Homework-6

Ivan Ojiambo

November 1, 2017

Exe1

```r
options(warn=-1)
library(caret)
```

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

```
## Loading required package: ggplot2
```

```r
library(htmlTable)
library(ggplot2)

data <- read.csv("ex1.csv")

TP_A <- sum(data$A == 1 & data$Actual == 1)   # True positive for A
FP_A <- sum(data$A == 1 & data$Actual == 0)   # False positives
TN_A <- sum(data$A == 0 & data$Actual == 0)   # True Negative
FN_A <- sum(data$A == 0 & data$Actual == 1)   # False Negative

(PPOs_A <- TP_A + FP_A )  #predicated positives  = True Positive  + False Positive
```

```
## [1] 133
```

```r
(PN_A <- TN_A + FN_A )     #predicated Negatives = True Negative + False Negative
```

```
## [1] 67
```

```r
(POs_A <- TP_A + FN_A)      #positives = true positives + false Negatives
```

```
## [1] 100
```

```r
(Neg_A <- TN_A + FP_A)     #Negatives = true Negatives + false positive
```

```
## [1] 100
```

```r
(Tot_A <- POs_A + Neg_A)    #Total = Total of all positives + total of all Negatives
```

```
## [1] 200
```

```r
library(knitr)
#drawing Table for classifier A
tableA <- data.frame("ACTUAL| PREDICTED" = c(paste("DISEASE 1"),paste("NO DISEASE 0"),  paste("TOTAL"))
                     "DISEASE (1)"      = c(paste(TP_A, "(TP)"), paste(FP_A, "(FP)"), PPOs_A ),
                     "NO DISEASE (0)"   = c(paste(FN_A, "(FN)"), paste(TN_A, "(TN)"), PN_A ),
                     "TOTAL"            = c(paste(POs_A), paste(Neg_A), paste(Tot_A) )
                     )
kable(tableA, caption = "Accuracy on classifier A")
```

Table 1: Accuracy on classifier A

ACTUAL..PREDICTED	DISEASE..1.	NO.DISEASE..0.	TOTAL
DISEASE 1	68 (TP)	32 (FN)	100
NO DISEASE 0	65 (FP)	35 (TN)	100
TOTAL	133	67	200

```r
Accuracy_A <- (TP_A + TN_A)/(TP_A + TP_A + FN_A + FP_A)  #accuracy on classifier A
print(paste("Accuracy on classifier A =", Accuracy_A))
```

```
## [1] "Accuracy on classifier A = 0.44206008583691"
```

- From classifier A, only 44.2% of the patients were correctly diagnosed.

```r
precision_A <-TP_A/(TP_A + FP_A )                #precision for classifier A
print(paste("Precision for classifier A is ", precision_A))
```

```
## [1] "Precision for classifier A is  0.511278195488722"
```

- From the precision value , we learn that more than 48% of the patients were incorrectly diagnoised as sick yet they were not

```r
recall_A <- TP_A/( TP_A + FN_A)
print(paste("Recall for classifier A is ", recall_A))
```

```
## [1] "Recall for classifier A is  0.68"
```

- From the recall value, we learn that of the patients who were sick, only 68% were correctly diagnoised .

```r
F_measure_A <- 2/((1/precision_A)+(1/recall_A))
print(paste("F-measure for classifier A is ", F_measure_A ))
```

```
## [1] "F-measure for classifier A is  0.583690987124463"
```

```r
TP_B <- sum(data$B == 1 & data$Actual == 1)
FP_B <- sum(data$B == 0 & data$Actual == 1)
TN_B <- sum(data$B == 1 & data$Actual == 0)
FN_B <- sum(data$B == 0 & data$Actual == 0)
#confusion Matrix for classifier A
#confusionMatrix(data$A, data$Actual)

PPOs_B <- TP_B + FP_B     #predicated positives = True Positive + False Positives

PN_B <- TN_B + FN_B       #predicated Negatives = True Negative + False Negative

POs_B <- TP_B + FN_B      #positives = true positives + false Negatives

Neg_B <- TN_B + FP_B      #Negatives = true Negatives + false positive

Tot_B <- POs_B + Neg_B    #Total = Total of all positives + total of all Negatives

#drawing Table for classifier B
tableB <- data.frame("ACTUAL| PREDICTED " = c(paste("DISEASE 1"),paste("NO DISEASE 0"), paste("TOTAL"))
                     "DISEASE1 "      = c(paste(TP_B, "(TP)"), paste(FP_B, "(FP)"), PPOs_B ),
                     "NO DISEASE0 "   = c(paste(FN_B, "(FN)"), paste(TN_B, "(TN)"), PN_B ),
                     "TOTAL "         = c(paste(POs_B), paste(Neg_B), paste(Tot_B) )
```

```
                    )
kable(tableB, caption = "Accuracy on classifier B")
```

Table 2: Accuracy on classifier B

ACTUAL..PREDICTED.	DISEASE1.	NO.DISEASE0.	TOTAL.
DISEASE 1	53 (TP)	78 (FN)	131
NO DISEASE 0	47 (FP)	22 (TN)	69
TOTAL	100	100	200

```
(Accuracy_B <- (TP_B + TN_A)/(TP_B + TP_B + FN_B + FP_B))  #accuracy on classifier B
```

```
## [1] 0.3809524
```

```
print(paste("Accuracy on classifier B =", Accuracy_B))
```

```
## [1] "Accuracy on classifier B = 0.380952380952381"
```

- From the accuracy we learn that only 38% of the patients were diagnoised correctly

```
precision_B <-TP_B/(TP_B + FP_B )  #precision for classifier B = True positive/(true pos + False Negati
print(paste("precision for classifier B =", precision_B))
```

```
## [1] "precision for classifier B = 0.53"
```

- From the precision value, we learn that of the patients who were sick, only 47% of the patients incorrectly diagnoised as sick

```
recall_B <- TP_B/(TP_B + FN_B)
print(paste("Recall for classifier B is ", recall_B))
```

```
## [1] "Recall for classifier B is  0.404580152671756"
```

- From the recale/sensitivity value , we learn that of the patients who were sick, only 40% were diagnoised correctly

```
F_measure_B <- 2/((1/precision_B)+(1/recall_B))
print(paste("F-measure for classifier B is ", F_measure_B ))
```

```
## [1] "F-measure for classifier B is  0.458874458874459"
```

- From the accuracy value, I can say that classifier A is better than classifier A since A has a high accuracy.

