

UNIVERSITY OF WESTMINSTER

MSc BUSINESS INTELLIGENCE AND ANALYTICS (FT)

DATA MINING AND MACHINE LEARNING

HARSH KADIA

INDEX

|  |  |  |
| --- | --- | --- |
| **Sr. No.** | **TITLE** | **Page No.** |
| 1. | Introduction | 3 |
| 2. | Data Set Selection and Visualization | 4 |
| 3. | Formation of Training and Test Sets | 10 |
| 4. | Build Train and Test a Random Forest type Classifier | 14 |
| 5. | Build Train and Test a Naïve Bayes type Classifier | 17 |
| 6. | Build Train and Test a K-NN type Classifier | 23 |
| 7. | Measure Performance | 27 |
| 8. | Appendix | 33 |

# **INTRODUCTION**

Dataset Used: Wine Quality Dataset

Link to the Dataset: <https://archive.ics.uci.edu/ml/datasets/Wine+Quality>

Using Machine Learning approaches, we can predict the quality of wine based on physico‐chemical properties from wine analyses. This report shows the use of certain classifiers like Random Forest, Naïve Bayes and K-NN. These models and results can be used in wine evaluation as the right wine quality evaluation prevents the adulteration of wines and assures the quality in the market. To classify wine, we here use data mining techniques on collected wine sets and evaluate the results on the raw data.

Wine can be classified by human experts or laboratory tests ‐pH or alcohol rate or density determination. We choose the machine approach to identify and bring solution to this problem to make it more automatic and easy handle with accurate results.

## **TASK:1 Data Set Selection and Visualisation**

**Dataset summary of main properties**

* + - Data format: text stored as .csv file
    - Type of Problem: Classification and Regression
    - Total number of instances: 6497
    - Number of Predictors/Attributes: 12
    - Response Variable: Quality

Predictors: fixed acidity, volatile acidity, citric acid, residual sugar, chlorides, free sulphur dioxide, total sulphur dioxide, density, pH, sulphates, alcohol

**SUMMARY** OF MAIN PROPERTIES ARE AS FOLLOWS:

SUMMARY(WINE)

|  |  |  |  |
| --- | --- | --- | --- |
| **PREDICTOR** | **Min** | **Mean** | **Max** |
| Fixed Acidity | 4.600 | 8.320 | 15.90 |
| Volatile Acidity | 0.120 | 0.527 | 1.580 |
| Citric Acid | 0.000 | 0.271 | 1.000 |
| Residual Sugar | 0.900 | 2.539 | 15.50 |
| Chlorides | 0.012 | 0.087 | 0.611 |
| Free Sulphur Dioxide | 1.000 | 15.87 | 72.00 |
| Total Sulphur Dioxide | 6.000 | 46.47 | 289.0 |
| Density | 0.990 | 0.996 | 1.003 |
| pH | 2.740 | 3.311 | 4.010 |
| Sulphates | 0.330 | 0.658 | 2.000 |
| Alcohol | 8.400 | 10.42 | 14.90 |
| Quality | 3.000 | 5.636 | 8.000 |

CHECKING NUMBER OF **UNIQUE VALUES**:

APPLY(WINE,2,FUNCTION(X) LENGTH(UNIQUE(X)))

fixed.acidity volatile.acidity citric.acid residual.sugar chlorides

96 143 80 91 153

free.sulfur.dioxide total.sulfur.dioxide density pH sulphates

60 144 436 89 96

alcohol quality

65 6

**TABLE** OF PREDICTION OF A WINE QUALITY:

TABLE(WINE$QUALITY)

3 4 5 6 7 8

10 53 681 638 199 18

**Visualisation in R of main data set properties**

WINE$QUALITY <- AS.INTEGER(WINE$QUALITY)

PAR(MFROW = C(4,3))

FOR (I IN C(1:11)) {

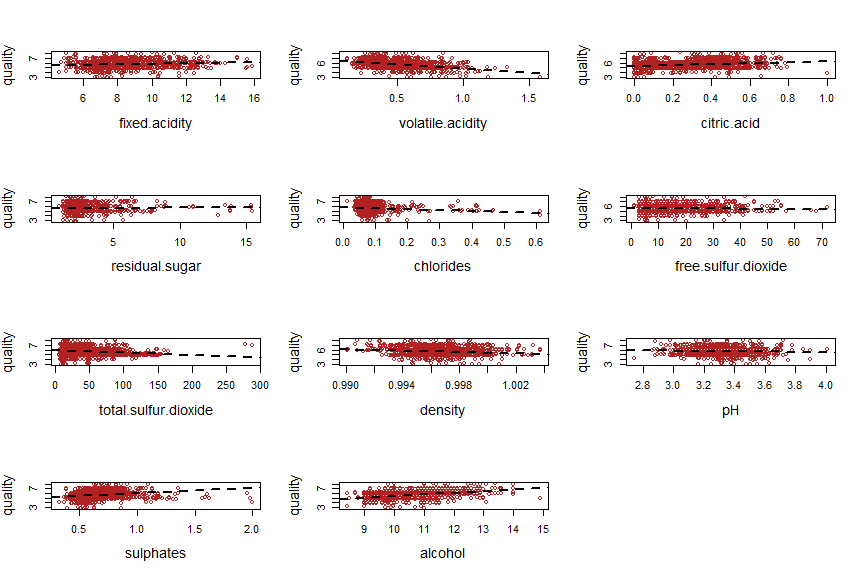
PLOT(WINE[, I], JITTER(WINE[, "QUALITY"]), XLAB = NAMES(WINE)[I],

YLAB = "QUALITY", COL = "FIREBRICK", CEX = 0.8, CEX.LAB = 1.3)

ABLINE(LM(WINE[, "QUALITY"] ~ WINE[ ,I]), LTY = 2, LWD = 2)

}

PAR(MFROW = C(1, 1))



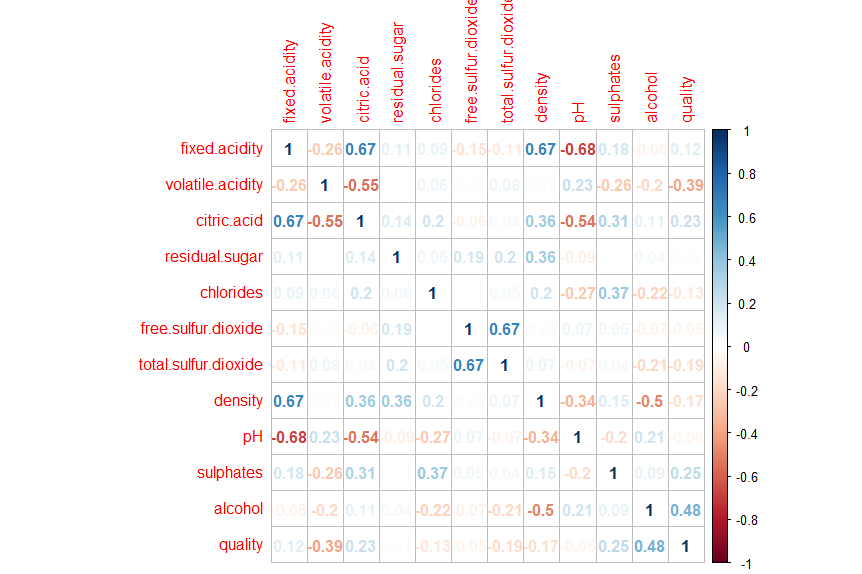
Firstly, Volatile.acidity, sulphates, and alcohol appear to have the strongest relationship with quality. Free.sulfur.dioxide and residual.sugar appear to have the weakest relationships with quality.

Next, plotting the correlations between each variable.

PAR(MFROW = C(1,1))

COR.WINE <- COR(WINE)

CORRPLOT(COR.WINE, METHOD = 'NUMBER')



Weak relationships between quality and residual.sugar, free.sulfur.dioxide, and pH can be seen in the correlation plot. A few predictors are correlated at 0.67. The size of the corrleations will not be a concern.

