Naive Bayes Classifier and Discriminant Analysis

Task 1: Application of a Classifier

The **Mushroom dataset**, available from the UCI Data Repository https://archive.ics.uci.edu/ml/datasets/Mushroom, contains 8124 observations describing mushrooms belonging to classes edible or potentially poisonous (first variable encoded 'e' or 'p', respectively). There are 22 categorical predictors (variables 2 to 23), one of them with missing values ('?'). Complete the following five instructions to develop and justify a supervised classifier using RStudio:

Question 1 Import the Mushroom Data into RStudio.

```
> mushrooms <- read.table("https://archive.ics.uci.edu/ml/machine-learning-
                          databases/mushroom/agaricus-lepiota.data", header = FALSE, sep =
 ",", dec = ".", na.strings = c("?")) # read the data in the csv file and load into R
                                      environment as a dataframe mushrooms

# Arguments of the function read.csv() are:
# a) file: name of the file with its remote access(path);
# b) header = FALSE: a logical (FALSE or TRUE) indicating if the file contains the names of
                     the variables on its first line;
# The csv file does  not contains names of variables;
# c) sep = ",": the field separator used in the file is comma, (",");
# d) dec = ".": the character used for the decimal point;
# e) na.strings = c("?"): the symbol "?" given to missing data and converted as NA by
                          default;

> summary(mushrooms) # result summary of the dataframe
       V1       V2       V3             V4       V5             V6       V7
    e:4208   b: 452   f:2320   n      :2284   f:4748   n      :3528   a: 210
    p:3916   c:   4   g:   4   g      :1840   t:3376   f      :2160   f:7914
             f:3152   s:2556   e      :1500            s      : 576
             k: 828   y:3244   y      :1072            y      : 576
             s:  32            w      :1040            a      : 400
             x:3656            b      : 168            l      : 400
                               (Other): 220          (Other): 484
       V8       V9            V10       V11        V12       V13      V14
     c:6812   b:5612   b      :1728   e:3516   b   :3776   f: 552   f: 600
     w:1312   n:2512   p      :1492   t:4608   c   : 556   k:2372   k:2304
                       w      :1202            e   :1120   s:5176   s:4936
                       n      :1048            r   : 192   y:  24   y: 284
                       g      : 752            NA's:2480
                       h      : 732
                    (Other):1170
       V15            V16       V17      V18      V19      V20
   w      :4464   w      :4384   p:8124   n:  96   n:  36   e:2776
   p      :1872   p      :1872            o:  96   o:7488   f:  48
   g      : 576   g      : 576            w:7924   t: 600   l:1296
   n      : 448   n      : 512            y:   8            n:  36
   b      : 432   b      : 432                              p:3968
   o      : 192   o      : 192
   (Other): 140   (Other): 156
       V21       V22      V23
   w      :2388   a: 384   d:3148
   n      :1968   c: 340   g:2148
   k      :1872   n: 400   l: 832
   h      :1632   s:1248   m: 292
   r      :  72   v:4040   p:1144
   b      :  48   y:1712   u: 368
  (Other): 144            w: 192
> dim(mushrooms) # get the dimensions of the dataframe
[1] 8124   23
```

```
sapply(mushrooms,class) # user-friendly version and wrapper of lapply function but returns a
                        vector of column classes
      V1       V2       V3       V4       V5       V6       V7       V8       V9      V10
"factor" "factor" "factor" "factor" "factor" "factor" "factor" "factor" "factor" "factor"
     V11      V12      V13      V14      V15      V16      V17      V18      V19      V20
"factor" "factor" "factor" "factor" "factor" "factor" "factor" "factor" "factor" "factor"
     V21      V22      V23
"factor" "factor" "factor"
# Rename the column names assigned by default as V1, V2...V23 into names shown below:
colnames(mushrooms) <- c("edibility", "cap_shape", "cap_surface",
                         "cap_color", "bruises", "odor",
                         "gill_attachement", "gill_spacing", "gill_size",
                         "gill_color", "stalk_shape", "stalk_root",
                         "stalk_surface_above_ring", "stalk_surface_below_ring",
                         "stalk_color_above_ring",
                         "stalk_color_below_ring", "veil_type", "veil_color",
                         "ring_number", "ring_type", "spore_print_color",
                         "population", "habitat")

# Recode each variable as a factor and display all factor levels for all variables:
mushrooms %>% mutate_all(as.factor) %>% map(levels)

$edibility
[1] "e" "p"
$cap_shape
[1] "b" "c" "f" "k" "s" "x"
$cap_surface
[1] "f" "g" "s" "y"
$cap_color
 [1] "b" "c" "e" "g" "n" "p" "r" "u" "w" "y"
$bruises
[1] "f" "t"
$odor
[1] "a" "c" "f" "l" "m" "n" "p" "s" "y"
$gill_attachement
[1] "a" "f"
$gill_spacing
[1] "c" "w"
$gill_size
[1] "b" "n"
$gill_color
 [1] "b" "e" "g" "h" "k" "n" "o" "p" "r" "u" "w" "y"
$stalk_shape
[1] "e" "t"
$stalk_root
[1] "b" "c" "e" "r"
$stalk_surface_above_ring
[1] "f" "k" "s" "y"
$stalk_surface_below_ring
[1] "f" "k" "s" "y"
$stalk_color_above_ring
[1] "b" "c" "e" "g" "n" "o" "p" "w" "y"
$stalk_color_below_ring
[1] "b" "c" "e" "g" "n" "o" "p" "w" "y"
$veil_type
[1] "p"
$veil_color
[1] "n" "o" "w" "y"
$ring_number
[1] "n" "o" "t"
$ring_type
[1] "e" "f" "l" "n" "p"
$spore_print_color
[1] "b" "h" "k" "n" "o" "r" "u" "w" "y"
$population
[1] "a" "c" "n" "s" "v" "y"
$habitat
[1] "d" "g" "l" "m" "p" "u" "w"
```

```r
# Define and rename the levels of all categorical variables:
levels(mushrooms$edibility) <- c("edible", "poisonous")
levels(mushrooms$cap_shape) <- c("bell", "conical", "flat", "knobbed", "sunken", "convex")
levels(mushrooms$cap_surface) <- c("fibrous", "grooves", "smooth", "scaly")
levels(mushrooms$cap_color) <- c("buff", "cinnamon", "red", "gray", "brown", "pink",
                                 "green", "purple", "white", "yellow")
levels(mushrooms$bruises) <- c("no", "yes")
levels(mushrooms$odor) <- c("almond", "creosote", "foul", "anise", "musty", "none", "pungent",
                            "spicy", "fishy")
levels(mushrooms$gill_attachement) <- c("attached", "free")
levels(mushrooms$gill_spacing) <- c("close", "crowded")
levels(mushrooms$gill_size) <- c("broad", "narrow")
levels(mushrooms$gill_color) <- c("buff", "red", "gray", "chocolate", "black", "brown",
                                  "orange","pink", "green", "purple", "white", "yellow")
levels(mushrooms$stalk_shape) <- c("enlarging", "tapering")
levels(mushrooms$stalk_root) <- c("bulbous", "club", "equal", "rooted")
levels(mushrooms$stalk_surface_above_ring) <- c("fibrous", "silky", "smooth", "scaly")
levels(mushrooms$stalk_surface_below_ring) <- c("fibrous", "silky", "smooth", "scaly")
levels(mushrooms$stalk_color_above_ring) <- c("buff", "cinnamon", "red", "gray", "brown",
                                              "orange", "pink", "white", "yellow")
levels(mushrooms$stalk_color_below_ring) <- c("buff", "cinnamon", "red", "gray", "brown",
                                              "orange", "pink","white", "yellow")
levels(mushrooms$veil_type) <- ("partial")
levels(mushrooms$veil_color) <- c("brown", "orange", "white", "yellow")
levels(mushrooms$ring_number) <- c("none", "one", "two")
levels(mushrooms$ring_type) <- c("evanescent", "flaring", "large", "none", "pendant")
levels(mushrooms$spore_print_color) <- c("buff", "chocolate", "black", "brown", "orange",
                                         "green", "purple", "white", "yellow")
levels(mushrooms$population) <- c("abundant", "clustered", "numerous", "scattered", "several",
                                  "solitary")
levels(mushrooms$habitat) <- c("woods", "grasses", "leaves", "meadows", "paths", "urban",
                               "waste")
mushrooms %>% map(levels) %>% map(length) %>% as.data.frame() # determine the number (length)
                                                               of factor levels for each variable
```

```
 edibility cap_shape cap_surface cap_color bruises odor gill_attachement gill_spacing gill_size gill_color
         2         6           4        10       2    9                2            2         2         12
 stalk_shape stalk_root stalk_surface_above_ring stalk_surface_below_ring stalk_color_above_ring
           2          4                        4                        4                     9
 stalk_color_below_ring veil_type veil_color ring_number ring_type spore_print_color population habitat
                      9         1          4           3         5                 9          6       7
```

