<u>Homework 2: Classification</u>

Gayathri Prerepa- gp2254

Loading the libraries

library(ggplot2)

library(dplyr)

library(tidyverse)

library(lubridate)

library(plotly)

library(naivebayes)

library(psych)

library(klaR)

library(caret)

library(e1071)

library(plotROC)

library(pROC)

library(tidyr)

library(class)

library(MLmetrics)

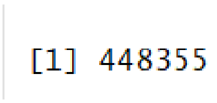
library(Metrics)

Removing Values which are entered as NOT REPORTED

df$PD_Code <- data$PD_CD

df_c <- subset(df, Borough!= "NOT REPORTED" & Latitude !="NOT REPORTED" & Longtitude != "NOT REPORTED")

print(nrow(df_c))

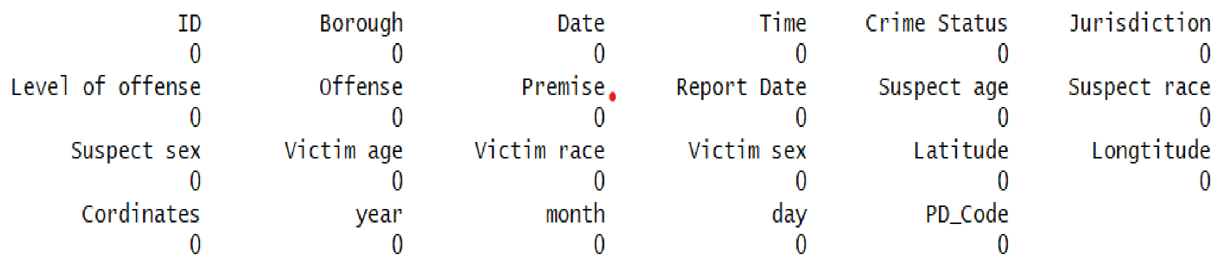


```
[1] 448355
```

Now the dataframe has 448,355 rows, initially it was 449,506

Check for any null values in each column and returns the sum

colSums(is.na(df_c))



```
              ID          Borough             Date             Time     Crime Status     Jurisdiction
               0                0                0                0                0                0
Level of offense          Offense          Premise      Report Date      Suspect age     Suspect race
               0                0                0                0                0                0
     Suspect sex       Victim age      Victim race       Victim sex         Latitude       Longtitude
               0                0                0                0                0                0
       Cordinates             year            month              day          PD_Code
               0                0                0                0                0
```

Assigning class values to each borough as 1,2,3,4,5 for classification

df_c$class <- df_c$Borough, levels = c("BRONX","BROOKLYN", "MANHATTAN", "QUEENS", "STATEN ISLAND"), labels = c(1,2,3,4,5))

df_c$class <- df_c$Borough

data_c<-select(df_c, c("Borough","Latitude","Longtitude","PD_Code", "class"))

Splitting data into train and test set (75-25 distribution)

data_split <- createDataPartition(y = data_c$class, p = 0.75, list = FALSE)

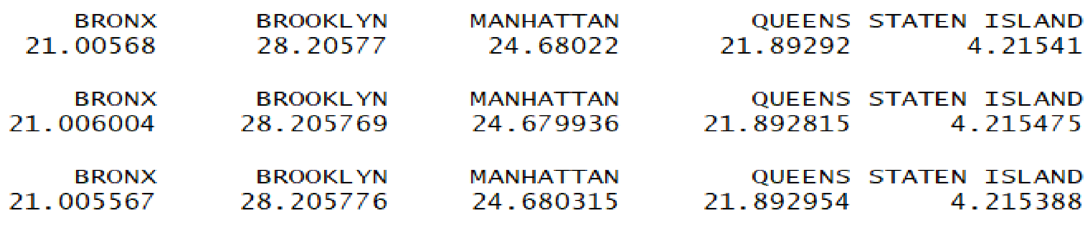
training <- data_c[data_split,]

testing <- data_c[-data_split,]

prop.table(table(data_c$class)) * 100

prop.table(table(testing$class)) * 100

prop.table(table(training$class)) * 100



```
    BRONX     BROOKLYN     MANHATTAN        QUEENS STATEN ISLAND
 21.00568     28.20577      24.68022      21.89292       4.21541

    BRONX     BROOKLYN     MANHATTAN        QUEENS STATEN ISLAND
21.006004    28.205769     24.679936     21.892815      4.215475

    BRONX     BROOKLYN     MANHATTAN        QUEENS STATEN ISLAND
21.005567    28.205776     24.680315     21.892954      4.215388
```

Assigning training data and class to x and y variables

x <- training[,1:4]

y <- training$class

<u>Classification Models:</u>

1. Training and testing the dataset with Naive Baye's model

The Bayes Theorem is used to create a Naive Bayes classifier. It calculates probabilities for each class, such as the likelihood that a certain record or data point belongs to that class.

The most likely class is defined as the one having the highest probability.

Given the class, the Bayes rule works on the assumption that the attributes are conditionally independent.

Code and results:

train_control <- trainControl(method = "cv", number = 10)

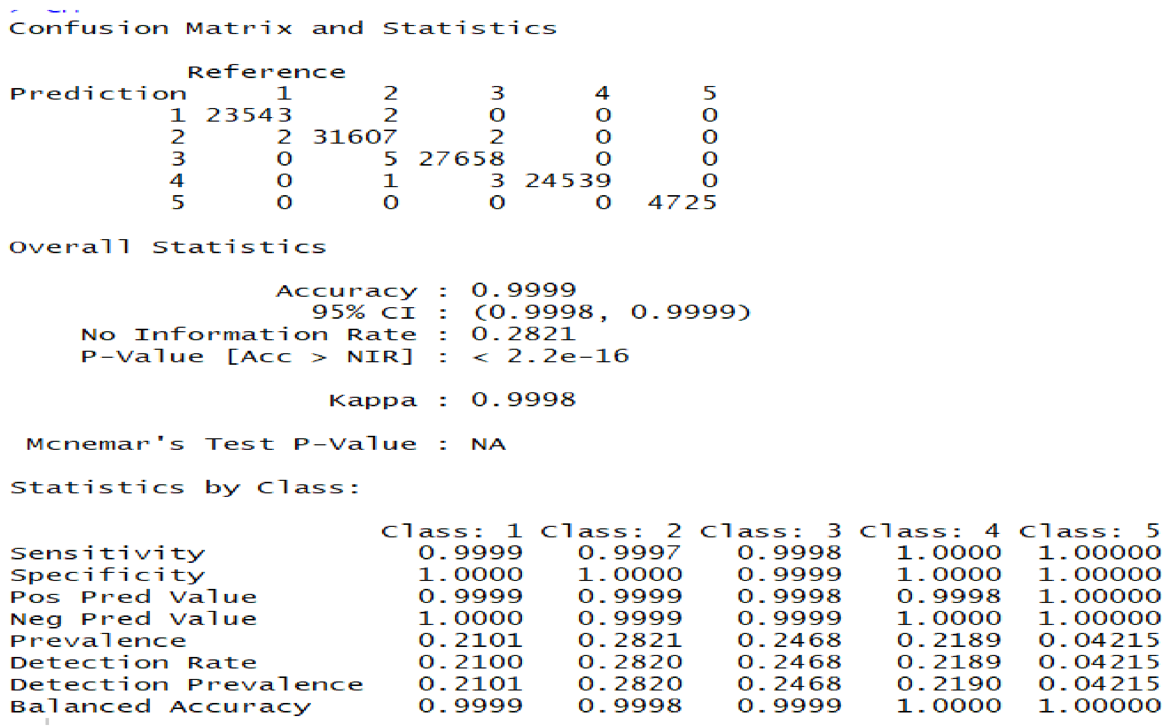
nb.m1 <- train(x = x_nb,y = y_nb,method = "nb",trControl = train_control)

