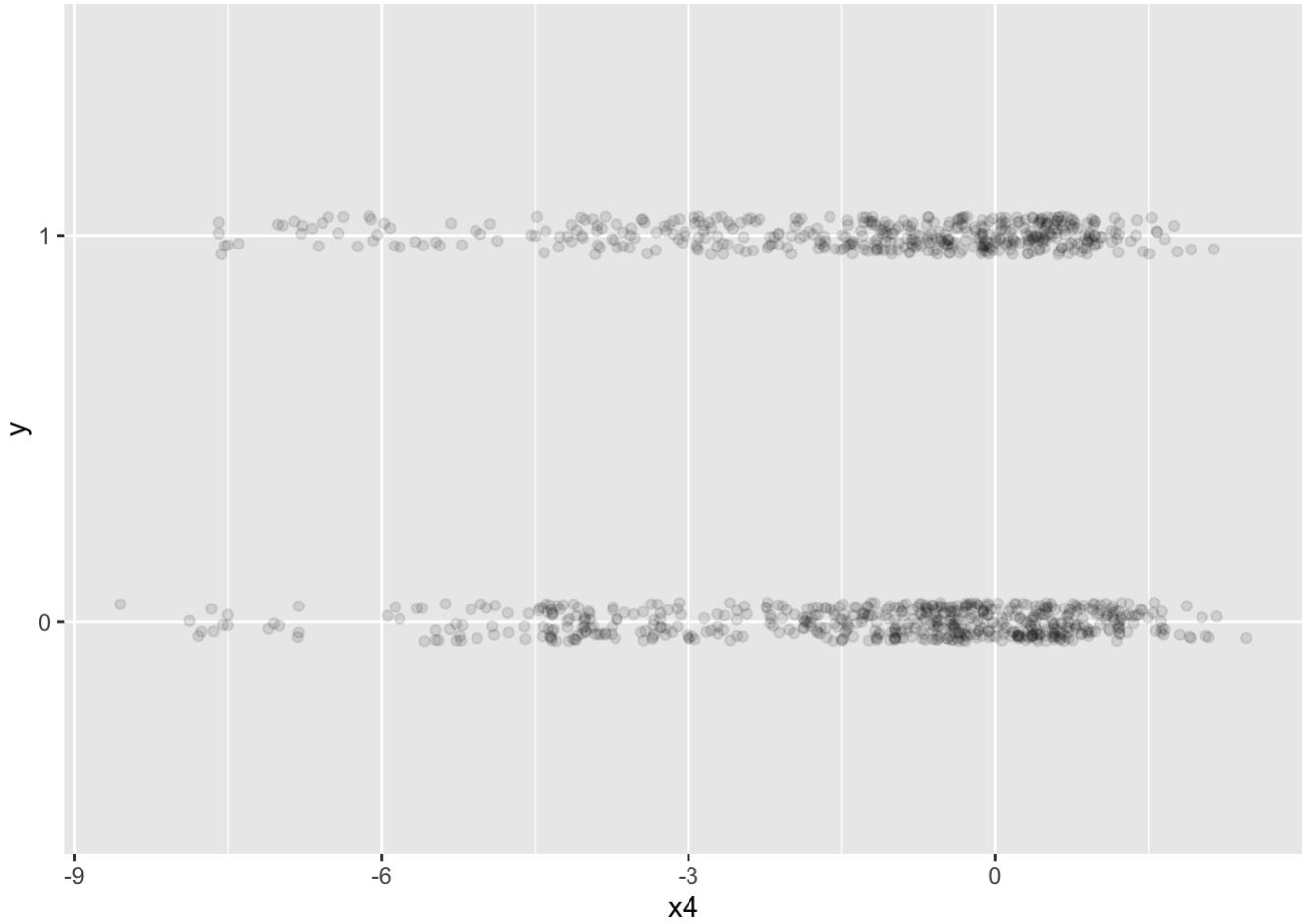

```
ggplot(train_data, aes(x=x2, y=y)) + geom_jitter(height = 0.05, alpha = 0.1)
```



```
ggplot(train_data, aes(x=x3, y=y)) + geom_jitter(height = 0.05, alpha = 0.1)
```