We can see that variable veil_type has only one factor level: "partial". However,it will bring issues during modelling stage.R will throw an error for any categorical variable that has only one level and low counts. However, this variable will be kept because it has 8124 counts and as per assessment requirements to use all 23 variables.

```r
map_dbl(mushrooms, function(.x) {sum(is.na(.x))}) # determine the number of missing values in
                                                  each variable
```

```
             edibility              cap_shape            cap_surface             cap_color
                     0                      0                      0                     0
               bruises                   odor        gill_attachement           gill_spacing
                     0                      0                      0                     0
             gill_size             gill_color             stalk_shape             stalk_root
                     0                      0                      0                  2480
stalk_surface_above_ring stalk_surface_below_ring   stalk_color_above_ring   stalk_color_below_ring
                     0                      0                      0                     0
             veil_type              veil_color            ring_number             ring_type
                     0                      0                      0                     0
      spore_print_color             population                habitat
                     0                      0                      0
```

Stalk_root variable has 2480 missing values

Question 2 Randomly split the dataset into a training subset and a test subset containing 80% and 20% of the data.

```
set.seed(0) # random seed
no_observations <- dim(mushrooms)[1] # number of observations
test_index <- sample(no_observations, size = as.integer(no_observations*0.2), replace = FALSE)
                                                                    # 20% data for testing
train_index <- -test_index # remaining 80% data observations for training
train_set = mushrooms[train_index,] # assigning 80% train data into a variable train_set
test_set = mushrooms[-train_index,] # assigning 20% train data into a variable test_set

# Comparison of class label distribution among mushroom datasets (Original dataset
 (mushrooms), Training dataset and Testing dataset):

mush_original <- round(table(mushrooms$edibility) %>% prop.table() * 100, 1) # class label
        distribution as frequency and proportions of original dataset (mushrooms) dataset
mush_train <- round(table(train_set$edibility) %>% prop.table() * 100, 1) # class label
                          distribution as frequency and proportions of training dataset
mush_test <- round(table(test_set$edibility) %>% prop.table() * 100, 1) # class label
                           distribution as frequency and proportions of testing dataset

mush_df <- as.matrix(cbind(mush_original, mush_train, mush_test)) # create the dataframe
colnames(mush_df) <- c("Original", "Training set", "Test set") # assign name to columns
pander(mush_df, style = "rmarkdown", caption = paste0("Comparison of Class Label Distribution
 among Mushroom Datasets")) # print an R object in Pandoc's markdown
```

	Original	Training set	Test set
edible	51.8	51.6	52.5
poisonous	48.2	48.4	47.5

Comparison of Class Label Distribution Among Mushroom Datasets

Question 3 ***Define and justify*** *a classifier to classify the mushroom population into edible or poisonous. The classifier needs to use all 22 predictors (variables V2 toV23) to model the dependent variable (V1) [2 Marks]*

The naïve Bayes algorithm is one of the most effective and efficient supervised learning algorithms for data mining . First of all, one of the reasons why naïve Bayes classifier is considered for this analysis is because the naïve Bayes model can be built very quickly, even for large training data. For example, when the predictors are all categorical variables as in this case with mushroom dataset, all that is necessary for the analysis are tables with the frequency distributions of the training data. Secondly, because of its ability to produce accurate classification results even for large number of predictors with a minimal amount of training data. It is well known that its accuracy increases with the increase in the size of the training data.

It is consider as a simple classifier and it is called "naïve" because it utilizes Bayes theorem founded on Bayes' Rule, that simplifies the probabilities of the predictor values by strongly assuming that all of the predictors are conditionally independent from each other with regard to the class which means that the effect of a predictor value on a given class is independent of the values of the other predictors. In real world practical applications, this assumption is often violated and it is not realistic because predictors are often closely related. Nontheless, naïve Bayes often delivers competitive performace accuracy for categorical (nominal) input values even when the independence assumption is violated. Reduction in the complexity of the calculations based on this assumption leads to naïve Bayes being widely applied in practice. In the literature, naïve Bayes algorithm has proven its competitivness in different fields such as text classification, image processing, tumor diagnostics, marketing fault predictions and etc.

Question 4. ***Implement*** *the proposed classifier from step 3 using the training data subset from step 1.*

```
library(naivebayes)
set.seed(0) # random seed
NaiveBayesModel <- naive_bayes(edibility ~. , data = train_set, laplace = 1,
                               na.action = na.pass, prior = NULL,
                               usekernel = FALSE, usepoisson = FALSE) # implementing
 Naive_Bayes classifier using training  data subset via general formula interface of
 naive_bayes function in which predictors are assumed to be independent.

NaiveBayesModel
# Arguments of the function are:
# a) x can be matrix or dataframe with categorical (character/factor/logical) or metric
                                                                    (numeric) predictors;
# b) y is a class vector (character/factor/logical);
# c) formula an object of class "formula"of the form: class ~ predictors (class has to be a
                                                                    factor/character/logical);
# d) data matrix or dataframe with categorical (character/factor/logical) or metric (numeric)
                                                                        predictors;
# e) prior vector with prior probabilities of the classes;
# f) laplace value used for Laplace smoothing. Defaults is 0;
# g) usekernel logical; if TRUE, density is used to estimate the class conditional densities of
                                                                        metric predictors;
# h) na.action is a function which indicates what should happen when the data contain NAs. By
                                                                            default
# (na.pass), missing values are not removed from the data and are then omited while
                                                                    constructing tables.
```

```
================================================== Naive Bayes ==================================================

Call:
naive_bayes.formula(formula = edibility ~ ., data = train_set,
    prior = NULL, laplace = 1, usekernel = FALSE, usepoisson = FALSE,
    na.action = na.pass)

---------------------------------------------------------------------------------------------------------------

Laplace smoothing: 1

---------------------------------------------------------------------------------------------------------------

A priori probabilities:

   edible poisonous
0.5175385 0.4824615

---------------------------------------------------------------------------------------------------------------

Tables:

---------------------------------------------------------------------------------------------------------------
::: cap_shape (Categorical)
---------------------------------------------------------------------------------------------------------------

cap_shape       edible    poisonous
  bell     0.0943620178 0.0105028644
  conical  0.0002967359 0.0015913431
  flat     0.3801186944 0.3978357734
  knobbed  0.0537091988 0.1524506684
  sunken   0.0074183976 0.0003182686
  convex   0.4640949555 0.4373010821

---------------------------------------------------------------------------------------------------------------
```

```
--------------------------------------------------------------------------------------------------------------------------------
 ::: cap_surface (Categorical)
--------------------------------------------------------------------------------------------------------------------------------

cap_surface       edible    poisonous
    fibrous 0.3723277910 0.1964968153
    grooves 0.0002969121 0.0015923567
    smooth  0.2663301663 0.3614649682
    scaly   0.3610451306 0.4404458599

--------------------------------------------------------------------------------------------------------------------------------
 ::: cap_color (Categorical)
--------------------------------------------------------------------------------------------------------------------------------

cap_color       edible   poisonous
  buff     0.011558980 0.029561348
  cinnamon 0.008298755 0.002860776
  red      0.147302905 0.222186904
  gray     0.242738589 0.209154482
  brown    0.300829876 0.260648442
  pink     0.012151749 0.021614749
  green    0.004742146 0.000317864
  purple   0.004149378 0.000317864
  white    0.169531713 0.078830261
  yellow   0.098695910 0.174507311

--------------------------------------------------------------------------------------------------------------------------------
 ::: bruises (Bernoulli)
--------------------------------------------------------------------------------------------------------------------------------

bruises    edible poisonous
    no  0.3472965 0.8492670
    yes 0.6527035 0.1507330

--------------------------------------------------------------------------------------------------------------------------------
 ::: odor (Categorical)
--------------------------------------------------------------------------------------------------------------------------------


odor            edible   poisonous
  almond   0.094871035 0.000317965
  creosote 0.000296472 0.050556439
  foul     0.000296472 0.556756757
  anise    0.098132227 0.000317965
  musty    0.000296472 0.009538951
  none     0.805217907 0.027980922
  pungent  0.000296472 0.061049285
  spicy    0.000296472 0.148489666
  fishy    0.000296472 0.144992051

--------------------------------------------------------------------------------------------------------------------------------

# ... and 17 more tables

--------------------------------------------------------------------------------------------------------------------------------
```