predi<- data.frame(matrix(ncol = 1, nrow = 112087))

predi$class <- predict(nb.m1, newdata = testing_nb)

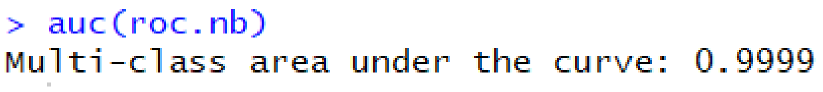
pred = subset(predi, select = -c(1))

CM<-confusionMatrix(pred$class1, testing_nb$class1)



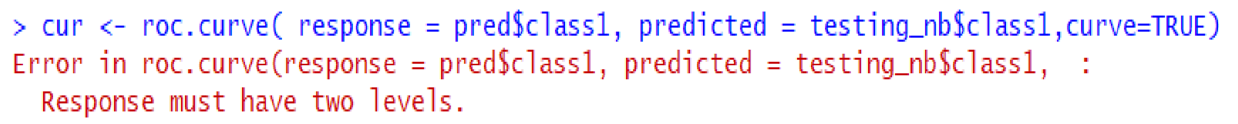
```
Confusion Matrix and Statistics

          Reference
Prediction     1     2     3     4     5
         1 23543     2     0     0     0
         2     2 31607     2     0     0
         3     0     5 27658     0     0
         4     0     1     3 24539     0
         5     0     0     0     0  4725

Overall Statistics

               Accuracy : 0.9999
                 95% CI : (0.9998, 0.9999)
    No Information Rate : 0.2821
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.9998

 Mcnemar's Test P-Value : NA

Statistics by Class:

                     Class: 1 Class: 2 Class: 3 Class: 4 Class: 5
Sensitivity            0.9999   0.9997   0.9998   1.0000  1.00000
Specificity            1.0000   1.0000   0.9999   1.0000  1.00000
Pos Pred Value         0.9999   0.9999   0.9998   0.9998  1.00000
Neg Pred Value         1.0000   0.9999   0.9999   1.0000  1.00000
Prevalence             0.2101   0.2821   0.2468   0.2189  0.04215
Detection Rate         0.2100   0.2820   0.2468   0.2189  0.04215
Detection Prevalence   0.2101   0.2820   0.2468   0.2190  0.04215
Balanced Accuracy      0.9999   0.9998   0.9999   1.0000  1.00000
```

roc.nb <- multiclass.roc(testing$class, pred$class) auc(roc.nb)



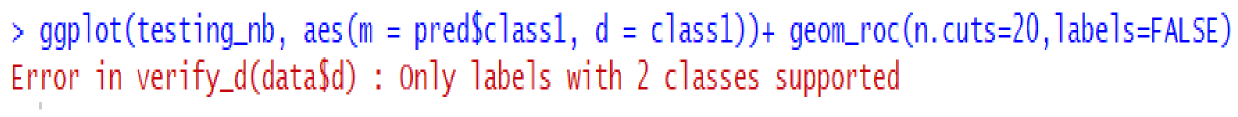
```
> auc(roc.nb)
Multi-class area under the curve: 0.9999
```

cur<-roc.curve(scores.class0=pred$class,weights.class0=testing$class, curve=TRUE)



```
> cur <- roc.curve( response = pred$class1, predicted = testing_nb$class1,curve=TRUE)
Error in roc.curve(response = pred$class1, predicted = testing_nb$class1,  :
  Response must have two levels.
```

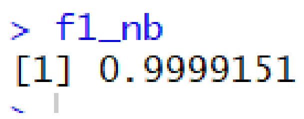
ggplot(testing_nb, aes(m = pred$class1, d = class1))+ geom_roc(n.cuts=20,labels=FALSE)



```
> ggplot(testing_nb, aes(m = pred$class1, d = class1))+ geom_roc(n.cuts=20,labels=FALSE)
Error in verify_d(data$d) : Only labels with 2 classes supported
```

f1_nb <- F1_Score(pred$class1, testing_nb$class1)

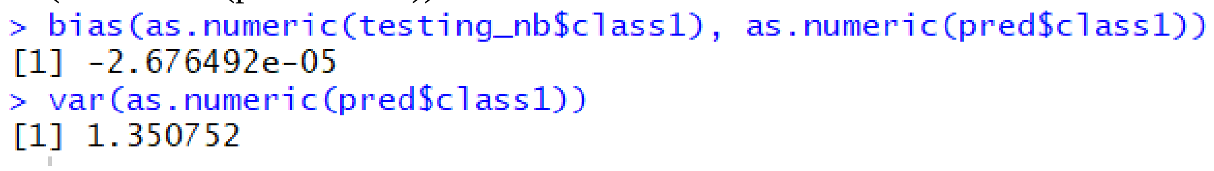
print(f1_nb)



```
> f1_nb
[1] 0.9999151
```

bias(as.numeric(testing_nb$class1), as.numeric(pred$class1))

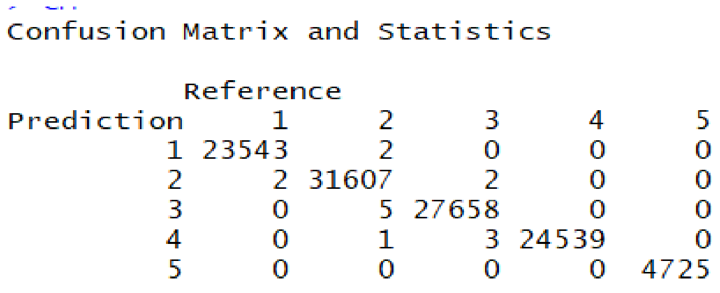
var(as.numeric(pred$class1))



```
> bias(as.numeric(testing_nb$class1), as.numeric(pred$class1))
[1] -2.676492e-05
> var(as.numeric(pred$class1))
[1] 1.350752
```