**FEATURE SELECTION:**

Rank features by importance

DF1<-WINE[,-12] # load the dataset

QUALITY <- WINE$QUALITY # preparing training data

CONTROL <- TRAINCONTROL(METHOD="REPEATEDCV", NUMBER=10, REPEATS=3)

# train the model

MODEL <- TRAIN(QUALITY~., DATA=WINE, METHOD = "TREEBAG",PREPROCESS="SCALE") # estimate variable importance

IMPORTANCE <- VARIMP(MODEL, SCALE=FALSE)

PRINT(IMPORTANCE)

treebag variable importance

Overall

volatile.acidity 1.0841

sulphates 0.8404

alcohol 0.7898

total.sulfur.dioxide 0.5503

citric.acid 0.5188

fixed.acidity 0.5036

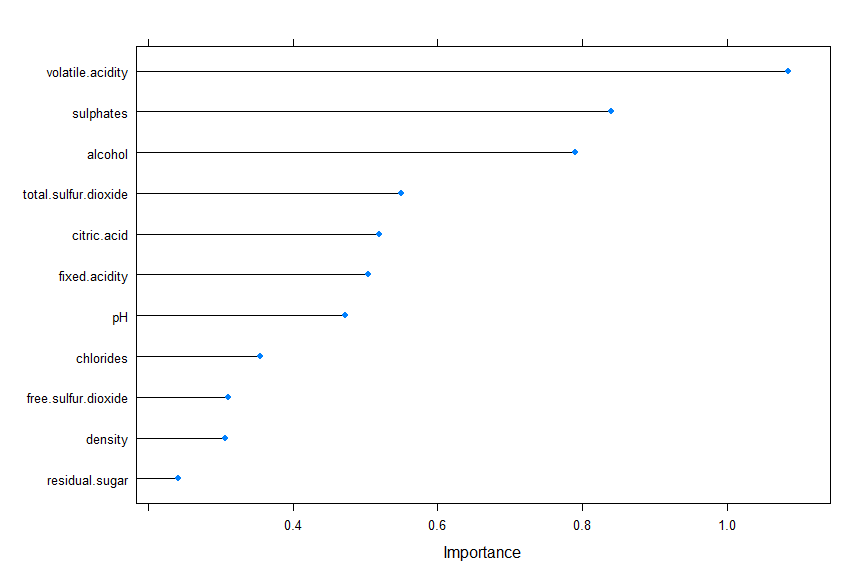
pH 0.4731

chlorides 0.3557

free.sulfur.dioxide 0.3104

density 0.3070

residual.sugar 0.2422

PLOT(IMPORTANCE)

It constructs an Learning Vector Quantization (LVQ) model. The varImp is then used to estimate the variable importance, which is printed and plotted. It shows that the volatile.acidity, sulphates and alcohol attributes are the top 3 most important attributes in the dataset and the residual.sugar attribute is the least important.

Feature Selection

X2 <- WINE # LOAD THE DATA

CONTROL <- RFECONTROL(FUNCTIONS=RFFUNCS, METHOD="CV", NUMBER=10) #DEFINE THE CONTROL USING A RANDOM FOREST SELECTION FUNCTION

RESULTS <- RFE(WINE[,1:11], WINE[,9], SIZES=C(1:11), RFECONTROL=CONTROL)

# RUN THE RFE ALGORITHM

PRINT(RESULTS)

Recursive feature selection

Outer resampling method: Cross-Validated (10 fold)

Resampling performance over subset size:

Variables RMSE Rsquared MAE RMSESD RsquaredSD MAESD Selected

1 0.003587 0.9991 0.0004052 0.003546 0.001534 0.0003628 \*

2 0.012158 0.9940 0.0047263 0.004615 0.004216 0.0006719

3 0.021189 0.9837 0.0103259 0.004807 0.006786 0.0015117

4 0.027935 0.9723 0.0150103 0.005227 0.009761 0.0014311

5 0.032330 0.9646 0.0187644 0.004357 0.009673 0.0013974

6 0.018541 0.9874 0.0089039 0.004326 0.005622 0.0009146

7 0.021939 0.9832 0.0111252 0.004088 0.006332 0.0012355

8 0.024876 0.9789 0.0132777 0.003958 0.006565 0.0010677

9 0.018059 0.9881 0.0085362 0.004442 0.005698 0.0010051

10 0.020590 0.9850 0.0101868 0.004729 0.006882 0.0012962

11 0.022838 0.9820 0.0116451 0.004372 0.006721 0.0011636

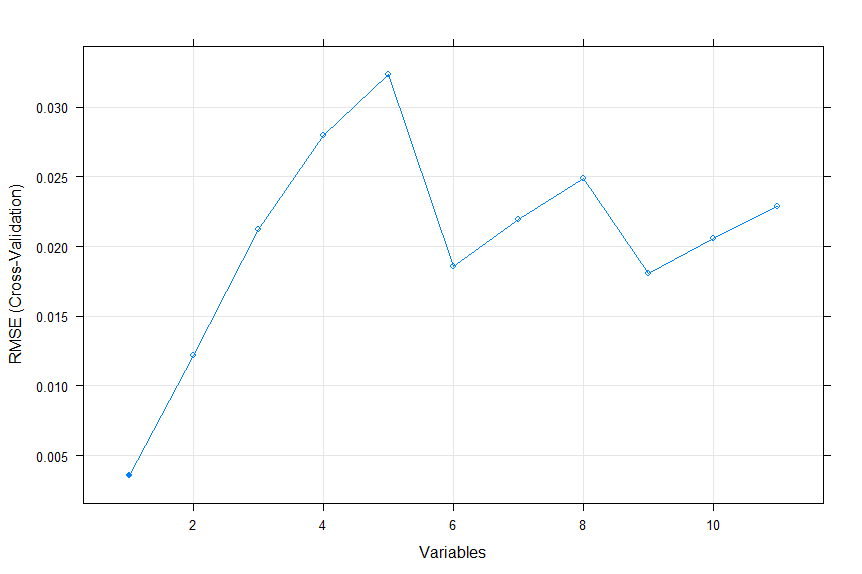
The top 1 variables (out of 1):

pH

PREDICTORS(RESULTS)

[1] "pH"

PLOT(RESULTS, TYPE=C("G", "O"))



Here, a Random Forest algorithm is chosen on each iteration to evaluate the model. The algorithm is configured to explore all possible subsets of the attributes. All 12 attributes are selected in this example, although in the plot showing the accuracy of the different attribute subset sizes, we can see that just 5 attributes gives almost comparable results.

### **Task:2 Formation of Training and Test Sets**

**Cross Validation:**

LIBRARY(CATOOLS)

LIBRARY(CARET)

WINE$QUALITY <- AS.FACTOR(WINE$QUALITY)

IND = CREATEDATAPARTITION(WINE$QUALITY, P = 2/3, LIST = FALSE)

TRAINWN<-WINE[IND,]

TESTWN<-WINE[-IND,]

#Using caret package for cross validation function and the parameters for train function in caret.

CONTROLPARAMETERS <- TRAINCONTROL(METHOD = "CV", NUMBER = 5, SAVEPREDICTIONS = TRUE, CLASSPROBS = TRUE) #traincontrol method tells which method you are trying to use. Cv means cross validation. Number = 5 means 5 fold cross validation i.e. we will create 5 partition of the Dataset. Savepredictions will save the prediction of the model. Class probabilities will see the probabilities predicted by the model.

PARAMETERGRID <- EXPAND.GRID(MTRY=C(2,3,4))

PARAMETERGRID

mtry

1 2

2 3

3 4

#For Random Forest

MODELRF <- TRAIN(QUALITY~., DATA = TRAINWN, METHOD="RF", TRCONTROL=CONTROLPARAMETERS)

MODELRF

Random Forest

1068 samples

11 predictor

6 classes: '3', '4', '5', '6', '7', '8'

No pre-processing

Resampling: Bootstrapped (25 reps)

Summary of sample sizes: 1068, 1068, 1068, 1068, 1068, 1068, ...

Resampling results across tuning parameters:

mtry Accuracy Kappa

2 0.6425749 0.4166659

6 0.6320871 0.4044275

11 0.6257362 0.3968516

Accuracy was used to select the optimal model using the largest value.

The final value used for the model was mtry = 2.

#For Naïve Bayes

MODELCVNB <- TRAIN(QUALITY~., DATA = TRAINWN, METHOD="NB", TRCONTROL=CONTROLPARAMETERS)

MODELCVNB

Naive Bayes

1068 samples

11 predictor

6 classes: '3', '4', '5', '6', '7', '8'

No pre-processing

Resampling: Bootstrapped (25 reps)

Summary of sample sizes: 1068, 1068, 1068, 1068, 1068, 1068, ...