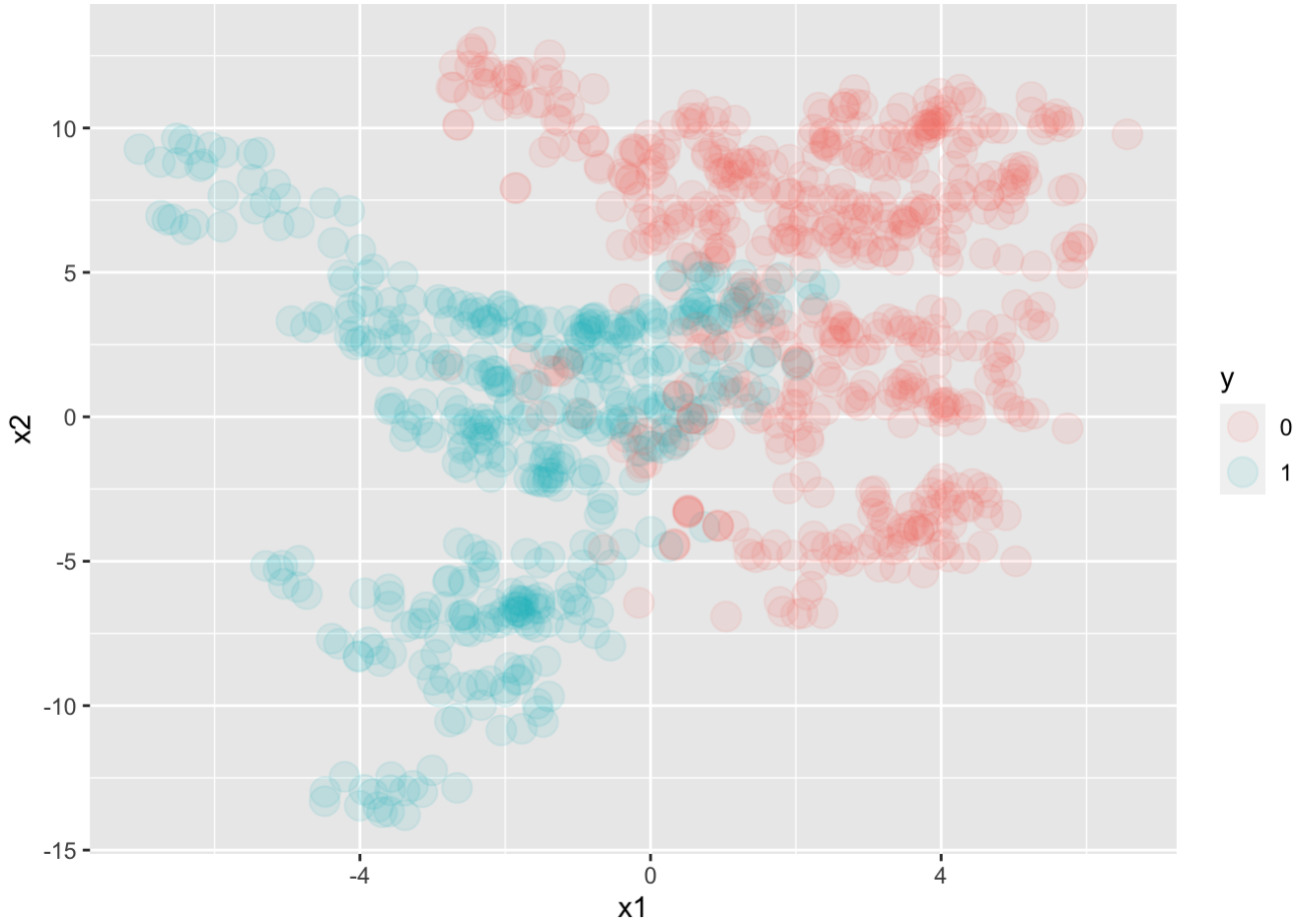



```
ggplot(train_data, aes(x=x4, y=y)) + geom_jitter(height = 0.05, alpha = 0.1)
```