<u>Summary of naïve bayes model:</u>

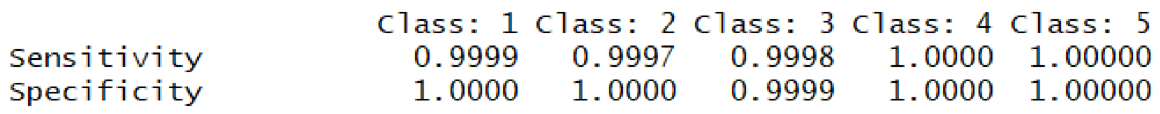
1) **AUC: 0.99**
2) **F1 score = 0.99**
3) **Confusion matrix**



```
Confusion Matrix and Statistics

          Reference
Prediction     1     2     3     4     5
         1 23543     2     0     0     0
         2     2 31607     2     0     0
         3     0     5 27658     0     0
         4     0     1     3 24539     0
         5     0     0     0     0  4725
```

4) **Accuracy = 0.99**
5) **Specificity**

6) Recall (Sensitivity)



```
                     Class: 1 Class: 2 Class: 3 Class: 4 Class: 5
Sensitivity            0.9999   0.9997   0.9998   1.0000  1.00000
Specificity            1.0000   1.0000   0.9999   1.0000  1.00000
```

Despite the fact, this independence requirement is frequently violated in practice, naive Bayes classification accuracy is generally high due to this reason.

So, it would be less biased and hence likely to overfit.

2. <u>Training and testing the dataset with Support Vector Machine</u>

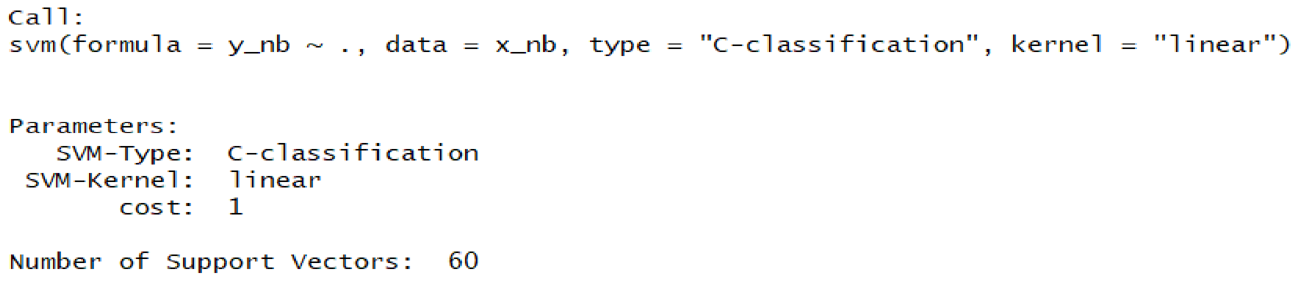
Support vector machines are a type of supervised learning algorithms for classification. SVM categorizes data points by mapping them to a high-dimensional feature space, even when the data is not otherwise linearly separable.

SVM is a basic concept: The algorithm generates a line or hyperplane that divides the data into categories (classes).

<u>Code and results:</u>

classifier = svm(formula = y_nb~. ,data = x_nb,type = 'C-classification',kernel = 'linear')

classifier



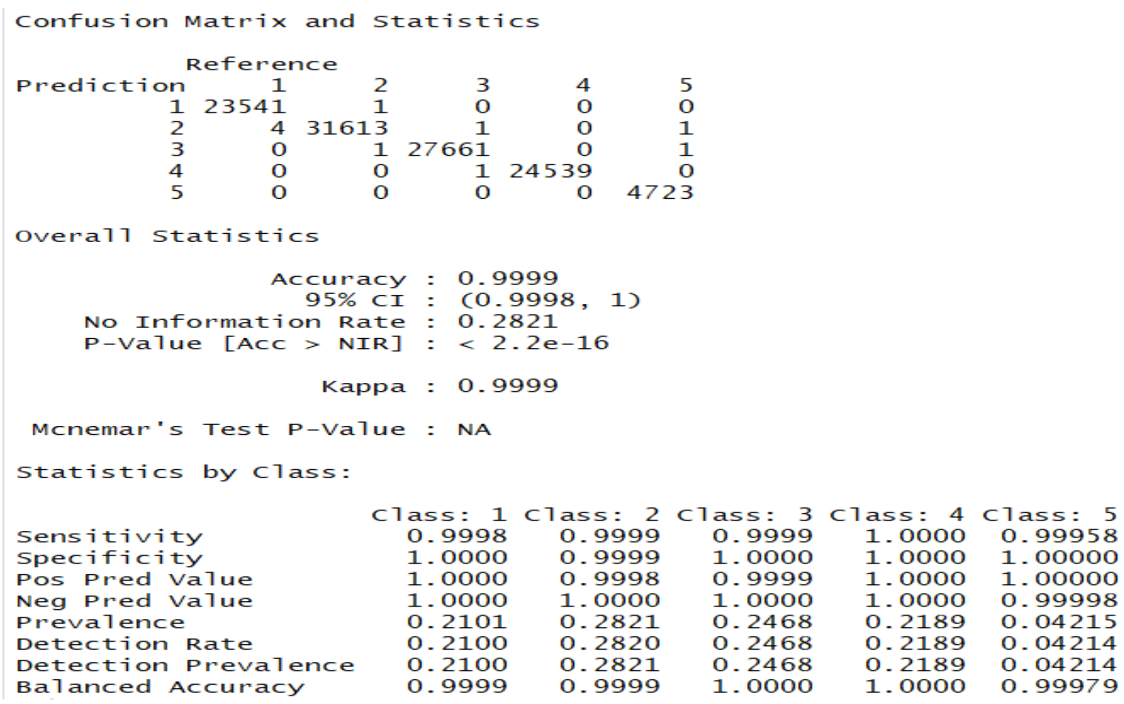
```
Call:
svm(formula = y_nb ~ ., data = x_nb, type = "C-classification", kernel = "linear")


Parameters:
   SVM-Type:  C-classification
 SVM-Kernel:  linear
       cost:  1

Number of Support Vectors:  60
```

y_pred = predict(classifier, newdata = testing)

y_pre<- data.frame(matrix(ncol = 1, nrow = 112088))