Resampling results across tuning parameters:

usekernel Accuracy Kappa

FALSE 0.5295275 0.2923938

TRUE 0.5742928 0.3432738

Tuning parameter 'fL' was held constant at a value of 0

Tuning parameter 'adjust' was held constant at a value of 1

Accuracy was used to select the optimal model using the largest value.

The final values used for the model were fL = 0, usekernel = TRUE and adjust = 1.

#For K-NN

MODELCVKNN <- TRAIN(QUALITY~., DATA = TRAINWN, METHOD="KNN", TRCONTROL=CONTROLPARAMETERS)

MODELCVKNN

k-Nearest Neighbors

1068 samples

11 predictor

6 classes: '3', '4', '5', '6', '7', '8'

No pre-processing

Resampling: Bootstrapped (25 reps)

Summary of sample sizes: 1068, 1068, 1068, 1068, 1068, 1068, ...

Resampling results across tuning parameters:

k Accuracy Kappa

5 0.4692991 0.1611668

7 0.4781651 0.1666050

9 0.4805208 0.1620804

Accuracy was used to select the optimal model using the largest value.

The final value used for the model was k = 9.

Cross Validation has a single parameter called k that refers to the number of groups that a given data sample is to be split into. As such, the procedure is often called k-fold cross-validation. When a specific value for k is chosen, it may be used in place of k in the reference to the model, such as k=10 becoming 10-fold cross-validation.

Cross-validation is primarily used in applied machine learning to estimate the skill of a machine learning model on unseen data. That is, to use a limited sample in order to estimate how the model is expected to perform in general when used to make predictions on data not used during the training of the model.

**Leave-one-out cross-validation (Jack Knife):**

TRAIN\_CONTROL <- TRAINCONTROL(METHOD="LOOCV")

# Model for Random Forest using LOOCV

MODELLOOCVRF <- TRAIN(QUALITY~., DATA=WINE, METHOD="RF", TRCONTROL=TRAIN\_CONTROL)

PRINT(MODELLOOCVRF)

Random Forest

1599 samples

11 predictor

6 classes: '3', '4', '5', '6', '7', '8'

No pre-processing

Resampling: Leave-One-Out Cross-Validation

Summary of sample sizes: 1599, 1599, 1599, 1599, 1599, 1599, ...

Resampling results across tuning parameters:

mtry Accuracy Kappa

2 0.6721921 0.4664424

6 0.6676422 0.4616261

11 0.6595281 0.4504205

Accuracy was used to select the optimal model using the largest value.

The final value used for the model was mtry = 2.

#Model for Naïve Bayes using LOOCV

MODELLOOCVNB <- TRAIN(QUALITY~., DATA=WINE, METHOD="NB", TRCONTROL=TRAIN\_CONTROL)

PRINT(MODELLOOCVNB)

Naive Bayes

1599 samples

11 predictor

6 classes: '3', '4', '5', '6', '7', '8'

No pre-processing

Resampling: Leave-One-Out Cross-Validation

Summary of sample sizes: 1599, 1599, 1599, 1599, 1599, 1599, ...

Resampling results across tuning parameters:

usekernel Accuracy Kappa

FALSE 0.5279596 0.2862220

TRUE 0.5603483 0.3235109

Tuning parameter 'fL' was held constant at a value of 0

Tuning parameter 'adjust' was held constant at a value of 1

Accuracy was used to select the optimal model using the largest value.

The final values used for the model were fL = 0, usekernel = TRUE and adjust = 1.

#Model for K-NN using LOOCV

MODELLOOCVKNN <- TRAIN(QUALITY~., DATA=WINE, METHOD="KNN", TRCONTROL=TRAIN\_CONTROL)

PRINT(MODELLOOCVKNN)

k-Nearest Neighbors

1599 samples

11 predictor

6 classes: '3', '4', '5', '6', '7', '8'

No pre-processing

Resampling: Leave-One-Out Cross-Validation

Summary of sample sizes: 1598, 1598, 1598, 1598, 1598, 1598, ...

Resampling results across tuning parameters:

k Accuracy Kappa

5 0.5071920 0.2096076

7 0.4990619 0.1883995

9 0.4978111 0.1787402

Accuracy was used to select the optimal model using the largest value.

The final value used for the model was k = 5.

LOOCV uses a single observation from the original sample as the validation data, and the remaining observations as the training data. This is repeated such that each observation in the sample is used once as the validation data. This is the same as a K-fold cross-validation with K being equal to the number of observations in the original sample. Leave-one-out cross-validation is usually very expensive from a computational point of view because of the large number of times the training process is repeated.

#### **Task:3 Build Train and Test a Random Forest type Classifier**

**Building of Random Forest type classifier in R**

WINE\_SET\_SIZE= FLOOR(NROW(WINE)\*0.80)

INDEX <- SAMPLE(1:NROW(WINE), SIZE = WINE\_SET\_SIZE)

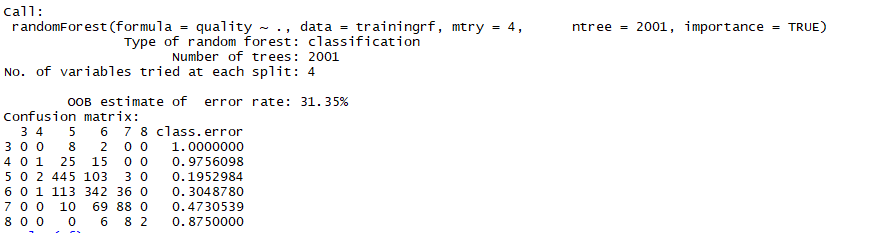
TRAININGRF <- WINE[INDEX,]

TESTINGRF <- WINE[INDEX,]

LIBRARY(RANDOMFOREST) #Used for both classification and regression

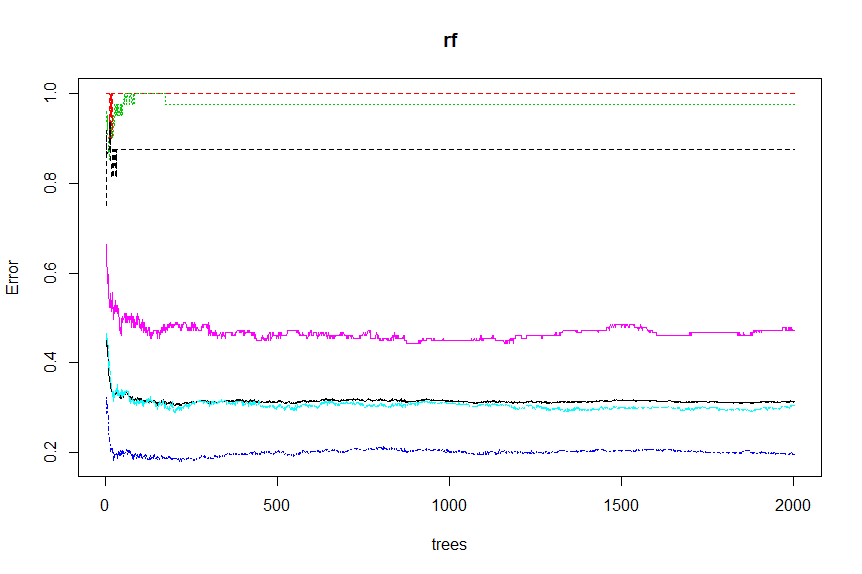
RF <- RANDOMFOREST(QUALITY ~ ., DATA = TRAININGRF, MTRY=4, NTREE=2001, IMPORTANCE=TRUE)

RF



The error rate is 31.35%. Therefore, accuracy is 68.65%. Always check your domain when you are sharing your stuff because if you are doing something that is life or a death maybe 69% accuracy might not be so good but if you are looking at where you are going to distribute the bottle's to for the wine and who is going to drink it, 69% accuracy is pretty good for that.