Exe2

```
train_data <- read.csv("training.csv")
test1_data <- read.csv("testing_1.csv")
test2_data <- read.csv("testing_2.csv")


ctrl <- trainControl(method="none", number = 1, repeats = 1)
rf_fit <- train(as.factor(y)~., data = train_data, method = 'rf', trControl = ctrl)

#predicated values
```

```r
predicated_values <- predict(rf_fit, test1_data)
confusionMatrix(predicated_values, test1_data$y, positive = "1")
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 235   8
##          1   2   5
##
##                Accuracy : 0.96
##                  95% CI : (0.9277, 0.9807)
##     No Information Rate : 0.948
##     P-Value [Acc > NIR] : 0.2448
##
##                   Kappa : 0.4811
##  Mcnemar's Test P-Value : 0.1138
##
##             Sensitivity : 0.3846
##             Specificity : 0.9916
##          Pos Pred Value : 0.7143
##          Neg Pred Value : 0.9671
##              Prevalence : 0.0520
##          Detection Rate : 0.0200
##    Detection Prevalence : 0.0280
##       Balanced Accuracy : 0.6881
##
##        'Positive' Class : 1
##
```

```r
#predicated values for test data 2
predicated_values <- predict(rf_fit, test2_data)
confusionMatrix(predicated_values, test2_data$y, positive = "1")
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 125 105
##          1   0  20
##
##                Accuracy : 0.58
##                  95% CI : (0.5162, 0.6419)
##     No Information Rate : 0.5
##     P-Value [Acc > NIR] : 0.006741
##
##                   Kappa : 0.16
##  Mcnemar's Test P-Value : < 2.2e-16
##
##             Sensitivity : 0.1600
##             Specificity : 1.0000
##          Pos Pred Value : 1.0000
##          Neg Pred Value : 0.5435
##              Prevalence : 0.5000
##          Detection Rate : 0.0800
##    Detection Prevalence : 0.0800
##       Balanced Accuracy : 0.5800
##
##        'Positive' Class : 1
##
```

- while testing the train data using test1 data, it gives an accuracy of 84.8% and a sensitivity of 0% ie no positive values were correctly predicated.
- while testing the train data using test2 data, it gives an accuracy of 60% and a sensitivity of 20% ie only 25 true positive values were predicated correctly and 100 of them were incorrectly predicated.
- Since this data is non symmetric, based on the low levels of sensitivity, this model is not sufficeinet model to come up with a correct model

```r
#balanced data
library(ROSE,  warn.conflicts = FALSE)
```

```
## Loaded ROSE 0.0-3
```

```r
#over sampling
bal_train_data_over <- ovun.sample(as.factor(y)~., data=train_data, N=1900,
                        seed=1, method="over")$data

bal_train_data_under <- ovun.sample(as.factor(y)~., data=train_data, N=100,
                        seed=1, method="under")$data

bal_train_data <- ovun.sample(as.factor(y)~., data=train_data, N=100, p = 0.5,
                        seed=1, method="both")$data
```

- For over sampling, we increase the size of the sample by multipling by two the sample size of the maximum of positive or negative label.
- For under sampling, we increase the sample size of the sample by multiplying by two the sample size of the minimum of positive or negative label.

```r
#confusion matrix on test1 data when the data is over sampled
trained_data_over      <- train(as.factor(y)~., data = bal_train_data_over, method = 'rf', trControl = 

predicated_values_test1_over <- predict(trained_data_over, test1_data)
confusionMatrix(predicated_values_test1_over, test1_data$y, positive = "1")
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 236  12
##          1   1   1
##
##                Accuracy : 0.948
##                  95% CI : (0.9127, 0.972)
##     No Information Rate : 0.948
##     P-Value [Acc > NIR] : 0.573071
##
##                   Kappa : 0.1211
##  Mcnemar's Test P-Value : 0.005546
##
```

```
##             Sensitivity : 0.07692
##             Specificity : 0.99578
##          Pos Pred Value : 0.50000
##          Neg Pred Value : 0.95161
##              Prevalence : 0.05200
##          Detection Rate : 0.00400
##    Detection Prevalence : 0.00800
##       Balanced Accuracy : 0.53635
##
##        'Positive' Class : 1
##
```

```
#confusion matrix on test2 data when data is over sampled
predicated_values_test2_over <- predict(trained_data_over, test2_data)
confusionMatrix(predicated_values_test2_over, test2_data$y, positive = "1")
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 125 115
##          1   0  10
##
##                Accuracy : 0.54
##                  95% CI : (0.4761, 0.603)
##     No Information Rate : 0.5
##     P-Value [Acc > NIR] : 0.1147
##
##                   Kappa : 0.08
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 0.0800
##             Specificity : 1.0000
##          Pos Pred Value : 1.0000
##          Neg Pred Value : 0.5208
##              Prevalence : 0.5000
##          Detection Rate : 0.0400
##    Detection Prevalence : 0.0400
##       Balanced Accuracy : 0.5400
##
##        'Positive' Class : 1
##
```

- After oversampling the train data and predicating on test1 data , we see that the accuracy has increased from 84.4% to 94.8% and sensitivity has increased from 0 to 7.6%.
- After oversampling the train data and predicating on test2 data, we see that the accuracy has decreased from 60% to 53% and sensitivity has also decreased from 20% to 7.2%.

Was there any improvement in performance

- There is a slight improvement in performance in while predicating test1 data ie there was increase in sentivity from 0% to 7.6% but thre was no improvement while predicating test2 data.

```r
#training the undersampled train data using random classifier
trained_data_under      <- train(as.factor(y)~., data = bal_train_data_under, method = 'rf', trControl =

#confusion matrix on test1 data when the data is under sampled
predicated_test1_under  <- predict(trained_data_under, test1_data)
confusionMatrix(predicated_test1_under, test1_data$y, positive = "1")
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 148   2
##          1  89  11
##
##                Accuracy : 0.636
##                  95% CI : (0.573, 0.6957)
##     No Information Rate : 0.948
##     P-Value [Acc > NIR] : 1
##
##                   Kappa : 0.1131
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 0.8462
##             Specificity : 0.6245
##          Pos Pred Value : 0.1100
##          Neg Pred Value : 0.9867
##              Prevalence : 0.0520
##          Detection Rate : 0.0440
##    Detection Prevalence : 0.4000
##       Balanced Accuracy : 0.7353
##
##        'Positive' Class : 1
##
```

```r
#confusion matrix on test2 data when the data is under sampled
predicated_test2_under  <- predict(trained_data_under, test2_data)
confusionMatrix(predicated_test2_under, test2_data$y, positive = "1")
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0  73  11
##          1  52 114
##
##                Accuracy : 0.748
##                  95% CI : (0.6894, 0.8006)
##     No Information Rate : 0.5
##     P-Value [Acc > NIR] : 9.405e-16
##
##                   Kappa : 0.496
##  Mcnemar's Test P-Value : 4.667e-07
##
##             Sensitivity : 0.9120
##             Specificity : 0.5840
##          Pos Pred Value : 0.6867
##          Neg Pred Value : 0.8690
##              Prevalence : 0.5000
##          Detection Rate : 0.4560
##    Detection Prevalence : 0.6640
##       Balanced Accuracy : 0.7480
##
##        'Positive' Class : 1
##
```