y_pre$class <- y_pred

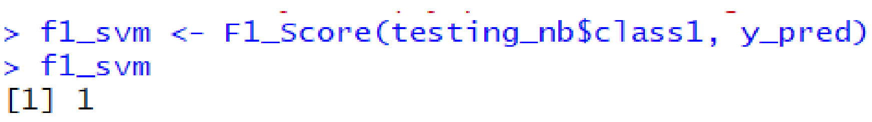
cm_svm<-confusionMatrix(as.factor(testing_nb$class1),as.factor(y_pred), mode ="prec_recall") cm_svm



```
Confusion Matrix and Statistics

          Reference
Prediction     1     2     3     4     5
         1 23541     1     0     0     0
         2     4 31613     1     0     1
         3     0     1 27661     0     1
         4     0     0     1 24539     0
         5     0     0     0     0  4723

Overall Statistics
                                          
               Accuracy : 0.9999          
                 95% CI : (0.9998, 1)     
    No Information Rate : 0.2821          
    P-Value [Acc > NIR] : < 2.2e-16       
                                          
                  Kappa : 0.9999          
                                          
 Mcnemar's Test P-Value : NA              

Statistics by Class:

                     Class: 1 Class: 2 Class: 3 Class: 4 Class: 5
Sensitivity            0.9998   0.9999   0.9999   1.0000  0.99958
Specificity            1.0000   0.9999   1.0000   1.0000  1.00000
Pos Pred Value         1.0000   0.9998   0.9999   1.0000  1.00000
Neg Pred Value         1.0000   1.0000   1.0000   1.0000  0.99998
Prevalence             0.2101   0.2821   0.2468   0.2189  0.04215
Detection Rate         0.2100   0.2820   0.2468   0.2189  0.04214
Detection Prevalence   0.2100   0.2821   0.2468   0.2189  0.04214
Balanced Accuracy      0.9999   0.9999   1.0000   1.0000  0.99979
```

f1_svm <- F1_Score(testing_nb$class1, y_pred)

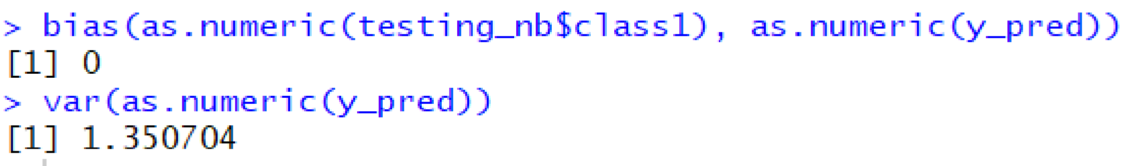
f1_svm



```
> f1_svm <- F1_Score(testing_nb$class1, y_pred)
> f1_svm
[1] 1
```

bias(as.numeric(testing_nb$class1), as.numeric(y_pred))

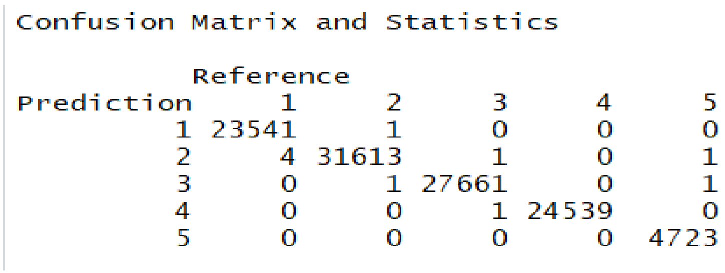
var(as.numeric(y_pred))



```
> bias(as.numeric(testing_nb$class1), as.numeric(y_pred))
[1] 0
> var(as.numeric(y_pred))
[1] 1.350704
```

Summary of SVM model:

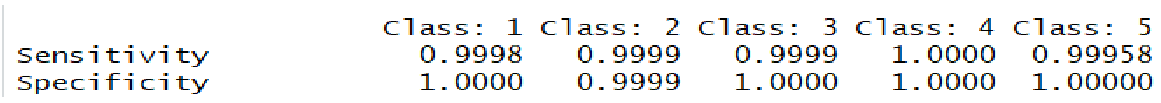
1) **F1 score = 1**
2) **Confusion matrix**



```
Confusion Matrix and Statistics

          Reference
Prediction     1     2     3     4     5
         1 23541     1     0     0     0
         2     4 31613     1     0     1
         3     0     1 27661     0     1
         4     0     0     1 24539     0
         5     0     0     0     0  4723
```

3) **Accuracy = 0.99**

4) **Specificity**
5) **Recall (Sensitivity)**



```
                     Class: 1 Class: 2 Class: 3 Class: 4 Class: 5
Sensitivity            0.9998   0.9999   0.9999   1.0000  0.99958
Specificity            1.0000   0.9999   1.0000   1.0000  1.00000
```

1. An overfit SVM, as in this case, has a high accuracy with the training set but struggles with new, previously unknown samples.
2. This model is highly susceptible to noise, and even minor changes in data point values can cause the classification results to vary.
3. Cross validation and changing test set size could cause the accuracy and performance of the model to vary.

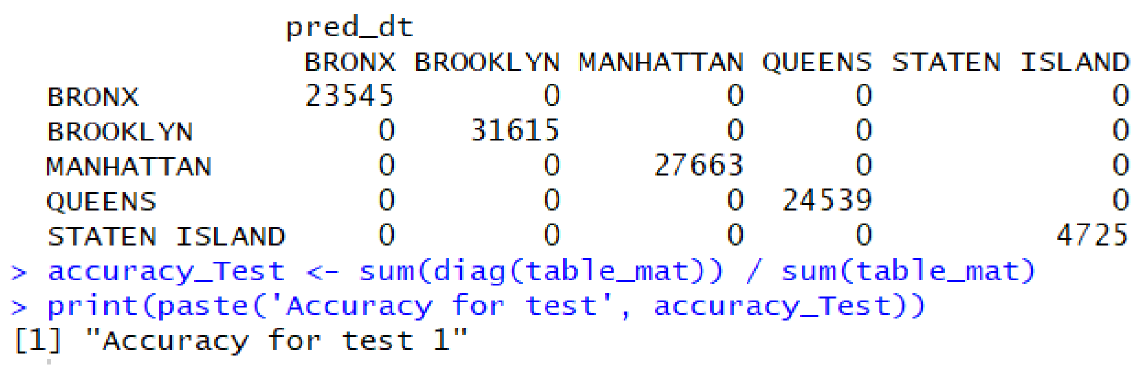
3. <u>Training and testing the dataset with Decision Tree</u>

Decision Trees are a type of Supervised Machine Learning where the data is continuously split according to a certain parameter.

It is a form of probability tree that allows you to make a decision on a given process. You could want to choose between classes A and B, for example.