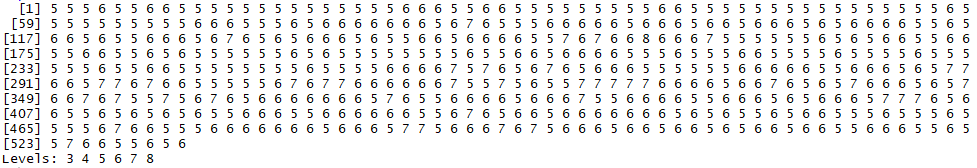
PLOT(RF)



#Checking the prediction of the test dataset

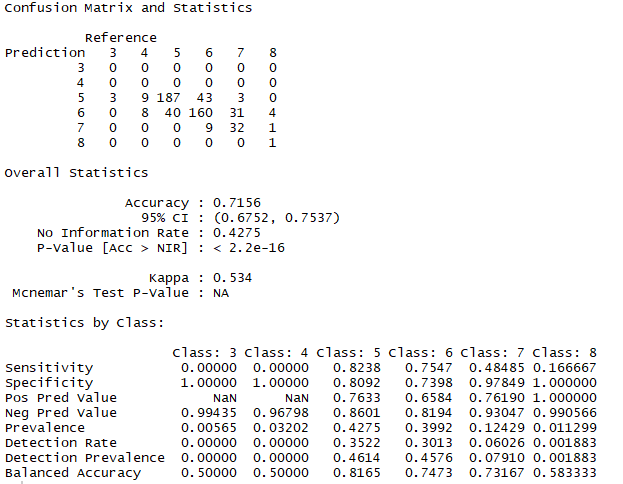
PREDICTIONS <- PREDICT(MODELRF, TESTWN[,-12])

PREDICTIONS



#Checking the confusion matrix

CONFUSIONMATRIX(PREDICTIONS, TESTWN$QUALITY)



**Training of Random Forest type Classifier in R**

WINE\_SET\_SIZE= FLOOR(NROW(WINE)\*0.80)

INDEX <- SAMPLE(1:NROW(WINE), SIZE = WINE\_SET\_SIZE)

TRAININGRF <- WINE[INDEX,]

TESTINGRF <- WINE[INDEX,]

**Testing of Random Forest type Classifier in R**

MODELRF <- TRAIN(QUALITY~., DATA = TRAINWN, METHOD="RF", PREPROCESS = C('CENTER','SCALE'))

MODELRF

Random Forest

1068 samples

11 predictor

6 classes: '3', '4', '5', '6', '7', '8'

Pre-processing: centered (11), scaled (11)

Resampling: Bootstrapped (25 reps)

Summary of sample sizes: 1068, 1068, 1068, 1068, 1068, 1068, ...

Resampling results across tuning parameters:

mtry Accuracy Kappa

2 0.6247855 0.3874872

6 0.6186487 0.3820486

11 0.6123106 0.3749546

Accuracy was used to select the optimal model using the largest value.

The final value used for the model was mtry = 2.

Random Forest uses multiple decision trees which are built on separate on separate set of examples drawn at random from the same dataset. In each tree, we can use a subset of all the features we have. By using more decision trees and averaging the result, the variance of the model can be greatly lowered. For Random Forest, there are two main parameters to be considered: number of trees and number of features they select at each decision point. Theoretically, accuracy will increase with more trees making decision.

##### **Task:4 Build Train and Test a Naive Bayes type Classifier**

**Building of Naive Bayes type classifier in R**

LIBRARY(CARET)

LIBRARY(KLAR)

# DEFINE AN 80%/20% TRAIN/TEST SPLIT OF THE DATASET

SPLIT=0.80

TRAININDEX <- CREATEDATAPARTITION(WINE$QUALITY, P=SPLIT, LIST=FALSE)

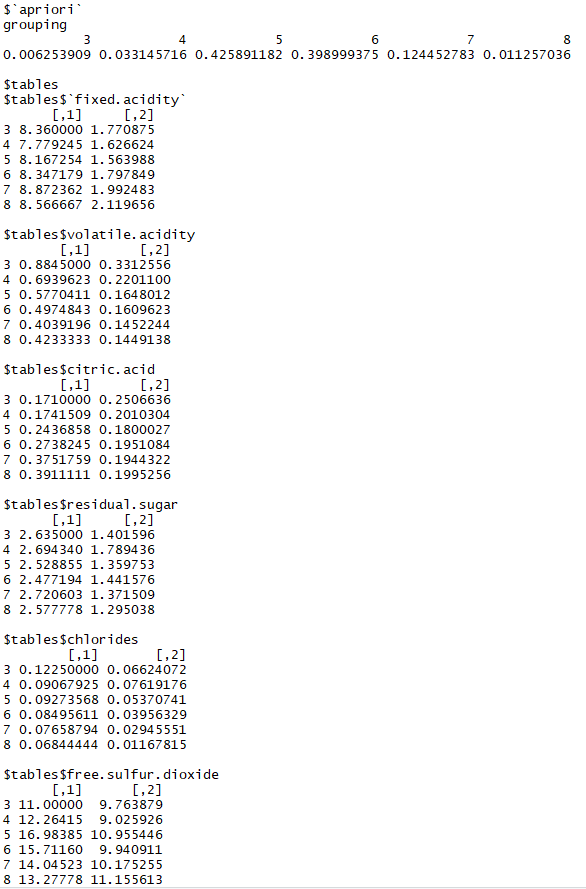
DATA\_TRAIN <- WINE[ TRAININDEX,]

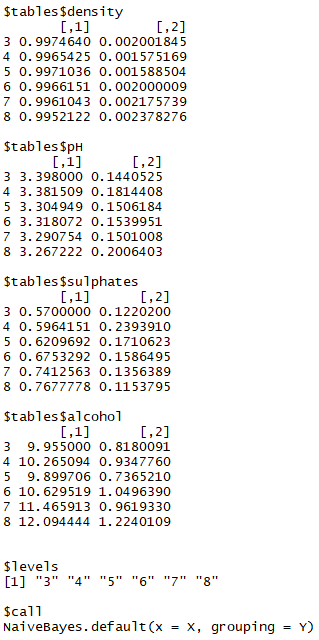
DATA\_TEST <- WINE[-TRAININDEX,]