Question 5 ***Display*** *the summary of the fitted model implemented in Question 4.*

```
summary(NaiveBayesModel)
# Calculate model performance metrics on test subset of data

nb_metrics = function(model, data){
  predictClass <- predict(model, newdata = data[, -1],  type = "class")
  (cm_nb = table(predictClass, test_set$edibility)) # create the confusion matrix
  accuracy = round((cm_nb[2,2] + cm_nb[1,1]) / sum(cm_nb[2,2],cm_nb[2,1],cm_nb[1,1],
  cm_nb[1,2])*100, 2) # calculate accuracy
  class_error_rate = round(sum(cm_nb[2,1] + cm_nb[1,2]) / sum(cm_nb[2,2] + cm_nb[2,1] +
  cm_nb[1,1] + cm_nb[1,2])*100,2) # calculate class_error_rate
  precision = round(cm_nb[2,2] / sum(cm_nb[2,2] + cm_nb[2,1]),2) # calculate precision
  sensitivity = round(cm_nb[2,2] / sum(cm_nb[2,2] + cm_nb[1,2]),2) # calculate sensitivity
  specificity = round(cm_nb[1,1] / sum(cm_nb[1,1] + cm_nb[2,1]),2) # calculate specificity
  recall = round((cm_nb[2,2])/sum(cm_nb[2,2] + cm_nb[1,2]),2) # calculate recall
  f1_score = round((2 * precision * sensitivity) / (precision + sensitivity),2) # calculate F1
                                                                                    score
  metrics = c(accuracy, class_error_rate, precision, sensitivity, specificity, recall,
 f1_score)
  names(metrics) = c("Accuracy", "Class_Error_Rate", "Precision", "Sensitivity", "Specificity",
 "Recall", "F1 score")
  return(metrics)
}

# Calculate model performance metrics on test subset set
nb_metrics(NaiveBayesModel, test_set)
```

Accuracy	Class_Error_Rate	Precision	Sensitivity	Specificity	Recall	F1 score
95.32	4.68	1.00	0.91	1.00	0.91	0.95

```
# Calculate classification rate on training and testing subsets of data:
mush_nb_trn_pred <- predict(NaiveBayesModel, train_set,  type = "class") # predict probability
                                                                           for training set
mush_nb_tst_pred = predict(NaiveBayesModel, test_set[, -1], type = "class") # predict
                                                                             probability for testing set

calc_class_err = function(actual, predicted) {
  mean(actual != predicted)
}
nb_train_err <- calc_class_err(predicted = mush_nb_trn_pred, actual = train_set$edibility)
                                                                    # apply function on training set
nb_test_err <- calc_class_err(predicted = mush_nb_tst_pred, actual = test_set$edibility)
                                                                  # apply function on testing set

training_set <- round(c(nb_train_err)*100, 2) # making percentages for train error
testing_set <- round(c(nb_test_err)*100, 2) # making percentages for test error
method <- c("NaiveBayes") # assigning a vector method
mush_df <- as.matrix(cbind(method, training_set,testing_set)) # create the dataframe
colnames(mush_df) <- c("Method", "Training set", "Test set") # assign name to columns
pander(mush_df, style = "rmarkdown", caption = paste0("Comparison of Classification Error Rates
 among Mushroom Datasets")) # print an R object in Pandoc's markdown
```

	Training set	*Test set*
NaiveBayes	*4.45*	*4.68*

Comparison of Class Label Distribution Among Mushroom Datasets

Question 6 ***Interpret*** *the relationships between the predictor and features of the fitted model using the role of a Naïve Bayes classifier. I.e.* ***Explain*** *the relationships of the model using Bayes theorem.*

The Naïve Bayes classifier is a probabilistic classifier based on applying Bayes' Theorem with strong assumptions between attributes. This model simplifies the probabilities of the predictor values by assuming that all of the predictors are independent of the others. This assumption of independence results into a significant reduction in the complexity of the calculations.

Namely, Bayes' Rule provides a strong tool to answer the question that based on the predictors that we are analyzing what is the probability that the outcome will be class C_1 for example, with this equation:

$$Pr[Y = C_\ell|X] = \frac{Pr[Y]Pr[X|Y = C_\ell]}{Pr[X]}$$

a) $Pr[Y = C_1|X]$ is posterior probability of the class;
b) $Pr[Y]$ is the prior probability of the outcome;
c) $Pr[X]$ is the probability of the predictor values;
d) $Pr[X|Y = C_1]$ is conditional probability

What is the probability to observe predictor values related to that data of class C_1. This is very complex calculation unless strong assumptions are postulated. To calculate the conditional probability we will use probability densities of each individual predictor:

$$Pr[X|Y = C_\ell] = \prod_{j=1}^{P} Pr[X_j|Y = C_\ell]$$

To estimate individual probaility then we have to make an assumption of normality for continous predictors, or to use other methods like nonparametric kernel density estimators. For the categorical predictors, the probaility distribution can be determined by observing frequencies in the training data set. If we visualize distribution of predictors and they are overlapping, it's unlikely that they will be independent. In order to compute overal probability than each predictor will be considered separately. To produce the class probability $Pr[X|Y = C_1]$ for the first class, two conditional probability values will be determined for predictors A and B then multiplied together to calculate the overall conditional probability for the class. For probability of the predictor values [X], everything is the same, except the probabilities for predictors A and B would be determined from the entire training set for both classess.

When the correlation between the predictors is very strong, then is almost unlikely to have a new sample. But if we use tha assumption of independence this probability will be overestimated. Bayes' Rule is essentially a probability statement. Class probabilities are created and the predicted class is the one associated with the largest class probability. The essence of the model is the determination of the conditional and unconditional probabilities associated with the predictors. This models accepts that not all of the predictors to be included in predicting probabilites due to the independence assumption. For example, the Bayes theorem equation to calculate posterior probability of poisonous mushroom that was bruised is:

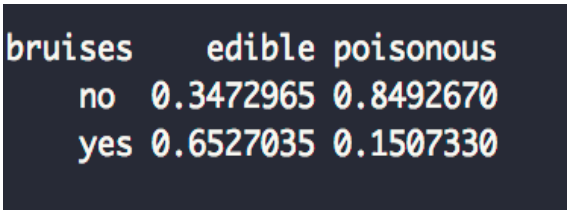
$$P(Poisonous|Bruised) = \frac{P(Bruised|Poisonous) * P(Poisonous)}{P(Bruised)}$$

We can subset bruised mushrooms from the training data subset with:

```
table(mushrooms$bruises, mushrooms$edibility)
```

	edible	poisonous
no	1456	3292
yes	2752	624

Calculations:
P(Bruised\Poisonous) = 624/3916 = 0.1594
P(Bruised) = 3376/6500 = 0.5194
P(Poisonous) = 3916/6500 = 0. 6025
P(Poisonous\Bruised) = (0.1594 * 0.6025)/0.5194 = 0.1849 * 100 = 18.49%
There is 18.49% chance that you select a bruised poisonous mushroom from the training dataset.
The tables of posterior probabilites as part of naïve_bayes() function show the probabilites of edible and posinous mushrooms given it were bruised.



bruises	edible	poisonous
no	0.3472965	0.8492670
yes	0.6527035	0.1507330

Our result of 18.49% is close to 0.1507330 or 15%. The difference in the results is perhaps due to random number generation during sampling the training and testing subsets data.

Task 2: Comparison of Classifiers

In this task compare the performance of the supervised learning algorithms Linear Discriminant Analysis, Quadratic Discriminant Analysis and the Naïve Bayes Classifier using a publicly available Blood Pressure Data. The data to be used for this task is provided in the HBblood.csv file.

The HBblood.csv dataset contains values of the percent HbA1c (a measure of the amount of glucose and haemoglobin joined together in blood) and systolic blood pressure (SBP) (in mm/Hg) for 1,200 clinically healthy female patients within the ages 60 to 70 years. Additionally, the ethnicity, *Ethno*, for each patient was recorded and discombobulated into three groups, *A, B or C,* for analysis.

Question 7 ***Discuss*** *the properties of using the supervised learning algorithms Linear Discriminant Analysis, Quadratic Discriminant Analysis and the Naïve Bayes Classifier to predict Ethno using HbA1c and SBP as the feature variables.*