<u>Code and results:</u>

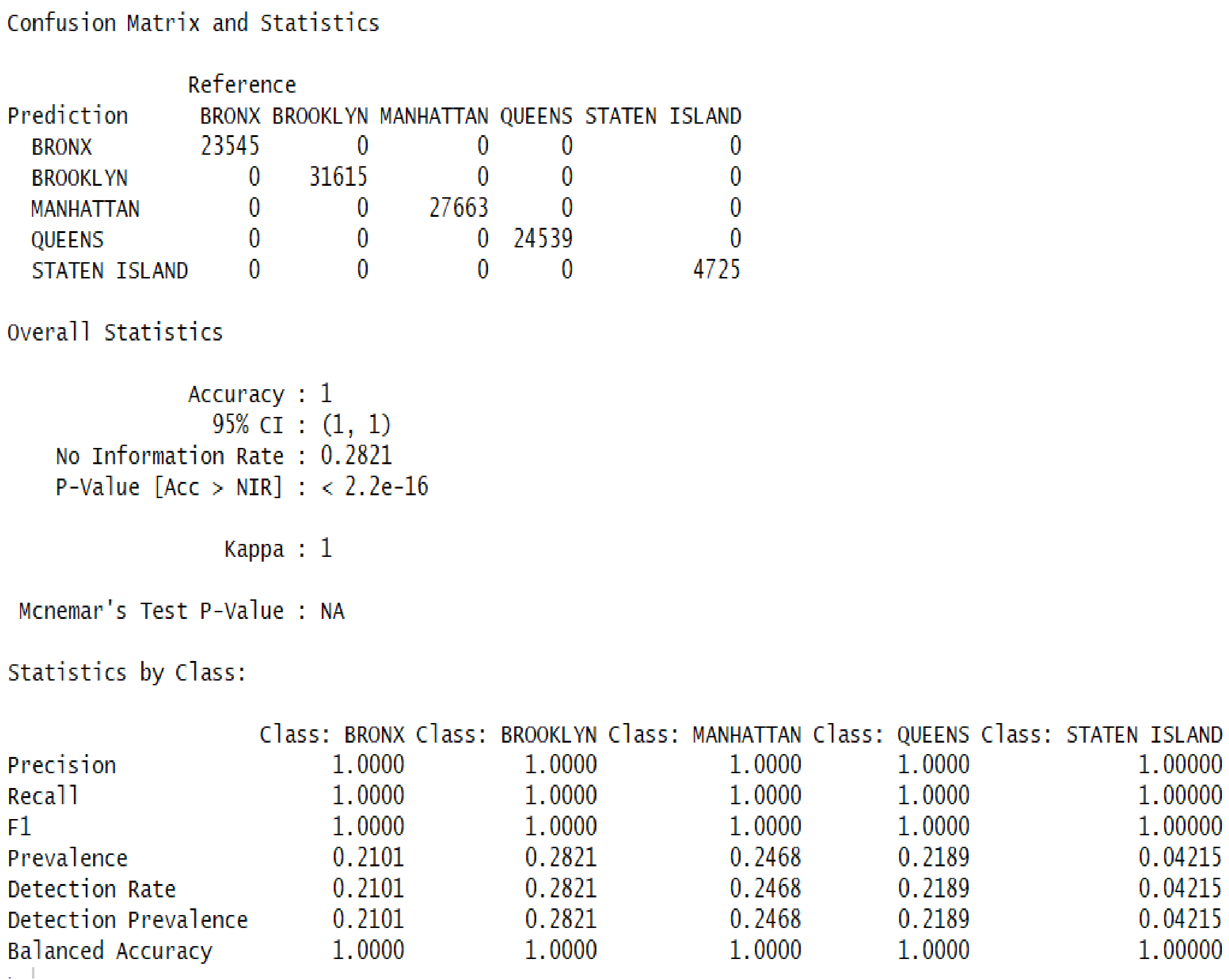
```
fit <- rpart(y~. ,data = x , method = 'class')

pred_dt <-predict(fit, testing, type = 'class')

table_mat=table(testing$class, pred_dt)

table_mat

accuracy_Test <- sum(diag(table_mat)) / sum(table_mat)

print(paste('Accuracy for test', accuracy_Test))
```



```
                   pred_dt
                    BRONX BROOKLYN MANHATTAN QUEENS STATEN ISLAND
  BRONX             23545        0         0      0             0
  BROOKLYN              0    31615         0      0             0
  MANHATTAN             0        0     27663      0             0
  QUEENS                0        0         0  24539             0
  STATEN ISLAND         0        0         0      0          4725
> accuracy_Test <- sum(diag(table_mat)) / sum(table_mat)
> print(paste('Accuracy for test', accuracy_Test))
[1] "Accuracy for test 1"
```

cm_dt<-confusionMatrix(as.factor(testing$class),as.factor(pred_dt),mode ="prec_recall")

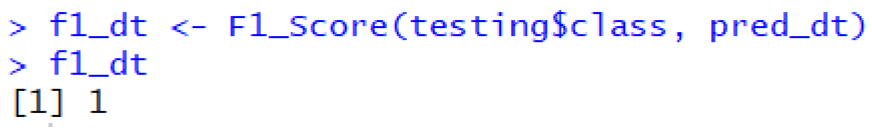
cm_dt



```
Confusion Matrix and Statistics

               Reference
Prediction      BRONX BROOKLYN MANHATTAN QUEENS STATEN ISLAND
  BRONX         23545        0         0      0             0
  BROOKLYN          0    31615         0      0             0
  MANHATTAN         0        0     27663      0             0
  QUEENS            0        0         0  24539             0
  STATEN ISLAND     0        0         0      0          4725

Overall Statistics

               Accuracy : 1
                 95% CI : (1, 1)
    No Information Rate : 0.2821
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 1

 Mcnemar's Test P-Value : NA

Statistics by Class:

                     Class: BRONX Class: BROOKLYN Class: MANHATTAN Class: QUEENS Class: STATEN ISLAND
Precision                  1.0000          1.0000           1.0000        1.0000              1.00000
Recall                     1.0000          1.0000           1.0000        1.0000              1.00000
F1                         1.0000          1.0000           1.0000        1.0000              1.00000
Prevalence                 0.2101          0.2821           0.2468        0.2189              0.04215
Detection Rate             0.2101          0.2821           0.2468        0.2189              0.04215
Detection Prevalence       0.2101          0.2821           0.2468        0.2189              0.04215
Balanced Accuracy          1.0000          1.0000           1.0000        1.0000              1.00000
```

f1_dt <- F1_Score(testing$class, pred_dt)

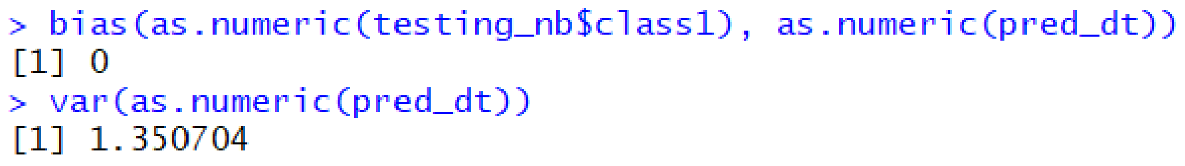
f1_dt



```
> f1_dt <- F1_Score(testing$class, pred_dt)
> f1_dt
[1] 1
```

bias(as.numeric(testing_nb$class1), as.numeric(pred_dt))