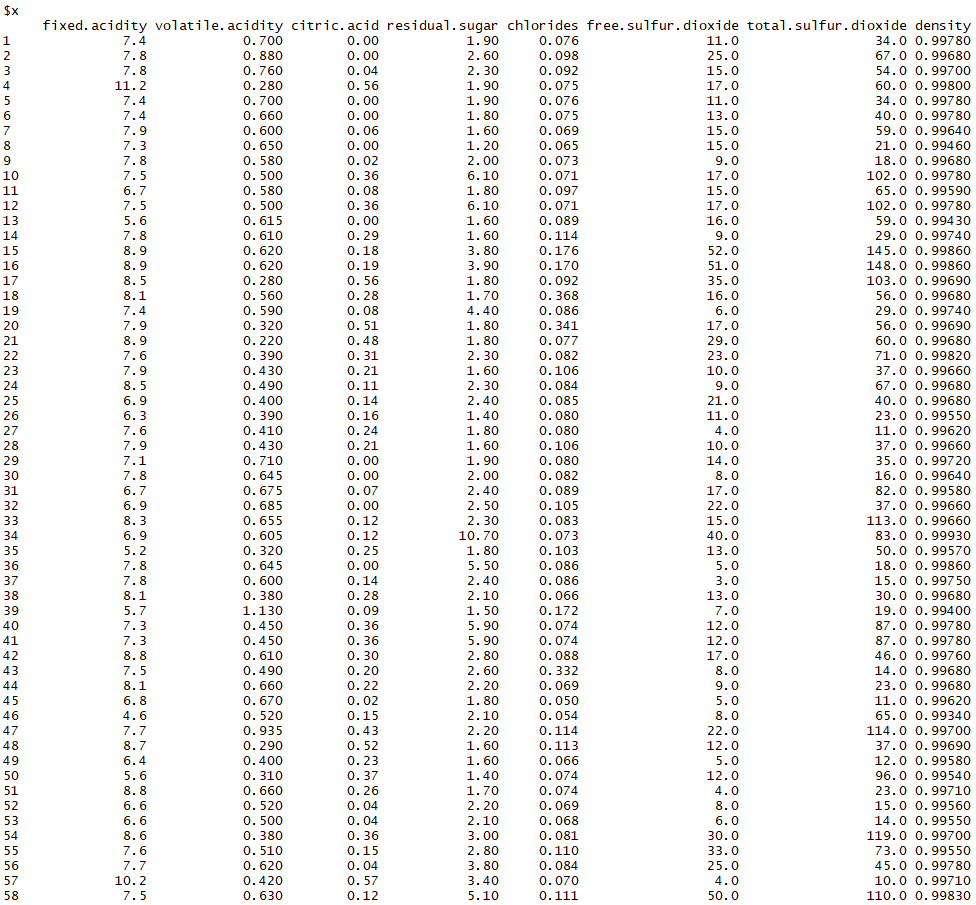
# Train a naive bayes model

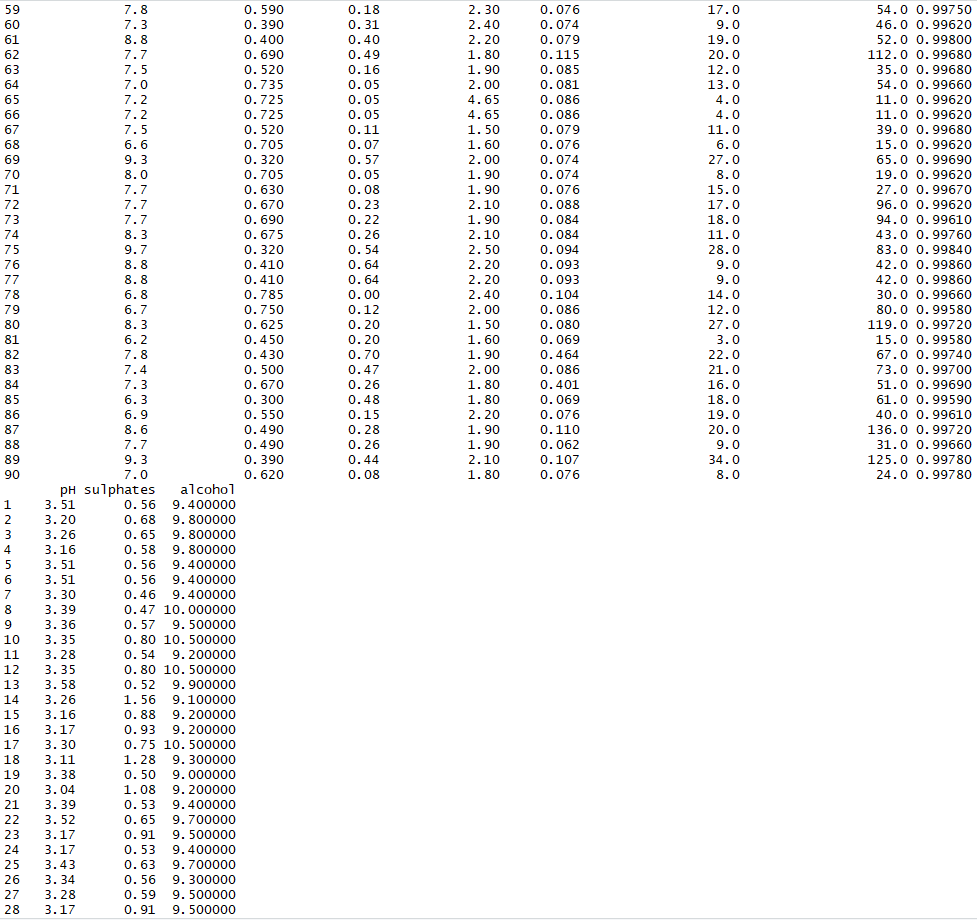
MODEL= NAIVEBAYES(QUALITY~., DATA = WINE)

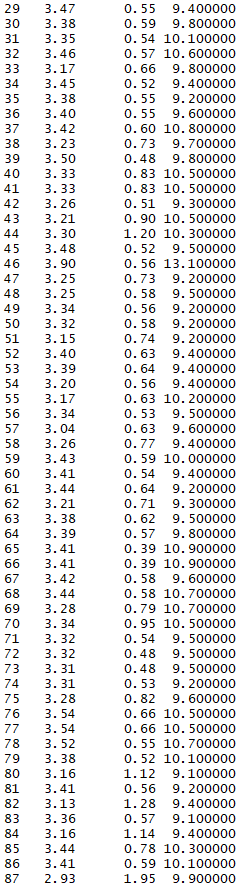
MODEL

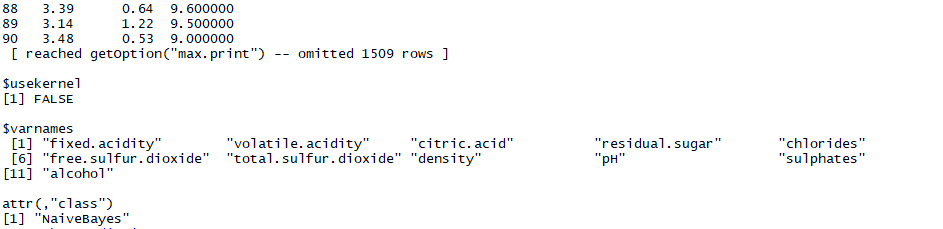












# MAKE PREDICTIONS

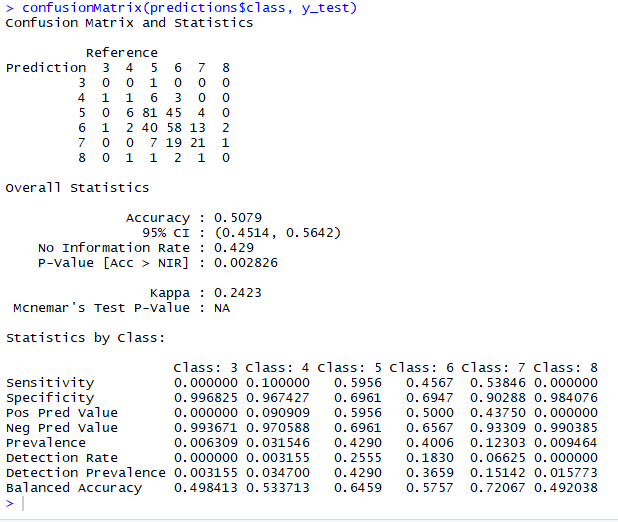
X\_TRAIN <- DATA\_TEST[,1:11]

Y\_TEST <- DATA\_TEST[,12]

PREDICTIONS <- PREDICT(MODEL, X\_TRAIN)

# Summarize results using Confusion Matrix

CONFUSIONMATRIX(PREDICTIONS$CLASS, Y\_TEST)



**Training of Naive Bayes type classifier in R**

SPLIT=0.80

TRAININDEX <- CREATEDATAPARTITION(WINE$QUALITY, P=SPLIT, LIST=FALSE)

DATA\_TRAIN <- WINE[ TRAININDEX,]

DATA\_TEST <- WINE[-TRAININDEX,]

**Testing of Naive Bayes type classifier in R**

# Make predictions

X\_TRAIN <- DATA\_TEST[,1:11]

Y\_TEST <- DATA\_TEST[,12]

PREDICTIONSNB <- PREDICT(MODEL, X\_TRAIN)

Naive Bayes classifier is a family of probabilistic classifiers based on applying Bayes theorem with the assumption that the features are independent. This machine learning approach predicts the quality. It classifies the data, builds a model based on the training set and values in a classifying attribute and uses it to classify new data.

###### **Task:5 Build Train and Test a K-NN type Classifier**

**Building of K-NN type classifier in R**

#Dividing dataset into training and testing

SET.SEED(3033)

INTRAIN <- CREATEDATAPARTITION(Y = WINE$QUALITY, P= 0.8, LIST = FALSE)

TRAININGKNN <- WINE[INTRAIN,]

TESTINGKNN <- WINE[-INTRAIN,]

#Exploratory Analysis

DIM(TRAININGKNN); DIM(TESTINGKNN);

[1] 1282 12

[1] 317 12

#Training the KNN model on our training data

TRCTRL <- TRAINCONTROL(METHOD = "REPEATEDCV", NUMBER = 10, REPEATS = 3)

SET.SEED(3333)

KNN\_FIT <- TRAIN(QUALITY~., DATA = TRAININGKNN, METHOD = "KNN",

TRCONTROL=TRCTRL,

PREPROCESS = C("CENTER", "SCALE"),

TUNELENGTH = 10)

KNN\_FIT

k-Nearest Neighbors

1282 samples

11 predictor

6 classes: '3', '4', '5', '6', '7', '8'

Pre-processing: centered (11), scaled (11)

Resampling: Cross-Validated (10 fold, repeated 3 times)

Summary of sample sizes: 1153, 1154, 1153, 1154, 1153, 1156, ...