Here, I tried to plot y, x1 and x2 on the same graph. I did a scatterplot of x1 vs x2 and used colors to denote the two classifications.
It is pretty clear that for a higher value of x1 and x2, the data point is likely to be classified as 0 and for lower values of x1 and x2 it is most likely to be classified as 1. But there is definitely some overlapping.

```
ggplot(train_data, aes(x=x1, y=x2, color=y)) + geom_jitter(height = 0.05, alpha = 0.
1, size=5)
```



Next, I have tried to build a Point-Biserial Correlation Matrix. These are my inferences:

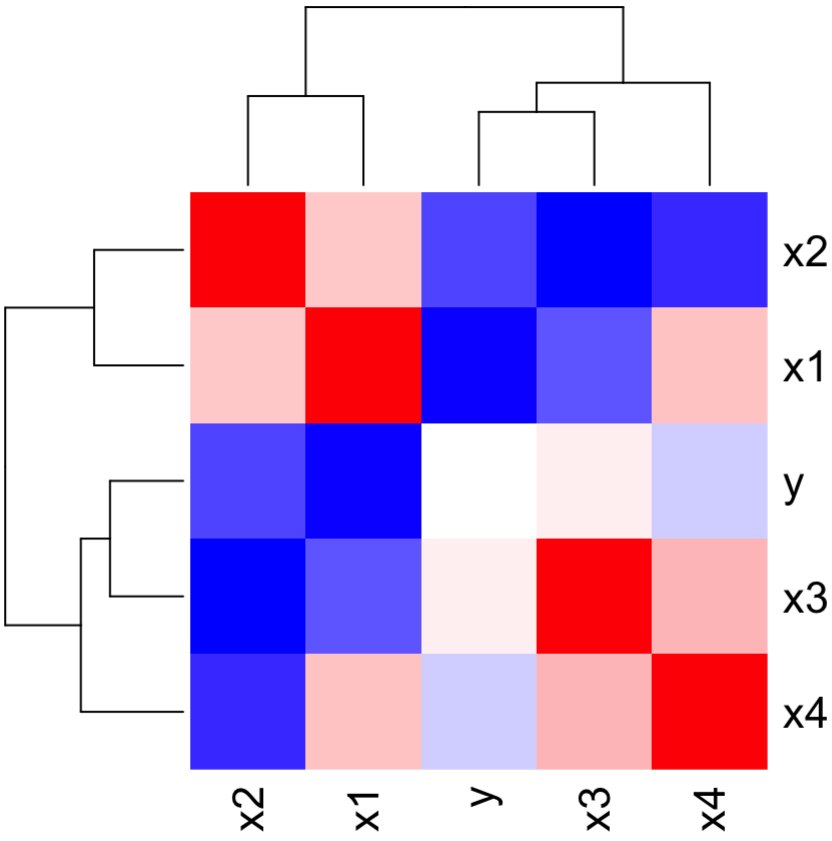
- ‘x1’ and ‘x2’ show relatively stronger relationships with ‘y’ compared to ‘x3’ and ‘x4’.
- ‘x1’ has the strongest negative correlation with ‘y’, followed by ‘x2’.
- ‘x3’ and ‘x4’ show weak correlations with ‘y’.
- ‘x1’ and ‘x2’ have a moderate positive correlation.
- ‘x1’ has a moderate negative correlation with ‘x3’ and moderate positive correlation with ‘x4’.
- ‘x2’ has a strong negative correlation with ‘x3’ and moderate negative correlation with ‘x4’.
- ‘x3’ and ‘x4’ show a weak positive correlation.

```
correlation_matrix <- matrix(NA, nrow = 5, ncol = 5)
for (i in 1:4) {
  for (j in 1:4) {
    if (i != j) {
      correlation_matrix[i, j] <- cor(data[[paste0("x", i)]], data[[paste0("x", j)]])
    } else {
      correlation_matrix[i, j] <- 1
    }
  }
}

for (i in 1:4) {
  correlation_matrix[i, 5] <- cor(data[[paste0("x", i)]], as.numeric(data$y))
  correlation_matrix[5, i] <- cor(data[[paste0("x", i)]], as.numeric(data$y))
}

colnames(correlation_matrix) <- c("x1", "x2", "x3", "x4", "y")
rownames(correlation_matrix) <- c("x1", "x2", "x3", "x4", "y")

heatmap(correlation_matrix,
        col = colorRampPalette(c("blue", "white", "red"))(100),  # Color scheme
        scale = "none",  # No scaling
        symm = TRUE,  # Show symmetric plot
        margins = c(6, 6))  # Adjust margins
```