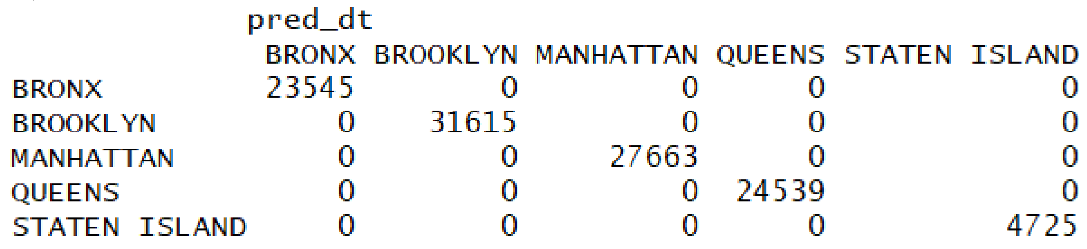
var(as.numeric(pred_dt))



```
> bias(as.numeric(testing_nb$class1), as.numeric(pred_dt))
[1] 0
> var(as.numeric(pred_dt))
[1] 1.350704
```

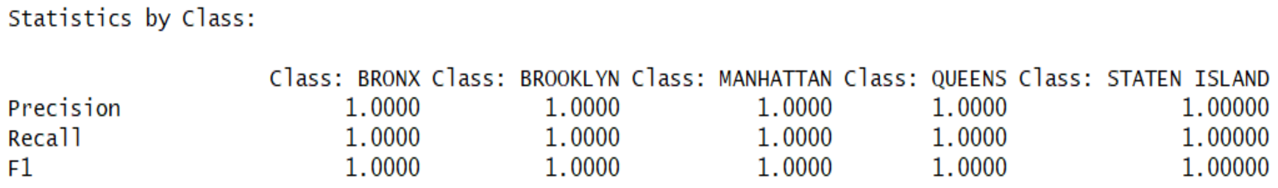
Summary of Decision Tree model:

1) **F1 score = 1**
2) **Confusion matrix**



```
               pred_dt
                BRONX BROOKLYN MANHATTAN QUEENS STATEN ISLAND
  BRONX         23545        0         0      0             0
  BROOKLYN          0    31615         0      0             0
  MANHATTAN         0        0     27663      0             0
  QUEENS            0        0         0  24539             0
  STATEN ISLAND     0        0         0      0          4725
```

3) **Accuracy = 1**
4) **Specificity**
5) **Recall (Sensitivity)**



```
Statistics by Class:

                    Class: BRONX Class: BROOKLYN Class: MANHATTAN Class: QUEENS Class: STATEN ISLAND
Precision                 1.0000          1.0000           1.0000        1.0000              1.00000
Recall                    1.0000          1.0000           1.0000        1.0000              1.00000
F1                        1.0000          1.0000           1.0000        1.0000              1.00000
```

1. When a decision tree model memorizes the noise in the training data, it becomes overfit and misses essential patterns.
2. When the tree is designed so as to perfectly, it fits all samples in the training data.
3. This might be the reason for high accuracy of the model.

4. Training and testing the dataset with KNN model:

By computing the distance between the test data and all of the training points, KNN tries to predict the proper class for the test data.

Then choose the K number of points that are the nearest to the test data.

If k is set to 5, the classes of the five nearest points will be examined. The majority class is used to make predictions.

Code and results:

train_subset <- training_nb[1:10000,] %>% drop_na()

test_subset <- testing_nb[1:10000,] %>% drop_na()

levels(train_subset$class1) <- c("x1", "x2",'x3','x4','x5')

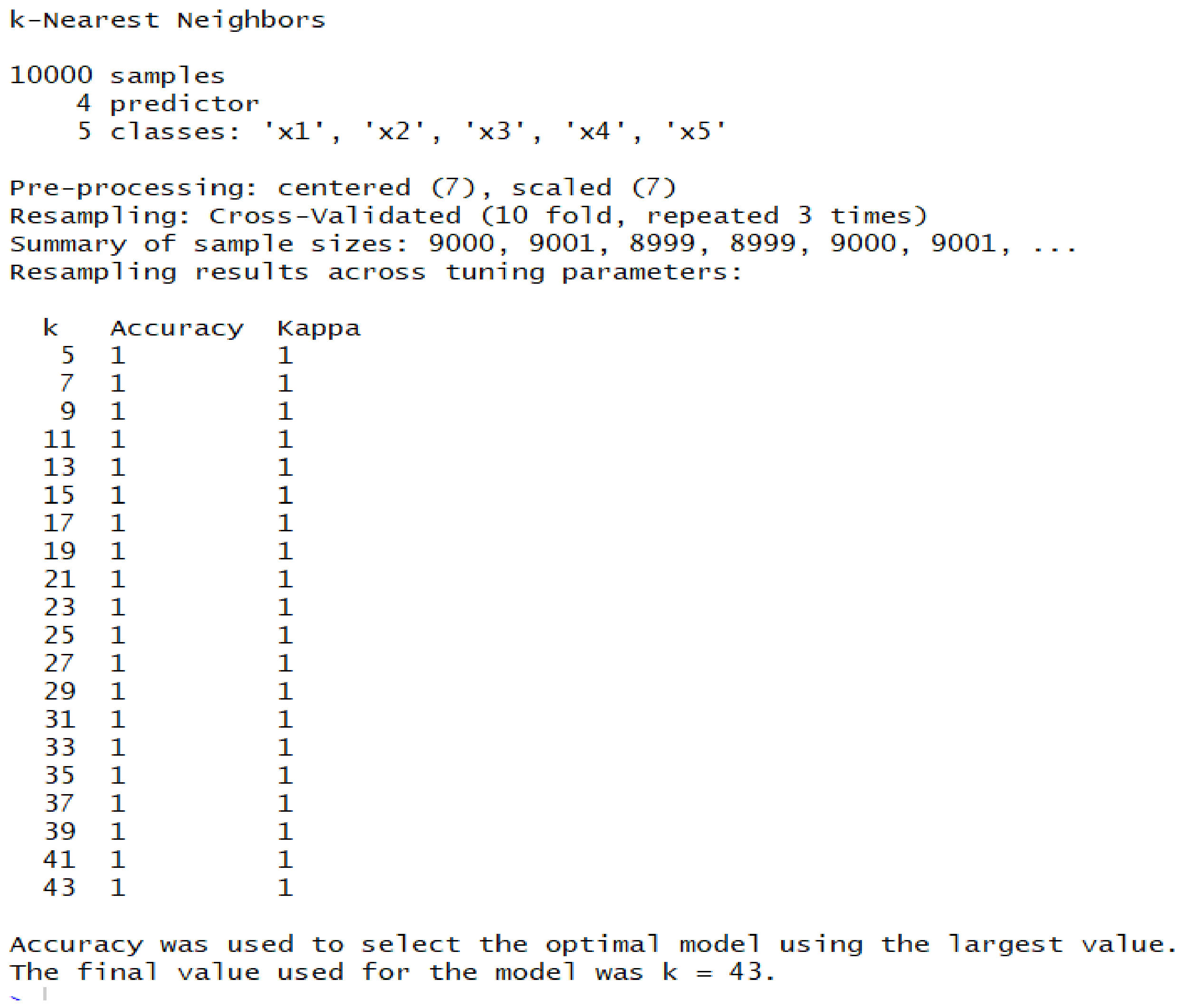
levels(training_nb$class1) <- c("x1", "x2",'x3','x4','x5')

set.seed(340)

fitControl <- trainControl(method = "repeatedcv",number = 10,repeats = 3, classProbs = TRUE)