Resampling results across tuning parameters:

k Accuracy Kappa

5 0.5548869 0.2915940

7 0.5550785 0.2856353

9 0.5512250 0.2785855

11 0.5559026 0.2831601

13 0.5654667 0.2971720

15 0.5652411 0.2928706

17 0.5626678 0.2874374

19 0.5631640 0.2863923

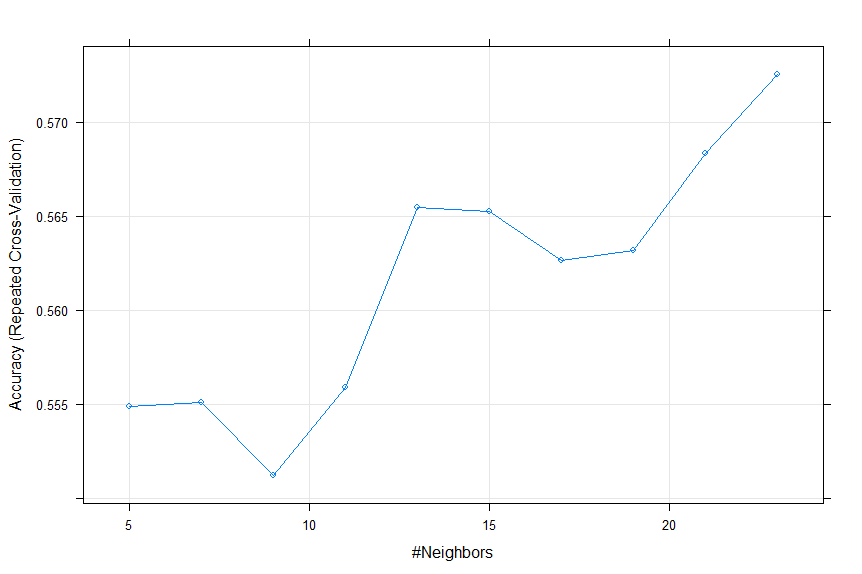
21 0.5683682 0.2922788

23 0.5725331 0.2992087

Accuracy was used to select the optimal model using the largest value.

The final value used for the model was k = 23.

PLOT(KNN\_FIT)



#Put model on to the test dataset

TEST\_PREDKNN <- PREDICT(KNN\_FIT, TESTINGKNN)

TEST\_PREDKNN

[1] 5 5 5 5 5 5 5 5 5 5 5 6 6 5 5 6 5 5 5 5 6 5 6 5 5 5 5 5 5 5 5 5 5 5 6 5 6 5 6 6 5 5 5 5 6 5 6 5 5 5 6 7 5 7 6 5 5 7

[59] 6 6 6 5 5 6 6 6 5 6 7 5 5 5 7 6 6 6 6 6 6 6 6 6 7 6 7 6 6 6 5 5 6 6 6 6 7 6 6 5 6 6 6 5 5 6 6 7 6 5 6 6 6 5 6 6 5 5

[117] 5 5 5 7 5 6 6 5 5 6 5 6 6 5 6 6 5 6 5 5 6 5 5 6 5 6 5 6 6 5 5 6 6 5 5 5 6 5 5 5 5 5 5 5 5 5 6 5 6 7 6 6 6 7 6 5 6 6

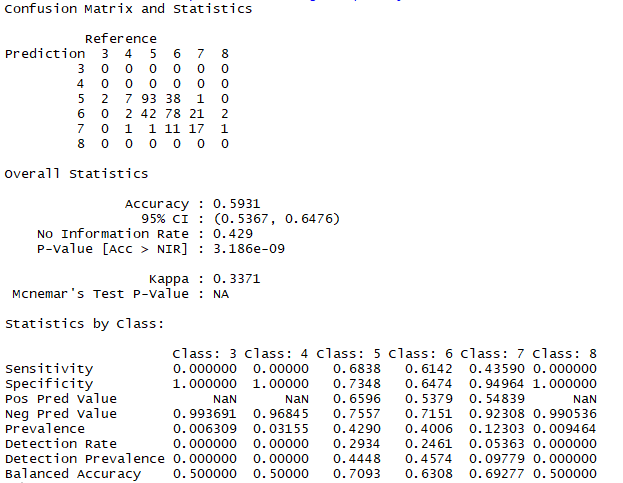
[175] 6 6 5 6 6 5 6 6 5 7 6 6 7 7 7 6 5 6 5 5 5 6 6 7 6 5 7 6 7 6 5 6 5 5 6 6 6 6 5 5 7 6 7 7 5 7 6 7 6 6 6 5 5 6 7 5 6 5

[233] 7 6 7 5 7 5 5 6 6 5 6 6 5 6 6 6 6 5 6 5 6 7 5 6 6 5 6 7 6 6 6 5 5 6 6 5 6 6 5 6 5 5 5 5 5 5 5 5 6 5 6 6 5 6 5 6 5 6

[291] 5 5 6 6 5 5 6 5 5 6 5 6 6 6 6 6 5 6 5 6 6 6 7 6 5 6 6

Levels: 3 4 5 6 7 8

#Plot the confusion matrix and check the classes

CONFUSIONMATRIX(TEST\_PREDKNN, TESTINGKNN$QUALITY)

**Training of K-NN type classifier in R**

#Training the KNN model on our training data

TRCTRL <- TRAINCONTROL(METHOD = "REPEATEDCV", NUMBER = 10, REPEATS = 3)

SET.SEED(3333)

KNN\_FIT <- TRAIN(QUALITY~., DATA = TRAININGKNN, METHOD = "KNN",

TRCONTROL=TRCTRL,

PREPROCESS = C("CENTER", "SCALE"),

TUNELENGTH = 10)

KNN\_FIT

k-Nearest Neighbors

1282 samples

11 predictor

6 classes: '3', '4', '5', '6', '7', '8'

Pre-processing: centered (11), scaled (11)

Resampling: Cross-Validated (10 fold, repeated 3 times)

Summary of sample sizes: 1153, 1154, 1153, 1154, 1153, 1156, ...

Resampling results across tuning parameters:

k Accuracy Kappa

5 0.5548869 0.2915940

7 0.5550785 0.2856353

9 0.5512250 0.2785855

11 0.5559026 0.2831601

13 0.5654667 0.2971720

15 0.5652411 0.2928706

17 0.5626678 0.2874374

19 0.5631640 0.2863923

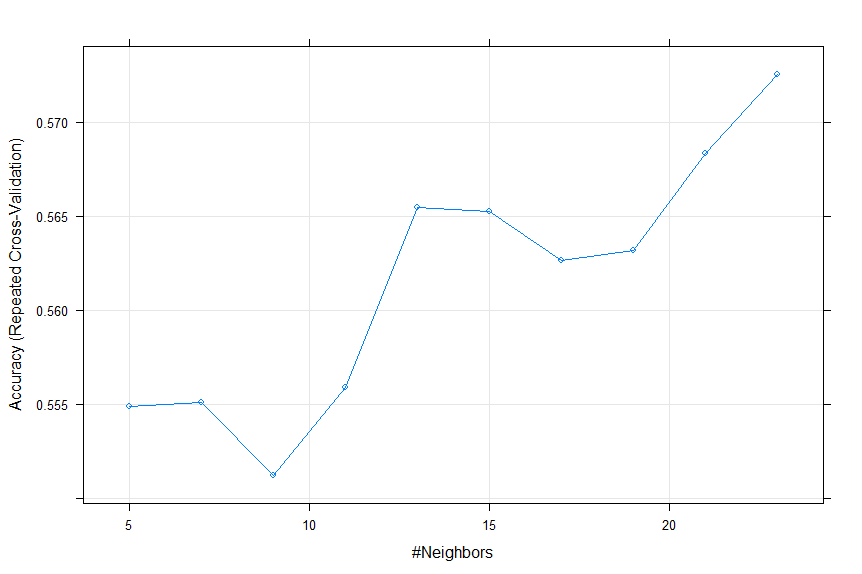
21 0.5683682 0.2922788

23 0.5725331 0.2992087

Accuracy was used to select the optimal model using the largest value.

The final value used for the model was k = 23.