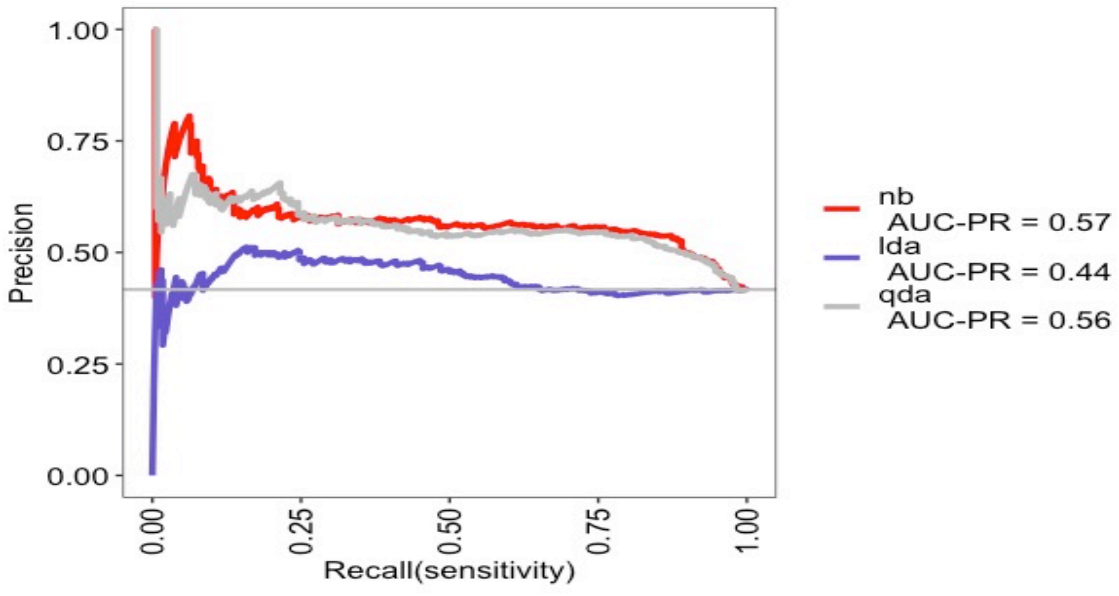


Fig. Recall and Precision estimation for naiveBayes, LDA and QDA models

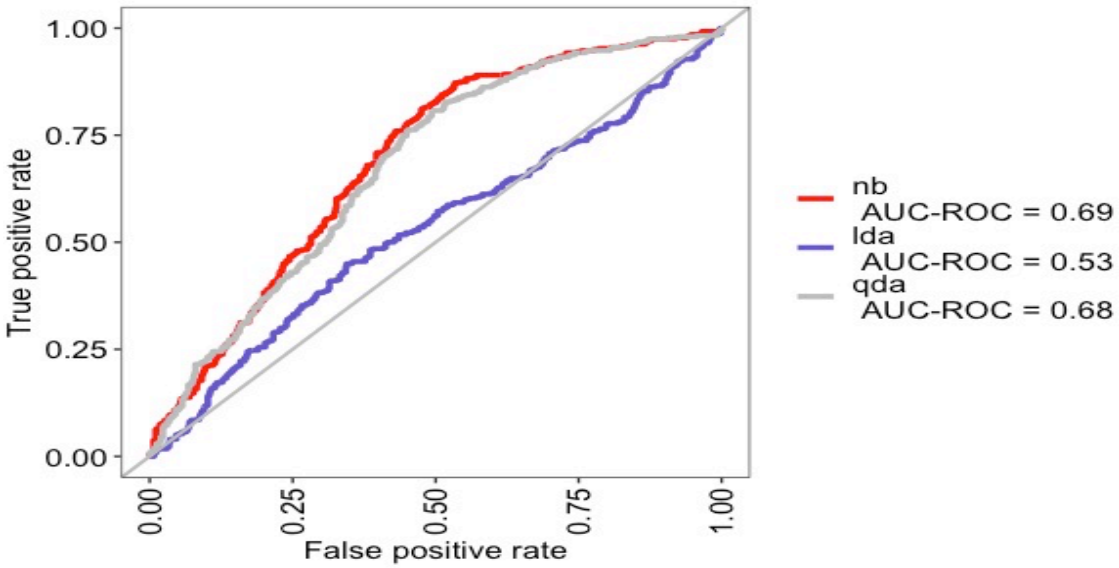


Fig 2: ROC curves for naïve bayes, LDA and QDA models

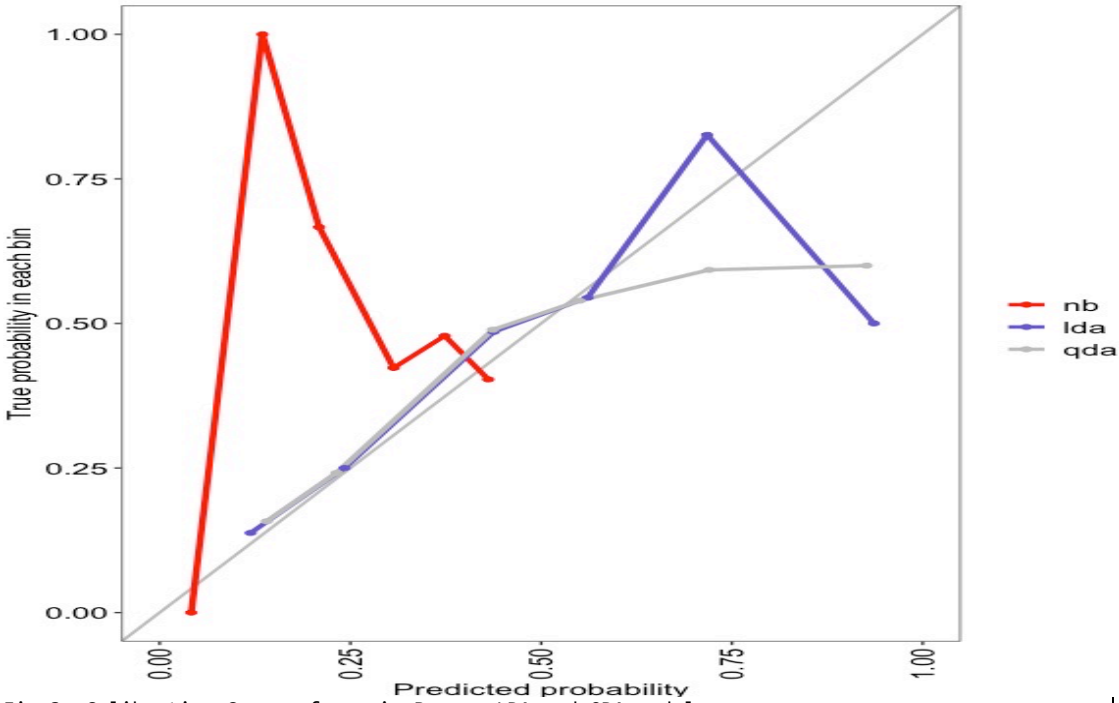


Fig 3: Calibration Curves for naiveBayes, LDA and QDA models

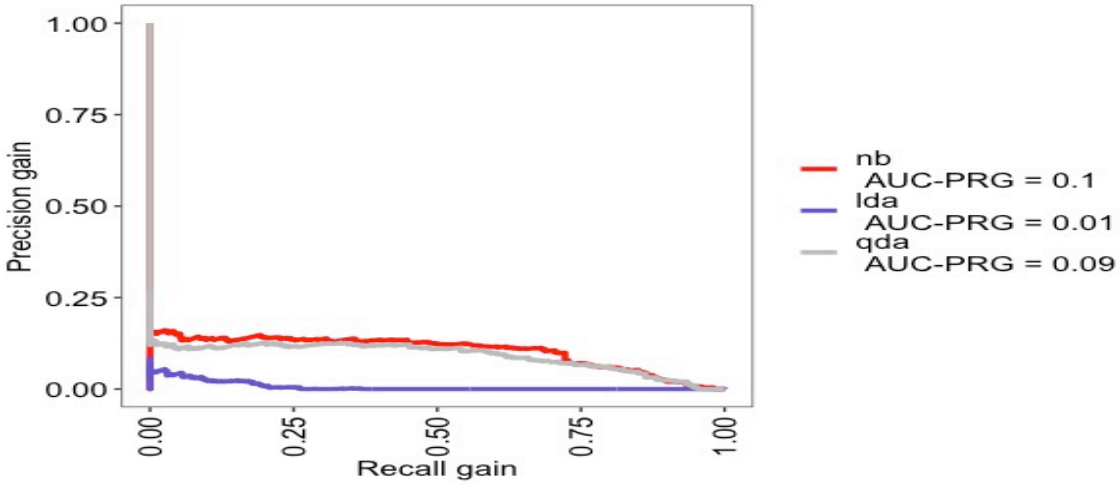


Fig 4. Recal gain versus Precision gain for naïve bayes, LDA and QDA models

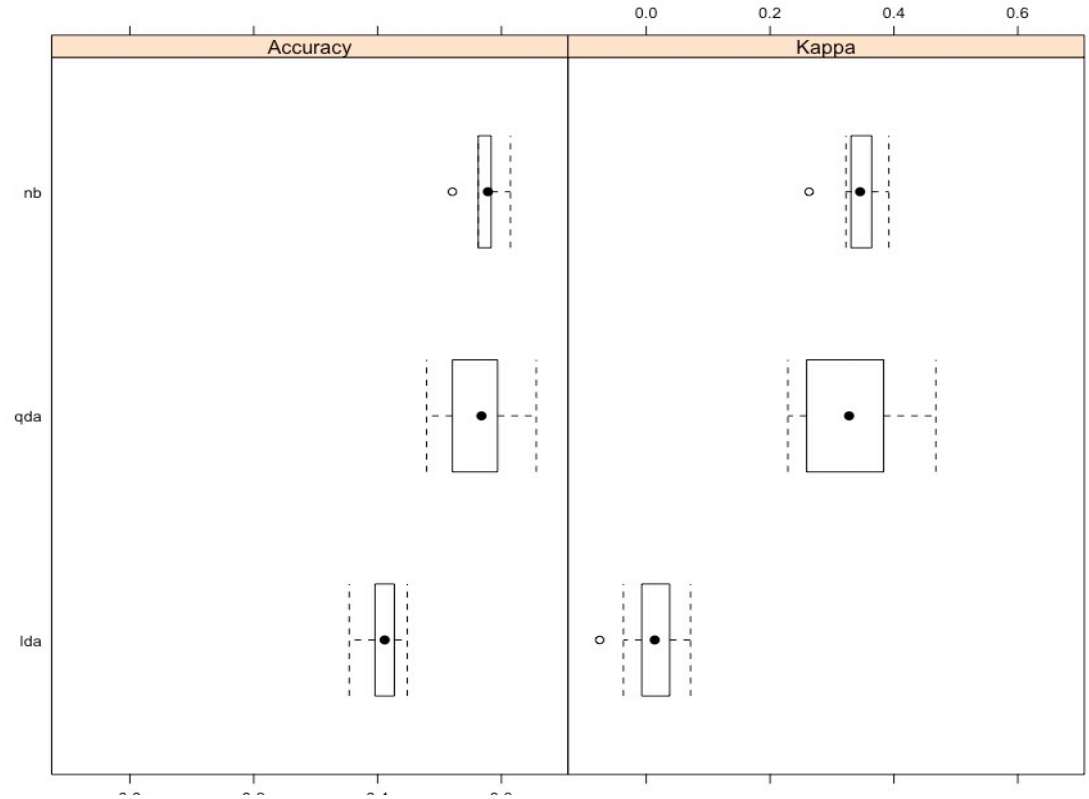


Fig 4. Accuracy and Kappa measures with Confidence level of 0.95 for naiveBayes, LDA and QDA