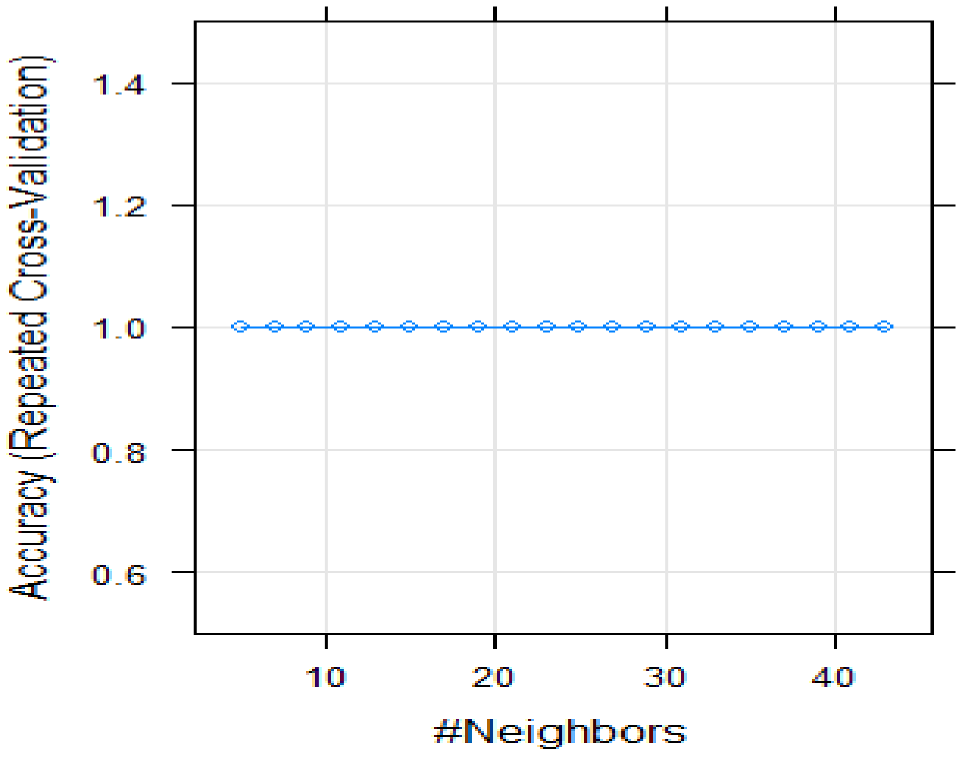
knnfit <- train(class1~., data = train_subset, method = "knn", trControl = fitControl, preProcess = c("center","scale"), tuneLength = 20, na.action="na.omit")

knnfit



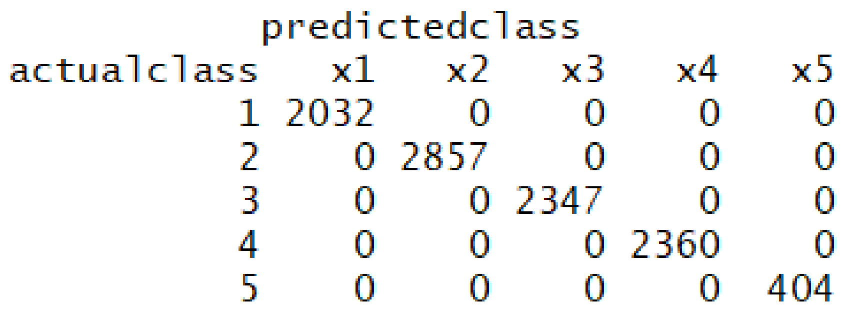
```
k-Nearest Neighbors

10000 samples
    4 predictor
    5 classes: 'x1', 'x2', 'x3', 'x4', 'x5'

Pre-processing: centered (7), scaled (7)
Resampling: Cross-Validated (10 fold, repeated 3 times)
Summary of sample sizes: 9000, 9001, 8999, 8999, 9000, 9001, ...
Resampling results across tuning parameters:

  k   Accuracy  Kappa
   5  1         1
   7  1         1
   9  1         1
  11  1         1
  13  1         1
  15  1         1
  17  1         1
  19  1         1
  21  1         1
  23  1         1
  25  1         1
  27  1         1
  29  1         1
  31  1         1
  33  1         1
  35  1         1
  37  1         1
  39  1         1
  41  1         1
  43  1         1

Accuracy was used to select the optimal model using the largest value.
The final value used for the model was k = 43.
```

pred_knn <- predict(knnfit, newdata = test_subset) plot(knnfit)



dataRev <- table(actualclass=test_subset$class1, predictedclass= pred_knn)

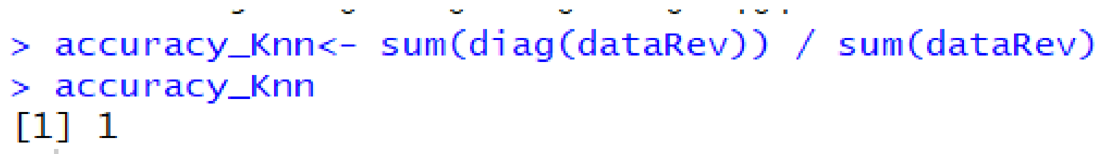
dataRev



```
           predictedclass
actualclass    x1   x2   x3   x4   x5
          1  2032    0    0    0    0
          2     0 2857    0    0    0
          3     0    0 2347    0    0
          4     0    0    0 2360    0
          5     0    0    0    0  404
```