- After under sampling the train data and predicating on test1 data , we see that the accuracy is 63.6% and sensitivity is 84.6%.
- For test2 data, we see the accuracy is 74.8% and sensitivity is 91.2
- There is a great improvement in performance in both test data ie we see the sensitivity increasing in test1 and test2 data to 84.6% and 91.2% respectively.

```
#training the balanced  data using random classifier
trained_data      <- train(as.factor(y)~., data = bal_train_data, method = 'rf', trControl = ctrl)

#confusion matrix on test1 data when the data is combined
predicated_test1  <- predict(trained_data, test1_data)
confusionMatrix(predicated_test1, test1_data$y, positive = "1")
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 191   4
##          1  46   9
##
##                Accuracy : 0.8
##                  95% CI : (0.745, 0.8478)
##     No Information Rate : 0.948
##     P-Value [Acc > NIR] : 1
##
##                   Kappa : 0.1972
##  Mcnemar's Test P-Value : 6.7e-09
##
##             Sensitivity : 0.6923
##             Specificity : 0.8059
##          Pos Pred Value : 0.1636
##          Neg Pred Value : 0.9795
##              Prevalence : 0.0520
##          Detection Rate : 0.0360
##    Detection Prevalence : 0.2200
##       Balanced Accuracy : 0.7491
##
##        'Positive' Class : 1
##
```

```
#confusion matrix on test2 data when the data is combined
predicated_test2  <- predict(trained_data, test2_data)
confusionMatrix(predicated_test2, test2_data$y, positive =
                  "1")
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 102  52
##          1  23  73
##
##                Accuracy : 0.7
##                  95% CI : (0.6391, 0.7561)
##     No Information Rate : 0.5
##     P-Value [Acc > NIR] : 1.105e-10
##
##                   Kappa : 0.4
##  Mcnemar's Test P-Value : 0.001224
##
##             Sensitivity : 0.5840
##             Specificity : 0.8160
##          Pos Pred Value : 0.7604
##          Neg Pred Value : 0.6623
##              Prevalence : 0.5000
##          Detection Rate : 0.2920
##    Detection Prevalence : 0.3840
##       Balanced Accuracy : 0.7000
##
##        'Positive' Class : 1
##
```

- When the train data is combined , for test1 data, we see that the accuracy is only 8% and sensitivity is 69.2%
- For test2 data, we see that the accuracy is 7% and sensitivity is 58.4%.
- There is also an improvement in performance since in both test cases the sensitivity is increasing.

Which sampling method worked best?

- Since the data is uneven and from the confusion matrix of the three sampling methods, ***down sampling*** was the best since it has a high a sentivity ie 84% and 91.2% in test1 and test2 data respectively compared to other sampling methods