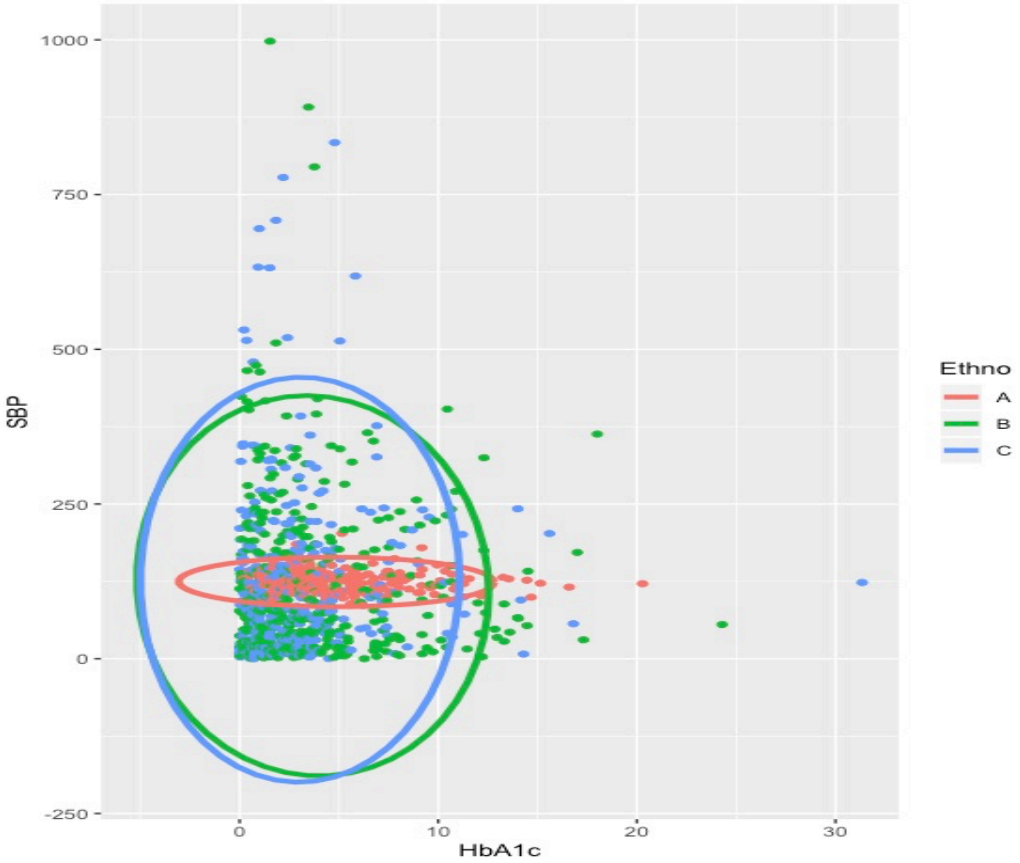


Fig 5: Density plot of the predictors by Ethno variable in the training subset

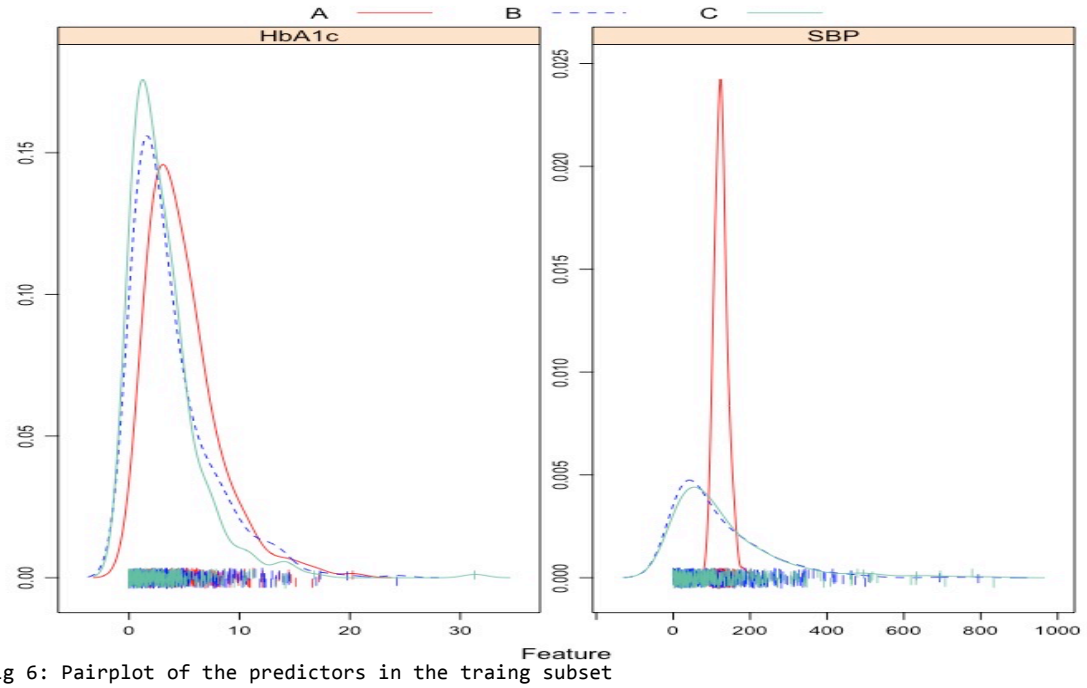


Fig 6: Pairplot of the predictors in the traing subset

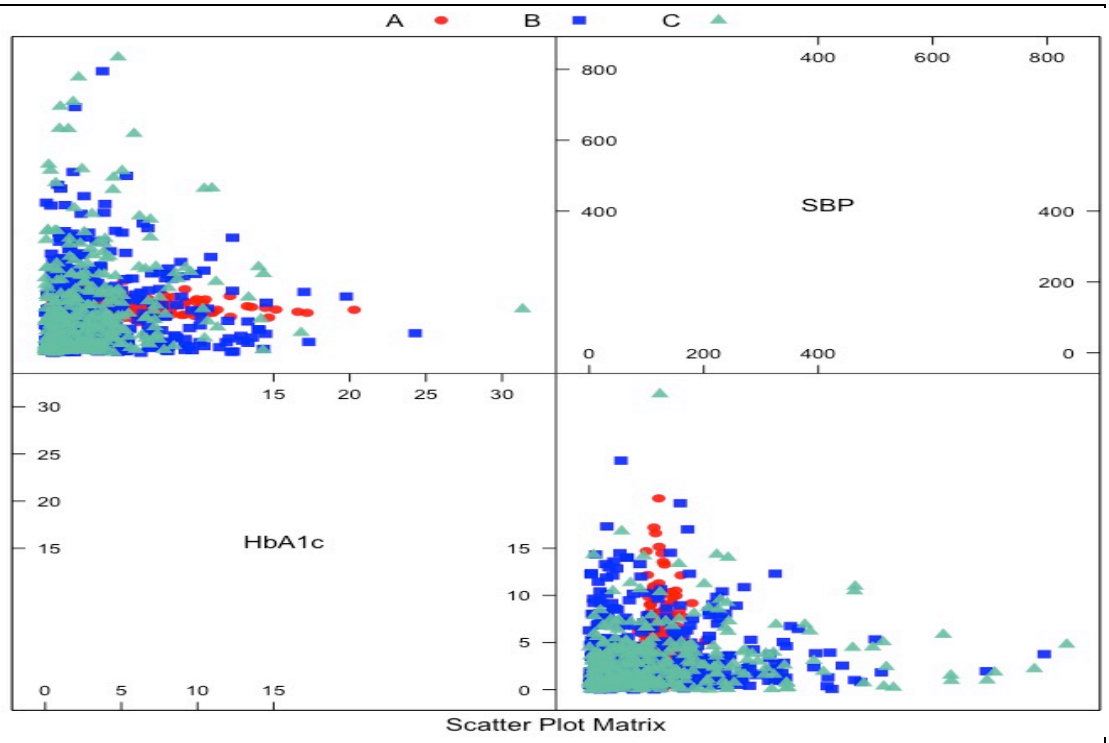


Fig 7. Scareplot of the predictor in the training subset

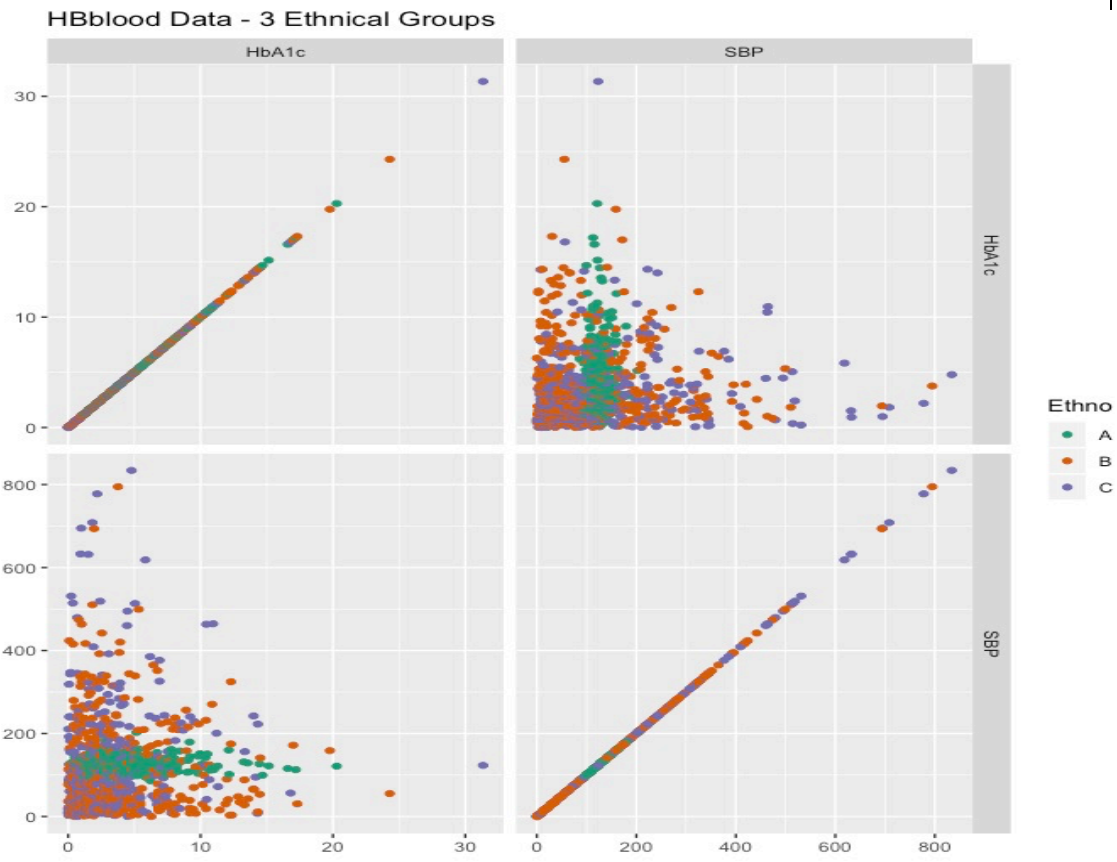


Fig 8: Pairplot of the classess of the response variable Ethno with library(WVPlots)