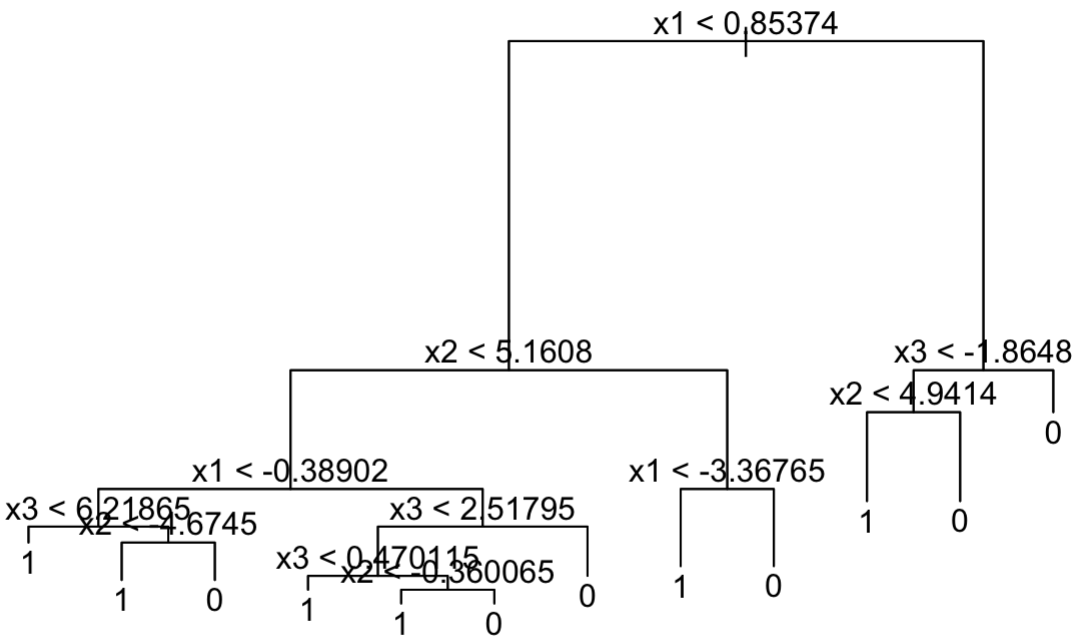
Decision-tree Classifier

First, we use Decision-tree classifiers to build a model. We train and fit a model based on the training data. Surprisingly, the tree() function automatically decided that x4 was not a useful variable and selected only the first 3 variables and fit the model.
From the summary we can see the misclassification error rate was only 0.02% which is very favorable.

```
tree.data = tree(y ~ . , train_data)
summary(tree.data)
```

```
##
## Classification tree:
## tree(formula = y ~ ., data = train_data)
## Variables actually used in tree construction:
## [1] "x1" "x2" "x3"
## Number of terminal nodes:  12
## Residual mean deviance:  0.02645 = 26.9 / 1017
## Misclassification error rate: 0.002915 = 3 / 1029
```

```
plot(tree.data)
text(tree.data, pretty=1)
```