PLOT(KNN\_FIT)



**Testing of K-NN type classifier in R**

#Put model on to the test dataset

TEST\_PREDKNN <- PREDICT(KNN\_FIT, TESTINGKNN)

TEST\_PREDKNN

[1] 5 5 5 5 5 5 5 5 5 5 5 6 6 5 5 6 5 5 5 5 6 5 6 5 5 5 5 5 5 5 5 5 5 5 6 5 6 5 6 6 5 5 5 5 6 5 6 5 5 5 6 7 5 7 6 5 5 7

[59] 6 6 6 5 5 6 6 6 5 6 7 5 5 5 7 6 6 6 6 6 6 6 6 6 7 6 7 6 6 6 5 5 6 6 6 6 7 6 6 5 6 6 6 5 5 6 6 7 6 5 6 6 6 5 6 6 5 5

[117] 5 5 5 7 5 6 6 5 5 6 5 6 6 5 6 6 5 6 5 5 6 5 5 6 5 6 5 6 6 5 5 6 6 5 5 5 6 5 5 5 5 5 5 5 5 5 6 5 6 7 6 6 6 7 6 5 6 6

[175] 6 6 5 6 6 5 6 6 5 7 6 6 7 7 7 6 5 6 5 5 5 6 6 7 6 5 7 6 7 6 5 6 5 5 6 6 6 6 5 5 7 6 7 7 5 7 6 7 6 6 6 5 5 6 7 5 6 5

[233] 7 6 7 5 7 5 5 6 6 5 6 6 5 6 6 6 6 5 6 5 6 7 5 6 6 5 6 7 6 6 6 5 5 6 6 5 6 6 5 6 5 5 5 5 5 5 5 5 6 5 6 6 5 6 5 6 5 6

[291] 5 5 6 6 5 5 6 5 5 6 5 6 6 6 6 6 5 6 5 6 6 6 7 6 5 6 6

Levels: 3 4 5 6 7 8

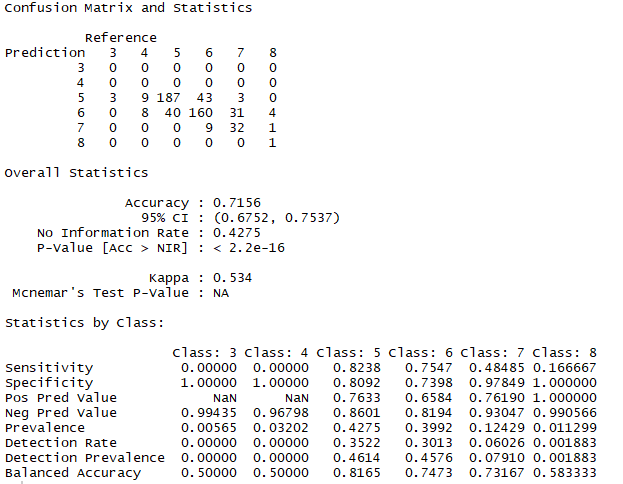
The *k*‐Nearest Neighbours algorithm is a non‐parametric method used for classification and regression. In both cases, the input consists of the *k* closest training examples in the feature space. The output depends on whether *k*‐NN is used for classification or regression. In this approach we are using K‐NN for classification. Here the output is a quality. An object is classified by a majority vote of its neighbours, with the object being assigned to the quality most common among its *k* nearest neighbours.

**Task:6 Measure Performance**

**Confusion Matrix Estimation:**

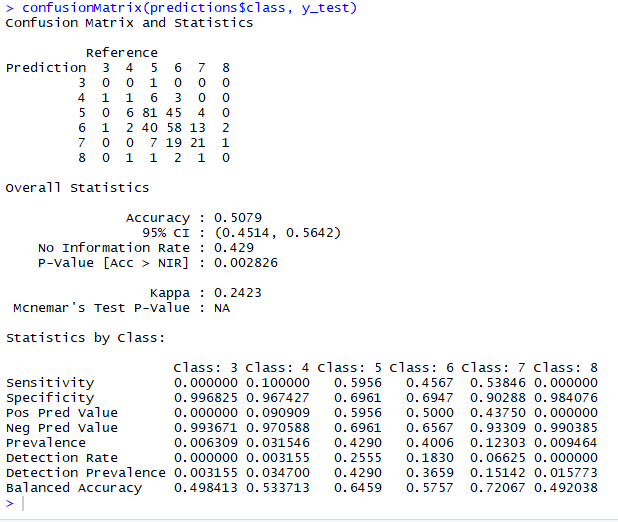
For Random Forest:

CONFUSIONMATRIX(PREDICTIONS, TESTWN$QUALITY)

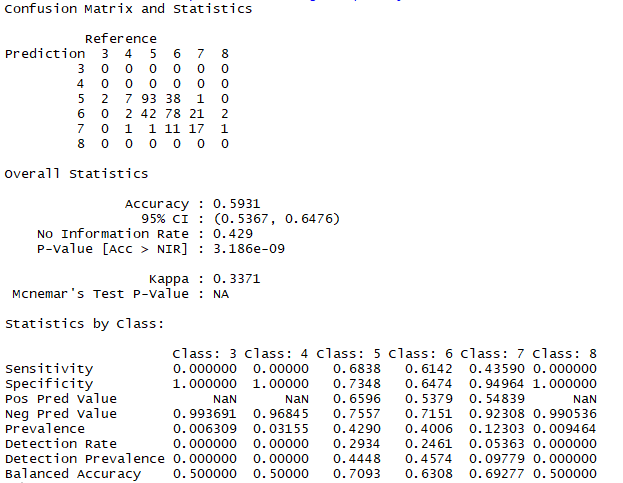


For Naïve Bayes:

CONFUSIONMATRIX(PREDICTIONS$CLASS, Y\_TEST)



For K-NN:

CONFUSIONMATRIX(TEST\_PREDKNN, TESTINGKNN$QUALITY)

Confusion Matrix in every model, compares the predicted values with the original values of the training dataset and then gives accuracy of the model, matrix of predicted and the original dataset. Matrix is checked by looking at the diagonal values to see the correct values of the diagonal and the other values are incorrect.

**Training Time:**

For Random Forest:

system.time(modelrf <- train(quality~., data = trainWN, method="rf", preProcess = c('center','scale')))

user system elapsed

59.63 0.95 60.63

For Naïve Bayes:

system.time(modelnb= NaiveBayes(quality~., data = wine))

user system elapsed

3.09 0.02 4.65

For K-NN:

system.time(knn\_fit <- train(quality~., data = trainingknn, method = "knn", trControl=trctrl, preProcess = c("center", "scale"), tuneLength = 10))

user system elapsed

2.98 0.01 3.01

**Testing Time:**

For Random Forest:

system.time(predictions <- predict(modelrf, testWN[,-12]))

user system elapsed

0.02 0.00 0.01

For Naïve Bayes

system.time(predictionsNB <- predict(modelnb, x\_train))

user system elapsed

0.07 0.00 0.07

For K-NN:

system.file(test\_predknn <- predict(knn\_fit, testingknn))

user system elapsed

0.01 0.00 0.02

The following table shows **Accuracy, Precision, Recall** for various Classification models:

|  |  |  |  |
| --- | --- | --- | --- |
| **Algorithm** | **Accuracy** | **Precision** | **Recall** |
| Random Forest | 71.56 | 0.966 | 0.943 |
| Naïve Bayes | 50.79 | 0.638 | 0.612 |
| K-NN (For k=23) | 59.31 | 0.602 | 0.541 |

**Precision vs Recall**:  
> record\_1  
An object of class "quality"  
Slot "x.name":  
[1] "Recall"  
  
Slot "y.name":  
[1] "Precision"  
  
Slot "alpha.name":  
[1] "Cutoff"  
  
Slot "x.values":  
[[1]]  
[1] 0.0000000 0.9666666 1.0000000  
  
  
Slot "y.values":  
[[1]]  
[1]       NaN 0.8500000 0.3157895  
  
  
Slot "alpha.values":  
[[1]]  
[1] Inf   2   1

> record\_2  
An object of class "performance"  
Slot "x.name":  
[1] "Recall"  
  
Slot "y.name":  
[1] "Precision"  
  
Slot "alpha.name":  
[1] "Cutoff"  
  
Slot "x.values":  
[[1]]  
[1] 0.0000000 0.638889 1.0000000  
  
  
Slot "y.values":  
[[1]]  
[1]       NaN 0.9444444 0.3157895  
  
  
Slot "alpha.values":  
[[1]]  
[1] Inf   2   1  
  
  
> record\_3  
An object of class "performance"  
Slot "x.name":  
[1] "Recall"  
  
Slot "y.name":  
[1] "Precision"  
  
Slot "alpha.name":  
[1] "Cutoff"  
  
Slot "x.values":  
[[1]]  
[1] 0.0000000 0.602221 1.0000000  
  
  
Slot "y.values":  
[[1]]  
[1]       NaN 0.9444444 0.3157895  
  
  
Slot "alpha.values":  
[[1]]  
[1] Inf   2   1

**Roc values:**  
  
> roc1  
An object of class "quality"  
Slot "x.name":  
[1] "False positive rate"  
  