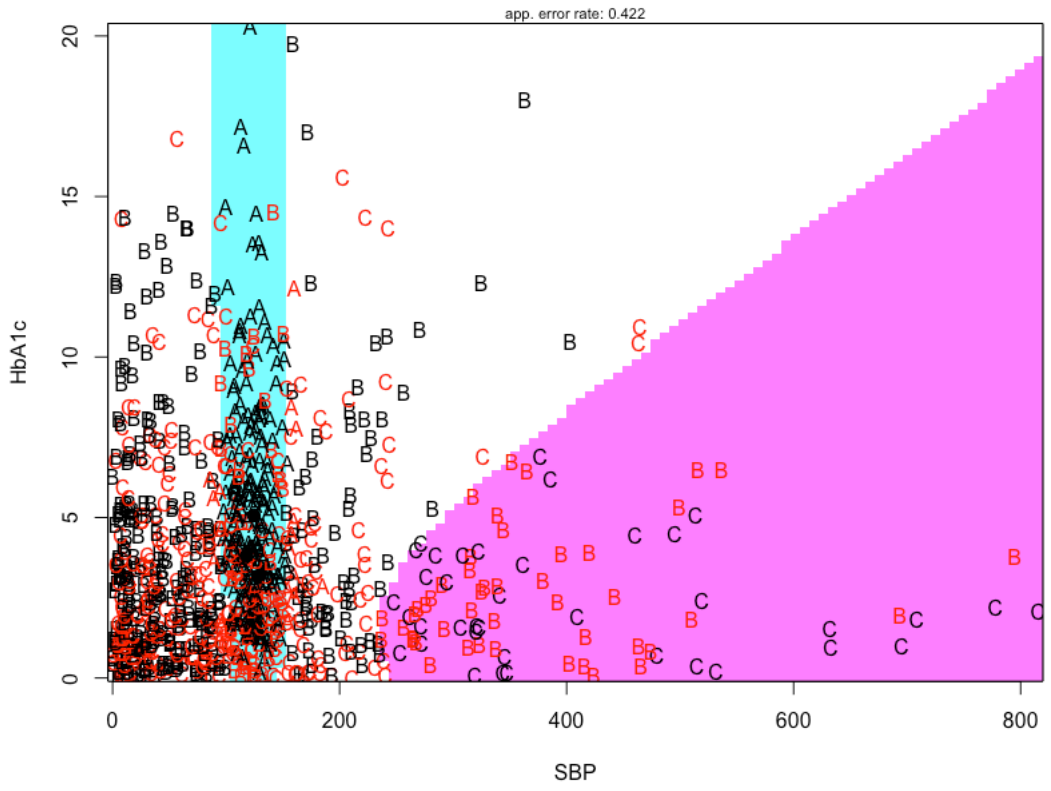


Fig 9: Decision boundries plot for QDA

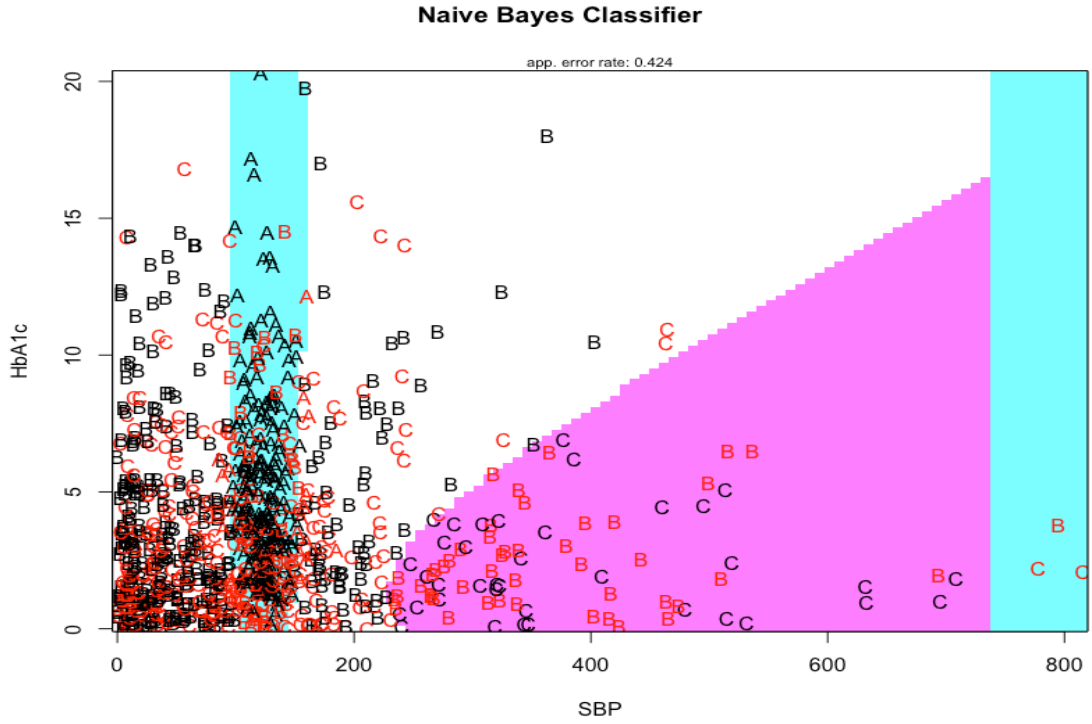


Fig 9: Decision boundries plot for naïve Bayes

Generative methods model estimate the joint probability, p(x,y), often by assuming some distribution for the conditional distribution of X given Y, f(x|y). Bayes theorem is then applied to classify according to p(y|x).