Next we predict based on our previously trained model. We can see that the classification accuracy is 99.12%

```
tree.data.pred = predict(tree.data, test_data, type = "class")
table(tree.data.pred, test_data$y)
```

```
##
## tree.data.pred   0   1
##              0 180   1
##              1   2 160
```

```
accuracy_tree1 = sum(tree.data.pred == test_data$y) / length(test_data$y)
print(paste("Test accuracy for Decision Tree Classifier = ", accuracy_tree1*100,
"%"))
```

```
## [1] "Test accuracy for Decision Tree Classifier =  99.1253644314869 %"
```

Finally, we perform pruning to see if it leads to better accuracy. To determine the optimal tree complexity we can use cross-validation. Here, we use the misclassification error rate to guide the pruning process.

```
cv.tree = cv.tree(tree.data, FUN = prune.misclass)
cv.tree
```

```
## $size
## [1] 12 10  8  6  4  3  2  1
##
## $dev
## [1]  15  20  35  64  76 119 178 449
##
## $k
## [1]  -Inf   3.5   5.0  13.5  15.0  30.0  64.0 278.0
##
## $method
## [1] "misclass"
##
## attr(,"class")
## [1] "prune"         "tree.sequence"
```

We can see that a tree with 12 terminal nodes has the least cross-validation errors. This is same as the unpruned tree. After prediction, we see that we obtain the same accuracy rate as before.

```
prune.data = prune.misclass(tree.data, best = 12)
prune.data.pred = predict(prune.data, test_data, type = "class")
table(prune.data.pred, test_data$y)
```

```
##
## prune.data.pred   0   1
##               0 180   1
##               1   2 160
```

```
accuracy_tree2 = sum(prune.data.pred == test_data$y) / length(test_data$y)
print(paste("Test accuracy for Decision Tree Classifier after pruning = ", accuracy_t
ree2*100, "%"))
```

```
## [1] "Test accuracy for Decision Tree Classifier after pruning =  99.1253644314869
%"
```

Support Vector Classifier

Next we try to use a Support Vector Classifier to work on our data. Here, we are using a linear kernel and setting cost as 0.01.
We get an accuracy of 97.96%.

```
svm.data = svm(y ~ ., data = train_data, kernel = "linear", scale = FALSE, cost=0.01)
summary(svm.data)
```

```
##
## Call:
## svm(formula = y ~ ., data = train_data, kernel = "linear", cost = 0.01,
##     scale = FALSE)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  linear
##        cost:  0.01
##
## Number of Support Vectors:  123
##
##  ( 62 61 )
##
##
## Number of Classes:  2
##
## Levels:
##  0 1
```

```
svm.data.pred = predict(svm.data, test_data)
accuracy_svm1 <- sum(svm.data.pred == test_data$y) / nrow(test_data)
print(paste("Test accuracy for Support Vector Classifier with cost as 0.01 = ", accur
acy_svm1*100, "%"))
```

```
## [1] "Test accuracy for Support Vector Classifier with cost as 0.01 =  97.959183673
4694 %"
```

We are trying to use cross-validation to pick the best cost. We use the tune function to automatically pick the best cost.
The function determined that 10 would be the optimal cost. Using 10 as cost, we predict the classification. We obtain a test accuracy of 98.84% which is higher than the previous attempt.

```
tune.out = tune(svm, y ~ ., data = train_data, kernel = "linear", ranges = list(cost
= c(0.001 , 0.01, 0.1, 1, 5, 10, 100)))
svm.bestmod = tune.out$best.model
summary(svm.bestmod)
```

```
##
## Call:
## best.tune(METHOD = svm, train.x = y ~ ., data = train_data, ranges = list(cost = c
(0.001,
##     0.01, 0.1, 1, 5, 10, 100)), kernel = "linear")
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  linear
##        cost:  5
##
## Number of Support Vectors:  42
##
##  ( 21 21 )
##
##
## Number of Classes:  2
##
## Levels:
##  0 1
```

```
svm.bestmod.pred = predict(svm.bestmod, test_data)
accuracy_svm2 <- sum(svm.data.pred == test_data$y) / nrow(test_data)
print(paste("Test accuracy for Support Vector Classifier with cost as 10 = ", accurac
y_svm2*100, "%"))
```

```
## [1] "Test accuracy for Support Vector Classifier with cost as 10 =  97.95918367346
94 %"
```