Slot "y.name":  
[1] "True positive rate"  
  
Slot "alpha.name":  
[1] "Cutoff"  
  
Slot "x.values":  
[[1]]  
[1] 0.00000000 0.7156357 1.00000000  
  
  
Slot "y.values":  
[[1]]  
[1] 0.0000000 0.71596314 1.0000000  
  
  
Slot "alpha.values":  
[[1]]  
[1] Inf   2   1  
  
  
> roc2  
An object of class "quality"  
Slot "x.name":  
[1] "False positive rate"  
  
Slot "y.name":  
[1] "True positive rate"  
  
Slot "alpha.name":  
[1] "Cutoff"  
  
Slot "x.values":  
[[1]]  
[1] 0.00000000 0.5079487 1.00000000  
  
  
Slot "y.values":  
[[1]]  
[1] 0.0000000 0.5079487 1.0000000  
  
  
Slot "alpha.values":  
[[1]]  
[1] Inf   2   1

> roc3  
An object of class "performance"  
Slot "x.name":  
[1] "False positive rate"  
  
Slot "y.name":  
[1] "True positive rate"  
  
Slot "alpha.name":  
[1] "Cutoff"  
  
Slot "x.values":  
[[1]]  
[1] 0.00000000 0.0593153 1.00000000  
  
  
Slot "y.values":  
[[1]]  
[1] 0.0000000 0.9666666 1.0000000  
  
  
Slot "alpha.values":  
[[1]]  
[1] Inf   2   1  
  
  
**RAUC Values:**  
> rauc\_value\_1  
[1] 0.716893  
> rauc\_value\_2  
[1] 0.556981  
> rauc\_value\_3  
[1] 0.603587

**Appendix**

wine <- read.csv("H:/MASTERS/Data Mining and Machine Learning/Assessment 2018-19/CW2/winequality-red.csv")

str(wine) #Structure of dataset

dim(wine) #Checking dimension of dataset

summary(wine) #Checking summary of the dataset

apply(wine,2,function(x) length(unique(x))) #Checking the number of unique values

table(wine$quality)

wine$quality <- as.integer(wine$quality)

par(mfrow = c(4,3))

for (i in c(1:11)) {

plot(wine[, i], jitter(wine[, "quality"]), xlab = names(wine)[i],

ylab = "quality", col = "firebrick", cex = 0.8, cex.lab = 1.3)

abline(lm(wine[, "quality"] ~ wine[ ,i]), lty = 2, lwd = 2)

}

par(mfrow = c(1, 1))

par(mfrow = c(1,1))

cor.wine <- cor(wine)

corrplot(cor.wine, method = 'number')

#####FEATURE SELECTION

# ensure results are repeatable

set.seed(7)

# load the library

library(mlbench)

library(caret)

# load the dataset

df1<-wine[,-12]

quality <- wine$quality

# prepare training scheme

control <- trainControl(method="repeatedcv", number=10, repeats=3)

# train the model

model <- train(quality~., data=wine, method = "treebag",preProcess="scale")

# estimate variable importance

importance <- varImp(model, scale=FALSE)

# summarize importance

print(importance)

# plot importance

plot(importance)

# ensure the results are repeatable

set.seed(7)

# load the library

library(mlbench)

library(caret)

# load the data

x2 <- wine

# define the control using a random forest selection function

control <- rfeControl(functions=rfFuncs, method="cv", number=10)

# run the RFE algorithm

results <- rfe(wine[,1:11], wine[,9], sizes=c(1:11), rfeControl=control)

# summarize the results

print(results)

# list the chosen features

predictors(results)

# plot the results

plot(results, type=c("g", "o"))

#Creating train and test dataset. Target variable is Quality

library(caTools)

library(caret)

wine$quality <- as.factor(wine$quality)

ind = createDataPartition(wine$quality, p = 2/3, list = FALSE)

trainWN<-wine[ind,]

testWN<-wine[-ind,]

#Using caret package for crossvalidation function and the parameters for train function in caret

ControlParameters <- trainControl(method = "cv", number = 5, savePredictions = TRUE, classProbs = TRUE) #trainControl method tells which method you are trying

#to use. CV means cross validation. number = 5 means 5 fold cross validation i.e. we will create 5 partition of the

#dataset. savePredictions will save the prediction of the model. class Probabilities will see the probabilities predicted

#by the model.

parameterGrid <- expand.grid(mtry=c(2,3,4))

parameterGrid

modelrf <- train(quality~., data = trainWN, method="rf", trControl=ControlParameters)

modelrf

modelcvnb <- train(quality~., data = trainWN, method="nb", trControl=ControlParameters)

modelcvnb

modelcvknn <- train(quality~., data = trainWN, method="knn", trControl=ControlParameters)

modelcvknn

#####LOOCV

train\_control <- trainControl(method="LOOCV")

# train the model

modelloocvrf <- train(quality~., data=wine, method="rf", trControl=train\_control)

# summarize results

print(modelloocvrf)

modelloocvnb <- train(quality~., data=wine, method="nb", trControl=train\_control)

print(modelloocvnb)

modelloocvknn <- train(quality~., data=wine, method="knn", trControl=train\_control)

print(modelloocvknn)

####RANDOM FOREST#################

wine\_set\_size= floor(nrow(wine)\*0.80)

index <- sample(1:nrow(wine), size = wine\_set\_size)

trainingrf <- wine[index,]

testingrf <- wine[index,]

library(randomForest) #Used for both classification and regression

rf <- randomForest(quality ~ ., data = trainingrf, mtry=4, ntree=2001, importance=TRUE)

rf #The error rate is 31.35%. Therefore, accuracy is 68.65%. Always check your

#domain when you are sharing your stuff because if you are doing something that is life or a death maybe 69% accuracy

#might not be so good but if you are looking at where you are going to distribute the bottle's to for the wine

#and who is going to drink it, 69% accuracy is pretty good for that. Probably better than having no accuracy and not

#even knowing what wines going out to who

plot(rf)

modelrf <- train(quality~., data = trainWN, method="rf", preProcess = c('center','scale'))

modelrf

#Checking the prediction of the test dataset

predictions <- predict(modelrf, testWN[,-12])

predictions

#Checking the confusion Matrix

confusionMatrix(predictions, testWN$quality)

#######NAIVE BAYES

library(caret)

library(klaR)

# define an 80%/20% train/test split of the dataset

split=0.80

trainIndex <- createDataPartition(wine$quality, p=split, list=FALSE)

data\_train <- wine[ trainIndex,]

data\_test <- wine[-trainIndex,]

# train a naive bayes model

modelnb= NaiveBayes(quality~., data = wine)

modelnb

# make predictions

x\_train <- data\_test[,1:11]

y\_test <- data\_test[,12]

predictionsNB <- predict(modelnb, x\_train)

# summarize results

confusionMatrix(predictions$class, y\_test)

#Divide the dataset into training and testing

set.seed(3033)

intrain <- createDataPartition(y = wine$quality, p= 0.8, list = FALSE)

trainingknn <- wine[intrain,]

testingknn <- wine[-intrain,]

#Exploratory Analysis

dim(trainingknn); dim(testingknn);

#Tarin the KNN model on our training data

trctrl <- trainControl(method = "repeatedcv", number = 10, repeats = 3)

set.seed(3333)

knn\_fit <- train(quality~., data = trainingknn, method = "knn",

trControl=trctrl,

preProcess = c("center", "scale"),

tuneLength = 10)

knn\_fit

plot(knn\_fit)

#Put your model on to the test dataset

test\_predknn <- predict(knn\_fit, testingknn)

test\_predknn

#Plot the confusion matrix and check the classes

confusionMatrix(test\_predknn, testingknn$quality)

#Time elapsed for KNN

system.time(knn\_fit <- train(quality~., data = trainingknn, method = "knn",

trControl=trctrl,

preProcess = c("center", "scale"),

tuneLength = 10))

system.file(test\_predknn <- predict(knn\_fit, testingknn))

#Time elapsed for RF

system.time(modelrf <- train(quality~., data = trainWN, method="rf", preProcess = c('center','scale')))

system.time(predictions <- predict(modelrf, testWN[,-12]))

#Time elapsed for NB

system.time(modelnb= NaiveBayes(quality~., data = wine))

system.time(predictionsNB <- predict(modelnb, x\_train))