LDA assumes that the predictors are multivariate normal conditioned on the classes.
X|Y=k∼N(μk,Σ)
fk(x)=1(2π)p/2|Σ|1/2exp[−12(x−μk)′Σ−1(x−μk)]
where Σ does **not** depend on k, that is, we are assuming the same Σ for each class.

QDA also assumes that the predictors are multivariate normal conditioned on the classes.
X|Y=k∼N(μk,Σk)
fk(x)=1(2π)p/2|Σk|1/2exp[−12(x−μk)′Σ−1k(x−μk)]
wher Σk **does** depend on k, that is, we are allowing a different Σk for each class. We only use information from class k to estimate Σk.

Naive Bayes comes in different form and with only numeric predictors, it often assumes a multivariate normal conditioned on the classes, but a very specific multivariate normal.
X|Y=k∼N(μk,Σk)
Naive Bayes assumes that the predictors X1,X2,…,Xp are independent. This is the "naive" part of naive Bayes. Since X1,X2,…,Xp are assumed independent, each Σk is diagonal, that is, we assume no correlation between predictors. Independence implies zero correlation.

One assumption that is common for all three models is that all predictors are normally distributed or follow a Gaussian distribution. Density and pair plots (fig 5, fig. 6, fig 7, fg8) plot shows that this assumption for both numeric variables for all three classes was not met because they were extremely positively skewed. Another assumption violation observed in the plots is that there are not an equal number of instances in classes A, B and C in this datasets that is violating the equal variance assumption.

The best performing classifier according to my results is QDA with 65% of accuracy, then naïve Bayes with 61% and only 44% for LDA.

Task 3: Implementation of Classifiers

In this task, compare the performance of the supervised learning algorithms Linear Discriminant Analysis and the Naïve Bayes Classifier using a publicly available Heart Diseases Data. The Heart.txt data contains average systolic blood pressures (SBP) for Men and Women from 41 different countries. It also gives the 95% confidence interval for each estimated blood pressure measure

Question 8 ***Implement*** *both the LDA and Naïve Bayes classifiers using the Heart.csv data to classify gender (women and men) using the variable SBP only.*

```
library(dplyr)
library(ggplot2)
library(ggridges)
library(tidyverse)
library(caret)
library(questionr)
library(MLmetrics)
library(lattice)
library(MASS)
library(pander)
library(tibble)
library(purrr)

heartData = read.table('heart.txt', col.names = c('Country', 'Sex', "SBP", "SBP_LCI",
 "SBP_UCI")) # read the data and load into R environment as a dataframe heartData

summary(heartData) # summarize the variable distribution
We can see that there are no missing values in the data.
```

```
 Country       Sex          SBP            SBP_LCI         SBP_UCI
World:82   Men  :41   Min.   :122.3   Min.   :121.0   Min.   :122.8
           Women:41   1st Qu.:122.8   1st Qu.:121.8   1st Qu.:123.6
                      Median :125.0   Median :123.1   Median :126.7
                      Mean   :124.6   Mean   :123.6   Mean   :125.7
                      3rd Qu.:126.4   3rd Qu.:125.3   3rd Qu.:127.3
                      Max.   :127.0   Max.   :126.1   Max.   :129.3
```

```
str(heartData) # data structure compact display
```

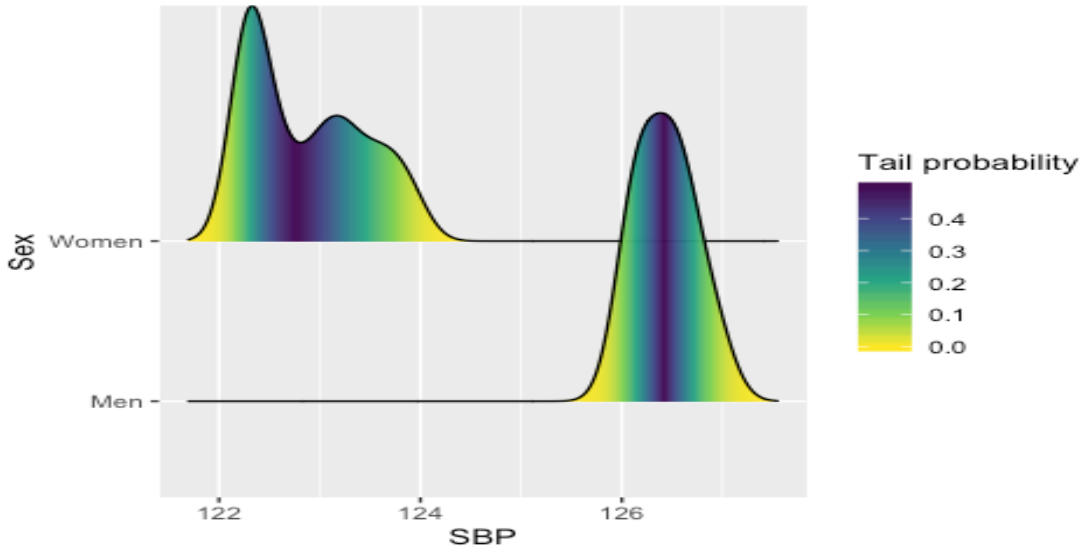
```
'data.frame':   82 obs. of  5 variables:
 $ Country: Factor w/ 1 level "World": 1 1 1 1 1 1 1 1 1 1 ...
 $ Sex    : Factor w/ 2 levels "Men","Women": 1 1 1 1 1 1 1 1 1 1 ...
 $ SBP    : num  127 127 127 127 127 ...
 $ SBP_LCI: num  124 124 124 125 125 ...
 $ SBP_UCI: num  129 129 129 129 128 ...
```

```
heartData$Sex = factor(heartData$Sex, ordered = FALSE, levels = c("Men", "Women")) # coerce
                                                                    variable Sex to a factor
heartData[,c(1,4,5)] <- NULL # remove the variables(Country, SBP_LCI and SBP_UCI)

# Randomly split the dataset into a training subset and a test subset containing 80% and 20% of
 the data:

set.seed(0) # random seed
heartsample <- caret::createDataPartition(y = heartData$Sex, times = 1, p = 0.8, list = FALSE)
# create a list of 80% of the rows in the original dataset we can use for training
train_heart <- heartData[heartsample, ] # use the remaining 80% of data to train the models
test_heart <- heartData[-heartsample, ] # select 20% of the data for testing
```

```
# Exploartory analysis of the train data:
# Density plots for the predictors with Ridgeline graph, also called a joyplot. It displays the
 distribution of a quantitative variable for Sex categorical variable. This allows us to map
 the probabilities in color:
Sex <- heartData$Sex # assigning variable as a vector
SBP <- heartData$SBP # assigning variable as a vector
ggplot(train_heart,
       aes(x = SBP,
           y = Sex,
           fill = 0.5 - abs(0.5 - stat(ecdf)))) +
  stat_density_ridges(geom = "density_ridges_gradient", calc_ecdf = TRUE) +
  scale_fill_viridis_c(name = "Tail probability", direction = -1)
```




```
Fig 1: Density plot of class probabilities on original dataset(heartdata)
# Comparison of class label distribution among datasets (original dataset (heartData), training
 dataset and testing dataset):
freq(train_heart$Sex, cum = FALSE, sort = "dec", total = TRUE) # class label distribution as
                                          frequency and proportions of training dataset

       n   % val%
 Men    33  50   50
 Women  33  50   50
 Total  66 100  100
Fig 2: Proprtion table for training dataset
freq(heartData$Sex, cum = FALSE, sort = "dec", total = TRUE) # class label distribution as
                                      frequency and proportions of original dataset (heart)

       n   % val%
  Men   41  50   50
  Women 41  50   50
  Total 82 100  100
Fig 3: Proprtion table for original dataset

freq(test_heart$Sex, cum = FALSE, sort = "dec", total = TRUE) # class label distribution as
                                            frequency and proportions of testing dataset

       n   % val%
  Men    8  50   50
  Women  8  50   50
  Total 16 100  100
Fig 4: Proprtion table for testing dataset
```

```r
# Implementing the models: LDA naive Bayes:
# set up tuning grid
set.seed(0) # random seed
control = trainControl(method = "cv", number = 5, repeats = 3, classProbs = TRUE) # control the
                                                    computational nuances of the train function
searchGrid <- expand.grid(fL = c(0:5), usekernel = c(TRUE, FALSE), adjust = seq(0, 5, by = 1))
                                              # define the grid parameters for the train function
naiveBayesModel <- train(Sex ~., data = train_heart, method = "nb", trControl = control,
                         na.action = "na.pass", metric = "Accuracy", tuneGrid = expand.grid(fL
 = 1, usekernel = FALSE, adjust = 1)) # fit NaiveBayes model with caret package

fit.lda <- train(Sex ~., data = train_heart, method = "lda", metric = "Accuracy", trControl =
 control) # implement the LDA model

# Make prediction for the new data on test subsets of data:
predictClass_1 <- predict(naiveBayesModel, test_heart) # test subset(naïve Bayes model)
predictClass_2 <- predict(fit.lda, test_heart) # make predictions on test subset (LDA model)

# Displaying model metrics of performance with the help of confusion matrix:

cm_nb <- confusionMatrix(reference = test_heart$Sex, data = predictClass_1,
                         mode = "everything", positive = "Men")
cm_nb # display the performace metics for naive Bayes model


Confusion Matrix and Statistics

          Reference
Prediction Men Women
     Men     8     0
     Women   0     8

               Accuracy : 1
                 95% CI : (0.7941, 1)
    No Information Rate : 0.5
    P-Value [Acc > NIR] : 1.526e-05

                  Kappa : 1

 Mcnemar's Test P-Value : NA

            Sensitivity : 1.0
            Specificity : 1.0
         Pos Pred Value : 1.0
         Neg Pred Value : 1.0
              Precision : 1.0
                 Recall : 1.0
                     F1 : 1.0
             Prevalence : 0.5
         Detection Rate : 0.5
   Detection Prevalence : 0.5
      Balanced Accuracy : 1.0

       'Positive' Class : Men
```