Naive Bayes Classifier

We attempt classification using a Naive Bayes Classifier. We use 10-fold cross-validation to get a more reliable estimate of the model's performance by evaluating it on different subsets of the data.
We obtain an accuracy of 83.38%.

```
nb.data = naiveBayes(y ~ ., data = train_data, cross = 10)
nb.data.pred = predict(nb.data, newdata=test_data)
accuracy_nb1 <- sum(nb.data.pred == test_data$y) / nrow(test_data)
print(paste("Test accuracy for Naive Bayes Classifier = ", accuracy_nb1*100, "%"))
```

```
## [1] "Test accuracy for Naive Bayes Classifier =  83.3819241982507 %"
```

Logistic Regression

Here, we attempt to classify using logistic regression by passing family = Binomial. Since logistic regression predicts probabilities, we convert it to binary predictions using ifelse().
We obtain a classification accuracy of 99.42%.

```
log.data = glm(y ~ ., data = train_data, family = binomial)
```

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

```
log.data.pred = predict(log.data, newdata= test_data, type = "response")
log.pred.binary = ifelse(log.data.pred > 0.5, 1, 0)
accuracy_log <- sum(log.pred.binary == test_data$y) / nrow(test_data)
print(paste("Test accuracy Naive Bayes Classifier = ", accuracy_log*100, "%"))
```

```
## [1] "Test accuracy Naive Bayes Classifier =  99.4169096209913 %"
```

We see a cross-validation error of 0.8% which is pretty good!

```
cv.log = cv.glm(train_data, log.data, K = 5)
```

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

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

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

```
print(paste("Cross-validated Error:", cv.log$delta[1], "\n"))
```

```
## [1] "Cross-validated Error: 0.00903771825902642 \n"
```

Random Forest

Finally, we try a Random Forest approach with a 5-fold cross validation. Since randomForest() does not support cross-validation out of the box, I used a control parameter from the caret package.

```
ctrl_param = trainControl(method = "cv", number = 5)
rf.data = randomForest(y ~ . , data = train_data, method = "rf", trControl = ctrl_par
am)
print(rf.data)
```

```
##
## Call:
##  randomForest(formula = y ~ ., data = train_data, method = "rf",      trControl =
ctrl_param)
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 2
##
##         OOB estimate of  error rate: 0.68%
## Confusion matrix:
##     0   1 class.error
## 0 575   5 0.008620690
## 1   2 447 0.004454343
```

Now on predicting the classificatiosn of the test data, we obtain an accuracy of 99.71%.

```
rf.data.pred = predict(rf.data, newdata = test_data)
accuracy_rf <- sum(rf.data.pred == test_data$y) / nrow(test_data)
print(paste("Test accuracy for Random Forest with 5-fold cross validation = ", accura
cy_rf*100, "%"))
```

```
## [1] "Test accuracy for Random Forest with 5-fold cross validation =  99.7084548104
956 %"
```

Results

We were given a comma-separated text file. This file consisted of 4 variables and 1 target variable. The target variable was either 0 or 1. On exploring the data, we realized that x1 and x2 were the most influential variables for classification.

We performed multiple methods of classification and we found that Random Forests had the highest classification accuracy, followed by Logistic regression. Next up was Decision Tree Classifier. Here we saw that pruning did not make any difference as the tree was optimally pruned already. After this came, Support Vector Classifiers. Although, using cross-validation to select the cost increased accuracy, it was lower than the previous methods. Finally, we have Naive Bayes Classifier which had the least accuracy by far.