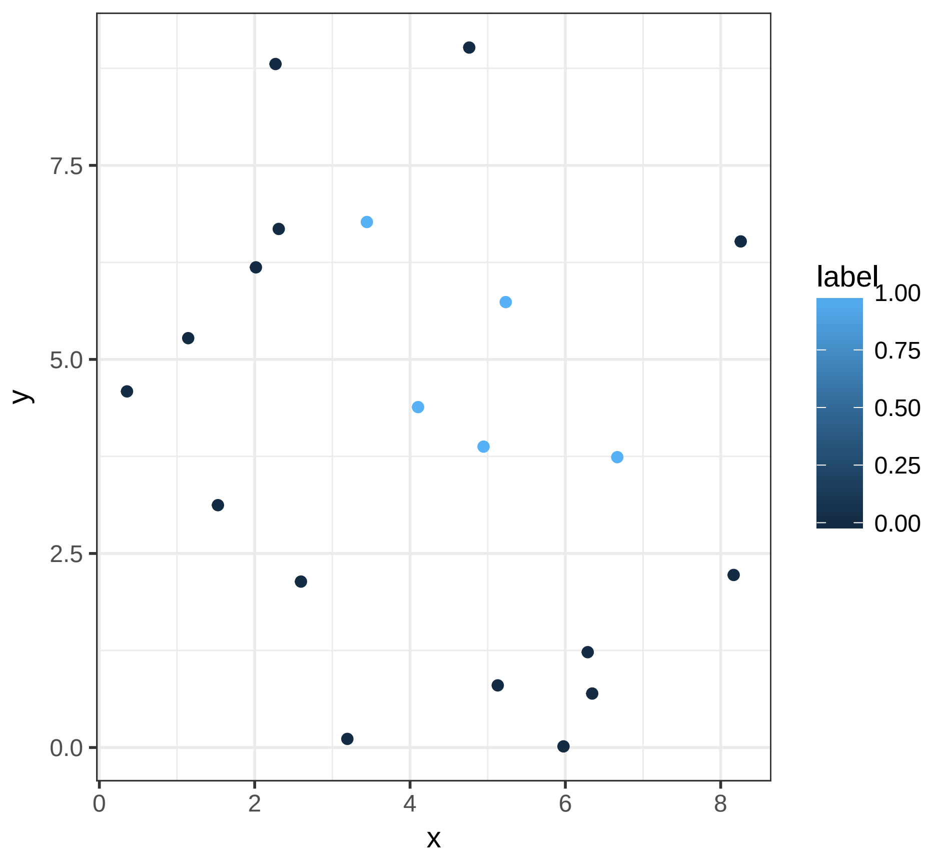
Exe3

a

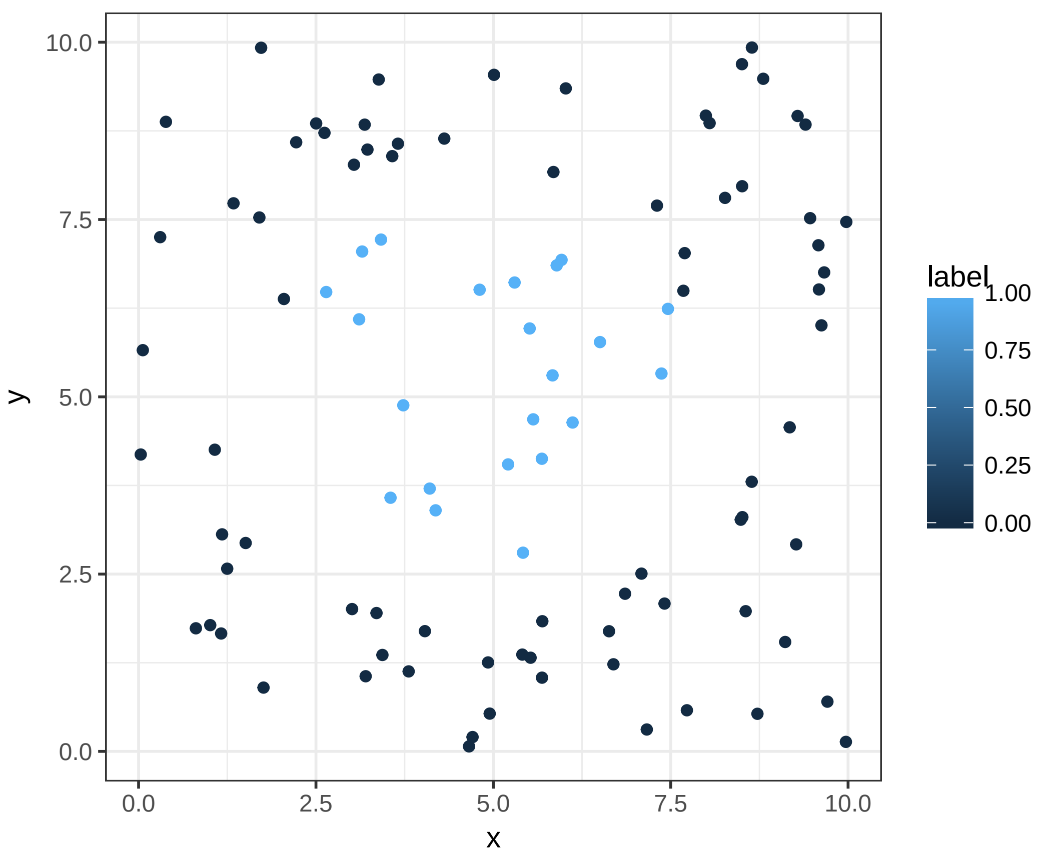
```
generateLabels <- function(df){
  for(i in 1:nrow(df)){
    x = df$x[i]
    y = df$y[i]
    if(((x - 5)^2 + (y - 5)^2) <= 9){
      df$label[i] <- 1
    }
    else{
      df$label[i] <- 0
    }
  }
  return(df)
}

#data frame with 20 datapoints
df_20dataPoints <- data.frame(x = (runif(20, min = 0, max = 10)),
                 y = (runif(20, min = 0, max = 10)),
                 label = 1:20)

#generating class labels
df_20dataPoints <- generateLabels(df_20dataPoints)

#dataframe with 100 datapoints
df_100dataPoints <- data.frame(x = (runif(100, min = 0, max = 10)),
                 y = (runif(100, min = 0, max = 10)),
                 label = 1:100)

df_100dataPoints <- generateLabels(df_100dataPoints)
```

b

```
library(ggplot2, warn.conflicts = FALSE)

draw_ggplot <- function(dataframe){
  p <- ggplot(dataframe, aes(x=x, y=y,  colour = label)) +
      geom_point() +
      coord_fixed() +
      theme_bw()
  return(p)
}
#plot for 100 data points
draw_ggplot(df_20dataPoints)
```



```r
#plot for 100 data points
draw_ggplot(df_100dataPoints)
```