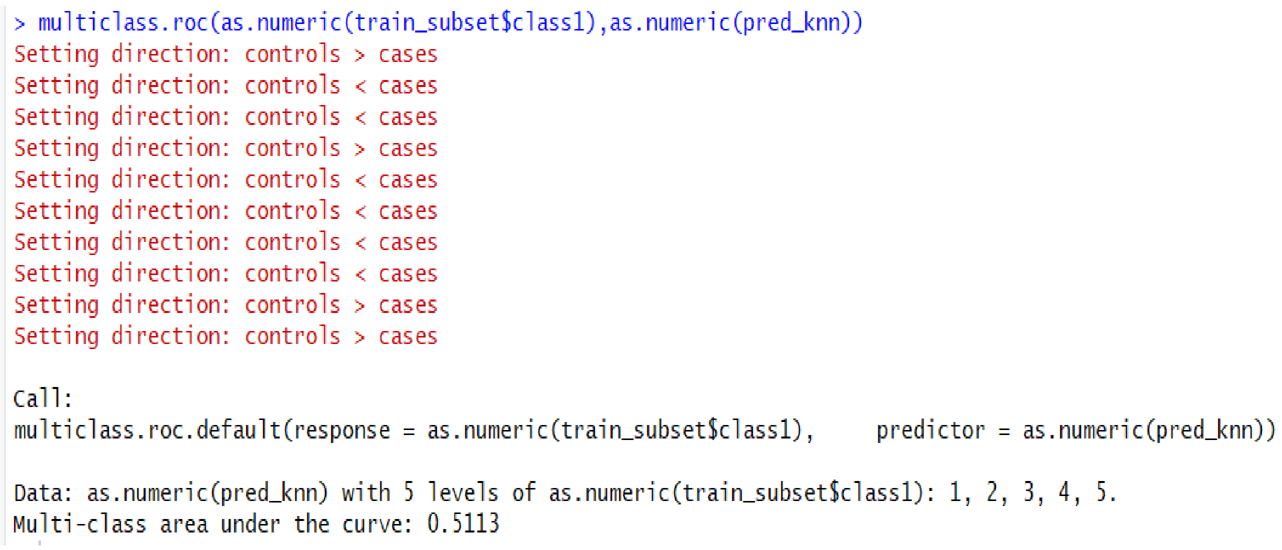
accuracy_Knn<- sum(diag(dataRev)) / sum(dataRev)

accuracy_Knn



```
> accuracy_Knn<- sum(diag(dataRev)) / sum(dataRev)
> accuracy_Knn
[1] 1
```

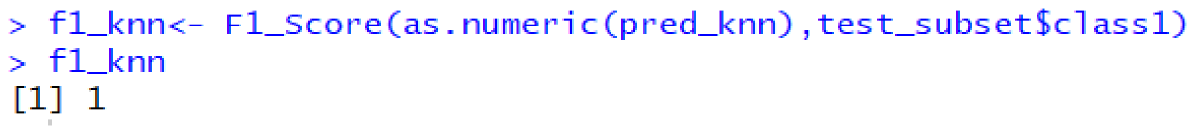
multiclass.roc(as.numeric(train_subset$class1),as.numeric(pred_knn))



```
> multiclass.roc(as.numeric(train_subset$class1),as.numeric(pred_knn))
Setting direction: controls > cases
Setting direction: controls < cases
Setting direction: controls < cases
Setting direction: controls > cases
Setting direction: controls < cases
Setting direction: controls < cases
Setting direction: controls < cases
Setting direction: controls < cases
Setting direction: controls > cases
Setting direction: controls > cases

Call:
multiclass.roc.default(response = as.numeric(train_subset$class1),     predictor = as.numeric(pred_knn))

Data: as.numeric(pred_knn) with 5 levels of as.numeric(train_subset$class1): 1, 2, 3, 4, 5.
Multi-class area under the curve: 0.5113
```

f1_knn<- F1_Score(as.numeric(pred_knn),test_subset$class1)

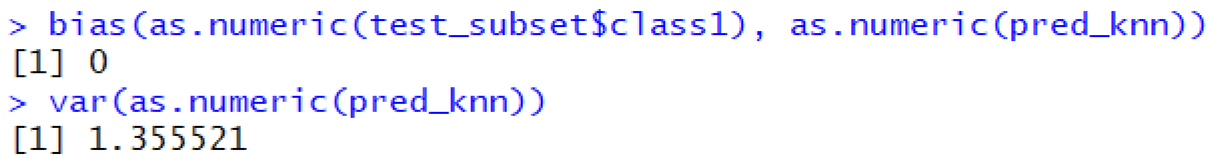
f1_knn



```
> f1_knn<- F1_Score(as.numeric(pred_knn),test_subset$class1)
> f1_knn
[1] 1
```

bias(as.numeric(test_subset$class1), as.numeric(pred_knn))

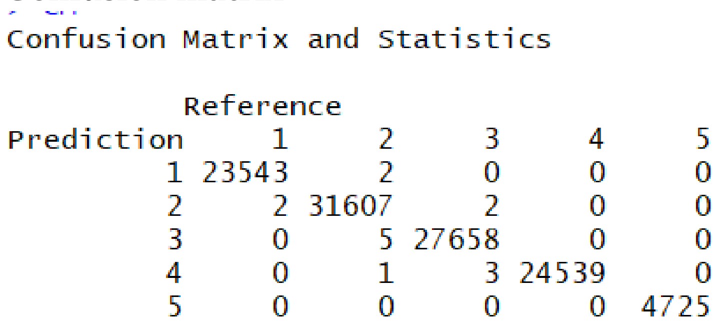
var(as.numeric(pred_knn))



```
> bias(as.numeric(test_subset$class1), as.numeric(pred_knn))
[1] 0
> var(as.numeric(pred_knn))
[1] 1.355521
```

Summary of KNN model:

1) **AUC: 0.5133**
2) **F1 score = 1**
3) **Confusion matrix**



```
Confusion Matrix and Statistics

          Reference
Prediction     1     2     3     4     5
         1 23543     2     0     0     0
         2     2 31607     2     0     0
         3     0     5 27658     0     0
         4     0     1     3 24539     0
         5     0     0     0     0  4725
```

4) **Accuracy = 1**
5) **Specificity**
6) **Recall (Sensitivity)**

1. A small k number can cause overfitting, whereas a large k value can cause underfitting. Overfitting implies that the model performs well on the training data but poorly when additional data is introduced.
2. A too large value of K would make the model ignore the local structure of the distribution, here the value of K was 43

<u>Other inferences of the models:</u>

Bias is defined as the error between average model prediction and the ground truth. Moreover, it describes how well the model matches the training data set. In my analysis, the bias computed was very low which means that the model built is closely matching the training data set.

Variance is the variability in the model prediction—how much the ML function can adjust depending on the given data set. Models with low bias will have high variance, as in case of these models.

F1-score combines the precision and recall of a classifier into a single metric by taking their harmonic mean. The F1 score is equal to one because it is able to perfectly classify each of the observations into a class.

AUC indicates how well the model distinguishes between positive and negative classes. The greater the AUC, the better. When AUC = 1, the classifier is capable of successfully distinguishing between all Positive and Negative class points.