c

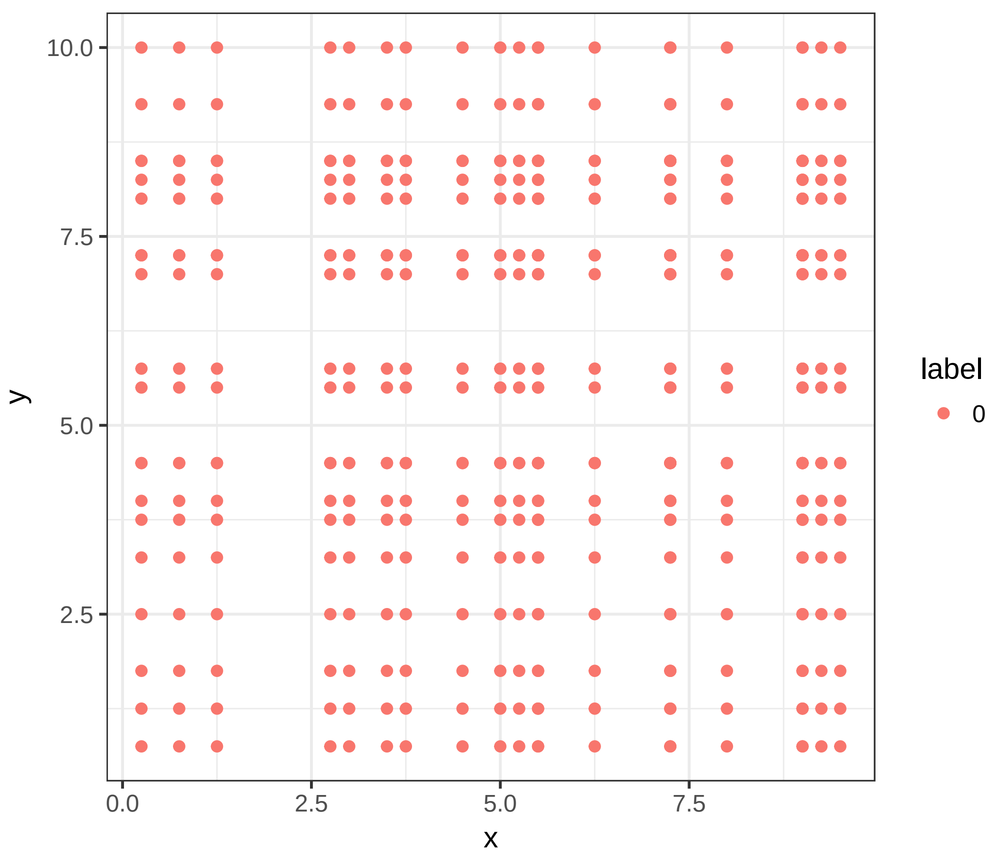
```
library(caret)
#Training  20 data points using Decision Tree on
decisionTree_20datapoints <- train(y = as.factor(df_20dataPoints$label), x = df_20dataPoints[, 1:2],
                                   method = "rpart", tuneGrid = data.frame(cp=0.01))

#Training  100 data points using Decision Tree on
decisionTree_100datapoints <- train(y = as.factor(df_100dataPoints$label), x = df_100dataPoints[, 1:2],
                                    method = "rpart", tuneGrid = data.frame(cp=0.01))

#Training  20 data points using Random forest
rf_20datapoints <- train(as.factor(label)~., data = df_20dataPoints,
                         method = 'rf', trControl = ctrl)

#Training  100 data points using Random forest
rf_100datapoints <- train(as.factor(label)~., data = df_100dataPoints,
                         method = 'rf', trControl = ctrl)

#Training 20 data points using SVM I (linear)
trctrl <- trainControl(method = "repeatedcv", number = 5, repeats = 3)
set.seed(3233)
svm_Linear_20datapoints <- train(as.factor(label)~., data = df_20dataPoints , method = "svmLinear",
                 trControl=trctrl,
                  preProcess = c("center", "scale"),
                 tuneLength = 10)
```

```
##
## Attaching package: 'kernlab'

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

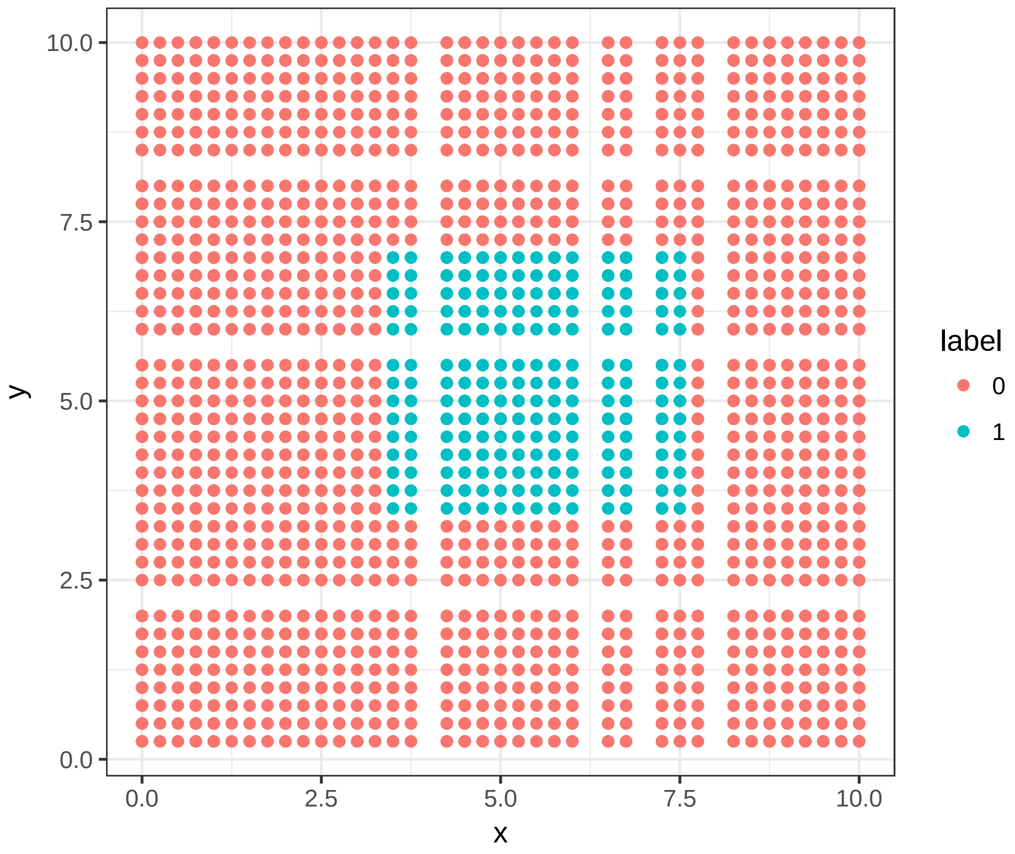
```
#Training 100 data points using  SVM I (linear)
trctrl <- trainControl(method = "repeatedcv", number = 5, repeats = 3)
set.seed(3233)
svm_Linear_100datapoints <- train(as.factor(label)~., data = df_100dataPoints , method = "svmLinear",
                              trControl=trctrl,
                              preProcess = c("center", "scale"),
                              tuneLength = 10)

#Training 20 data points using   SVM II (radial kernel)
set.seed(3233)
svmkernel_20datapoints <- train(as.factor(label)~., data = df_20dataPoints , method = "svmRadial",
                           trControl=trctrl,
                           preProcess = c("center", "scale"),
                           tuneLength = 10)

#Training 100 data points using   SVM II (radial kernel)
set.seed(3233)
svmKernel_100datapoints <- train(as.factor(label)~., data = df_100dataPoints , method = "svmRadial",
                 trControl=trctrl,
                  preProcess = c("center", "scale"),
                 tuneLength = 10)
```

d

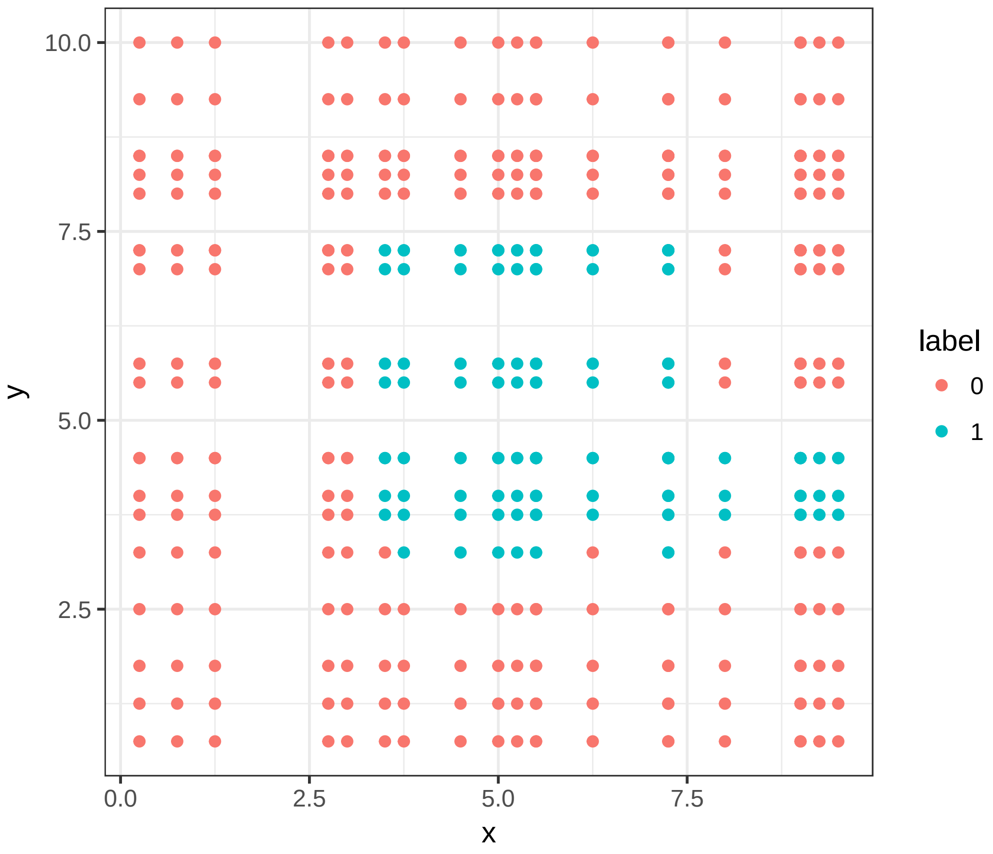
```
x <- sample(seq(from = 0, to = 10, by = 0.25), size = 20, replace = TRUE)
y <- sample(seq(from = 0, to = 10, by = 0.25), size = 20, replace = TRUE)

test_data_20points <- expand.grid(x = x, y = y, KEEP.OUT.ATTRS = FALSE)

x <- sample(seq(from = 0, to = 10, by = 0.25), size = 100, replace = TRUE)
y <- sample(seq(from = 0, to = 10, by = 0.25), size = 100, replace = TRUE)
test_data_100points <- expand.grid(x = x, y = y, KEEP.OUT.ATTRS = FALSE)

#classifyig test using 20 data points trained using Decision Tree on
predicated_dt_20points <- predict(decisionTree_20datapoints, test_data_20points)
test_data_20points$label <- predicated_dt_20points
draw_ggplot(test_data_20points)
```



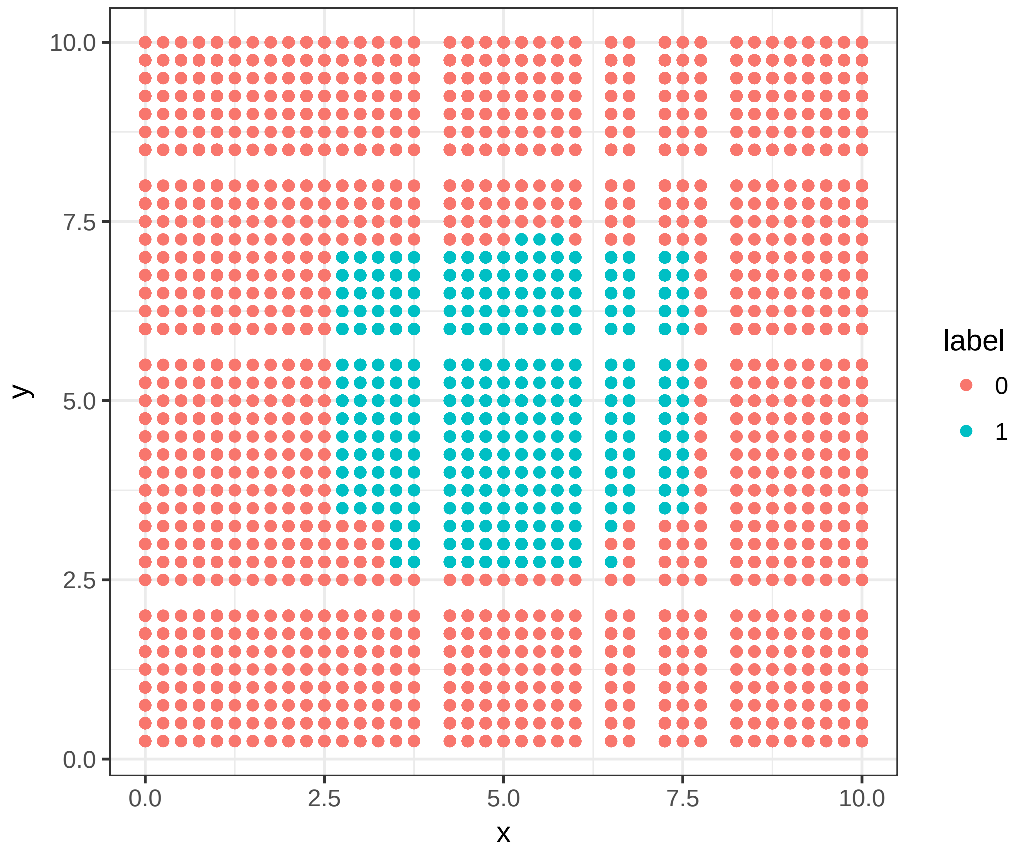
```
#classifyig test using 100 data points trained using Decision Tree on
test_data_100points$label  <- predict(decisionTree_100datapoints, test_data_100points)
draw_ggplot(test_data_100points)
```



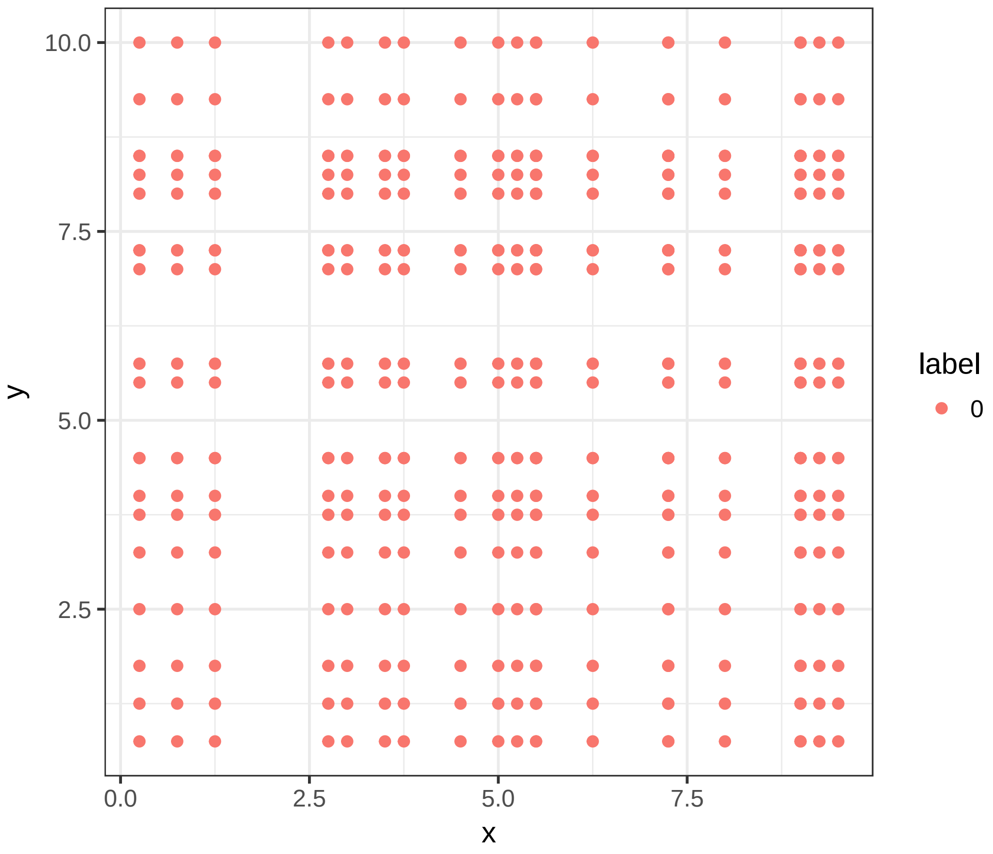
```
#classifying test data using 20 trained data trained using random forests
test_data_20points$label <- predict(rf_20datapoints, test_data_20points )
draw_ggplot(test_data_20points)
```



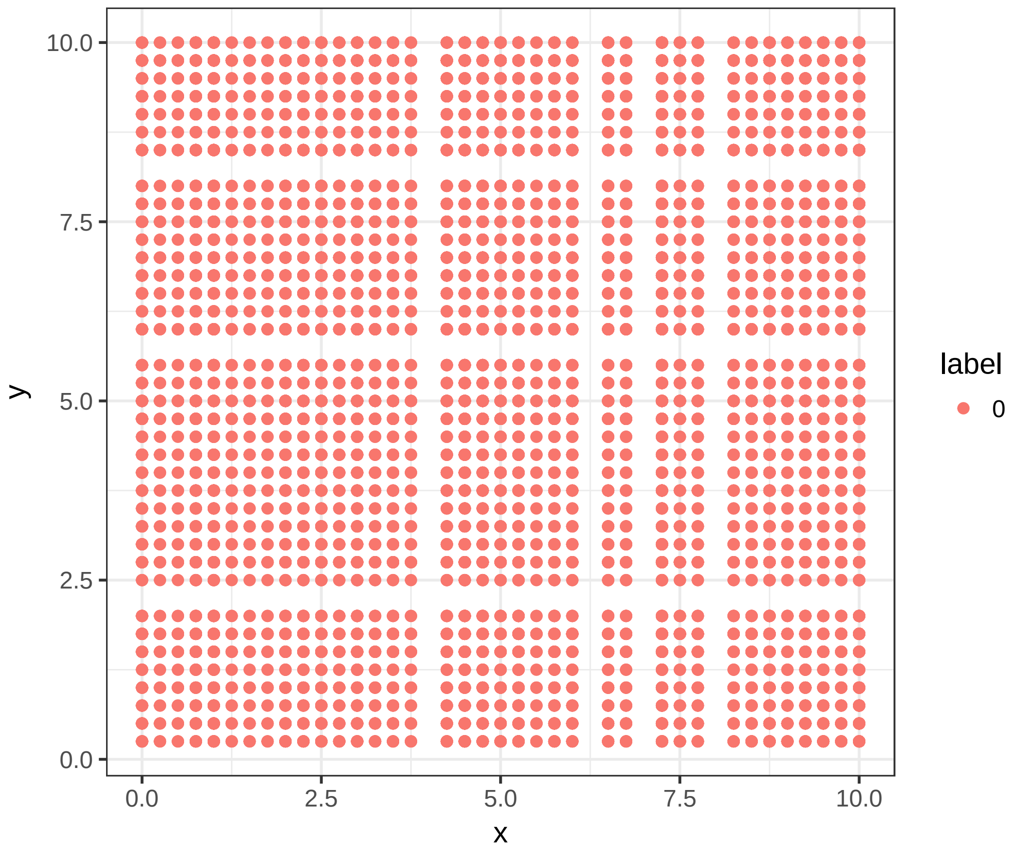
```r
#classifying test data using 100 trained data trained using random forests
test_data_100points$label <- predict(rf_100datapoints, test_data_100points )
draw_ggplot(test_data_100points)
```



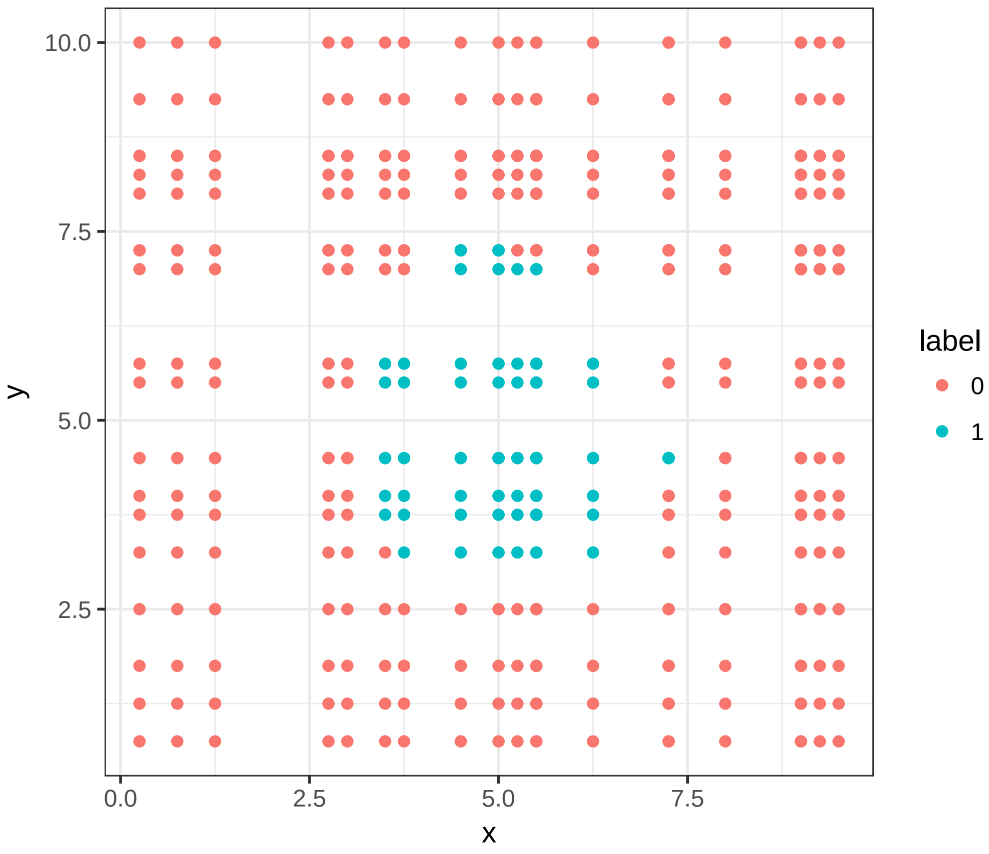
```
#classifying test data using 20 trained data trained using SVM I (linear)
test_data_20points$label <- predict(svm_Linear_20datapoints, test_data_20points )
draw_ggplot(test_data_20points)
```



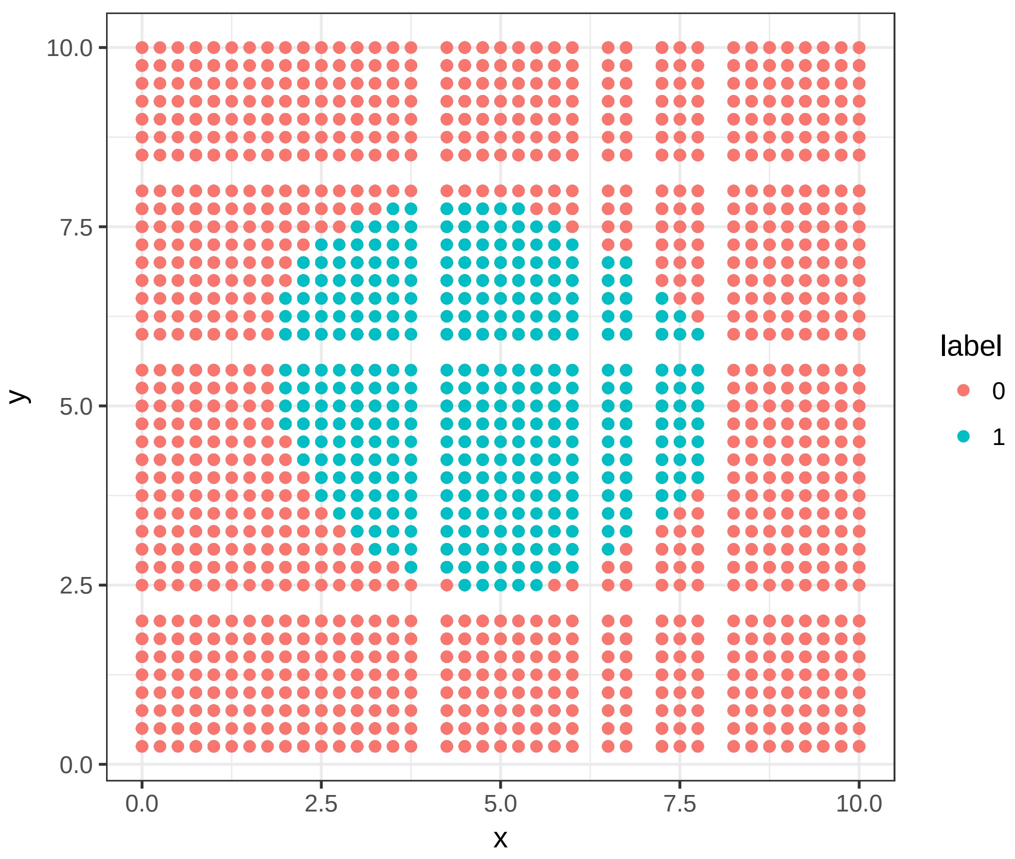
```
#classifying test data using 100 trained data trained using SVM I (linear)
test_data_100points$label <- predict(svm_Linear_100datapoints, test_data_100points )
draw_ggplot(test_data_100points)
```



```
#classifying test data using 20 trained data trained using SVM II (radial kernel)
test_data_20points$label <- predict(svmkernel_20datapoints, test_data_20points )
draw_ggplot(test_data_20points)
```



```
#classifying test data using 100 trained data trained using SVM II (radial kernel)
test_data_100points$label <- predict(svmKernel_100datapoints, test_data_100points )
draw_ggplot(test_data_100points)
```



e Interpret the results. Which classifiers were able to recognize the circular shape

- Decision Tree classifier with training data of size 20 did not predict any positive value ie all the predication were negative(0s). But with training data of size 100, it recorgnised a square.
- Random forest classifier recorgnised some shape which seemed like rectangle
- classfier SVM Linear for training data size 20 did not predict any positive value ie all labels were 0s.
- Classifier ***SVM II (radial kernel)*** with training dats of size 100 was able to recorgnise the circular shape.

are the results what you would have expected?

- The results are not what I expected, I expected all of the classifier to produce a circular shape.

Was the original training data size important and how did it influence the results?

- Yes the origin data size was important because classifiers that used large data size ie 100 produced a more accurate sharp compared to those with small training data size.

“‘