Fig 5: Performance metrics with Naïve bayes modet on testing dataset

```
cm_lda <- confusionMatrix(reference = test_heart$Sex, data = predictClass_2,
                          mode = "everything", positive = "Men")
cm_lda # display the performace metics for LDA model

Confusion Matrix and Statistics

          Reference
Prediction Men Women
     Men     8     0
     Women   0     8

               Accuracy : 1
                 95% CI : (0.7941, 1)
    No Information Rate : 0.5
    P-Value [Acc > NIR] : 1.526e-05

                  Kappa : 1

 Mcnemar's Test P-Value : NA

            Sensitivity : 1.0
            Specificity : 1.0
         Pos Pred Value : 1.0
         Neg Pred Value : 1.0
              Precision : 1.0
                 Recall : 1.0
                     F1 : 1.0
             Prevalence : 0.5
         Detection Rate : 0.5
   Detection Prevalence : 0.5
      Balanced Accuracy : 1.0

       'Positive' Class : Men
Fig 6: Performance metrics with Naïve bayes modet on testing dataset
# Present the summary of Accuracy and Kappa for both models:
results <- resamples(list(lda = fit.lda, nb = naiveBayesModel)) # connect the results
summary(results) # summarise the results
Call:
  summary.resamples(object = results)

  Models: lda, nb
  Number of resamples: 5

  Accuracy
      Min. 1st Qu. Median Mean 3rd Qu. Max. NA's
  lda    1       1      1    1       1    1    0
  nb     1       1      1    1       1    1    0

  Kappa
      Min. 1st Qu. Median Mean 3rd Qu. Max. NA's
  lda    1       1      1    1       1    1    0
  nb     1       1      1    1       1    1    0
Fig 7: Performance metrics (Accuracy and Kappa) with Naïve bayes and LDA models

# Measures of predicted classes for both models:
models <- list(nb = naiveBayesModel, lda = fit.lda) # list of models
testPred <- predict(models, newdata = test_heart) # predictions on test data of both models
lapply(testPred, function(x)x) # returns a list of the same length as X, each element of which
 is the result of applying FUN to the corresponding element of X

 $nb
 [1] Men   Men   Men   Men   Men   Men   Men   Men   Women Women Women Women Women Women Women
 Women
Levels: Men Women

$lda
 [1] Men   Men   Men   Men   Men   Men   Men   Men   Women Women Women Women Women Women Women
 Women
Levels: Men Women
Fig 8: Predicted classes on both models
```

```
cm_list <- list(naiveBayesModel = cm_nb, LDA = cm_lda)
results <- map_df(cm_list, function(x)x$byClass) %>% as_tibble() %>%
     mutate(stat = names(cm_nb$"byClass")) # integrated display of summary statistics for both
 models
```

Question 9 ***Compare*** *the model error of the LDA and Naïve Bayes classifiers derived in Question 8.*

```
Calculate classification error rate of the modelsperformance:

heart_nb_trn_pred <- predict(naiveBayesModel, train_heart) # predict probability
# for training set naiveBayes
heart_nb_tst_pred = predict(naiveBayesModel, test_heart) # predict
# probability for testing set naiveBayes

heart_lda_trn_pred <- predict(fit.lda, train_heart) # predict probability
# for training set LDA

calc_class_err = function(actual, predicted) {
  mean(actual != predicted)
}
nb_train_err <- calc_class_err(predicted = heart_nb_trn_pred, actual = train_heart$Sex)
nb_test_err <- calc_class_err(predicted = heart_nb_tst_pred, actual = test_heart$Sex)
# calculate error for training and test subsets for naïve Bayes model

lda_train_err <- calc_class_err(predicted = heart_lda_trn_pred, actual = train_heart$Sex)
lda_test_err <- calc_class_err(predicted = heart_lda_tst_pred, actual = test_heart$Sex)
# calculate error for training and test subsets for LDA model

training_set <- c(nb_train_err, lda_train_err)
testing_set <- c(nb_test_err, lda_test_err)
method <- c("NaiveBayes", "LDA")
heart_df <- as.data.frame(cbind(method, training_set,testing_set)) # create the dataframe
heart_df

colnames(heart_df) <- c("Method", "Training set", "Test set") # assign name to columns
pander(heart_df, style = "rmarkdown", caption = paste0("Comparison of Classification Error
 Rates among heartData Datasets")) # print an R object in Pandoc's markdown

Fig 9: Comparison of Classification Error Rates Among heartData Datasets

 Method      Training set   Test set
 NaiveBayes  0              0
 LDA         0              0
```

Question 10 ***Discuss*** *your findings from Question 8 and Question 9 using the algorithm assumptions of both LDA and Naïve Bayes classifiers as the basis of your discussion.*

The summary result form the Naïve Bayes (NB) and Linear Discriminant Analysis (LDA) both yielded the same predictive results on the test data and train data. This is evident in the Summary Matrix and Statistics tables, Classification Error Rates table, ROC curves, Calibration Curves, Lift Charts and Precision Recall Gain Curves (see plots below). The performance of both models are absolutely equivalent, accuracy of probability is 100 % on both models and both models are favoured. One way to assess the quality of the class probabilities is using a calibration plot. Created Calibration plots on test data are overlapping and this pattern is indicative of a model that has both excellent calibration and excellent performance.

The ROC curve is used for valuating the class probabilites and for a quantitative assessment of the models. Both models are perfect models because they completely separate the two classes (Man and Woman) and have 100% sensitivity and specificity. Graphically, these ROC curve would be a single step between (0, 0) and (0, 1) and the area under the ROC curve for both model is one.

The train, test and original datasets have an equal number of both classes that is shown by frequency and proprtion tables (fig2, fi3, and fig 4) and the distribution of classes that is shown with the density plots (fig 1). The dataset of interest has an equal number of instances in the two classes implying that it has equal variance of classes. This fact is

adhered to the first assumtipn by these models.

The distribution of classes that is shown with the pair density plots (fig 1). It is normal Gaussian distribution of both classes that is adhered to the second assumption by these models.

Both models assume that there is a real correlation between the predictors, according to thethird assumption of the models. Generated pair plot (library WVPlots) demonstrates a very large differences between the predictor values (SBP) between classes.

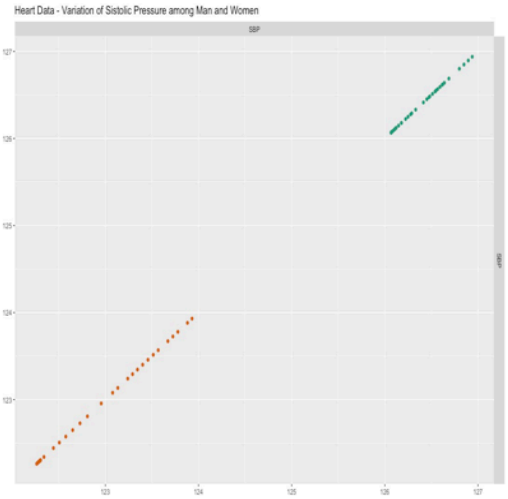


Fig 10: Pair plot of distribution of classes

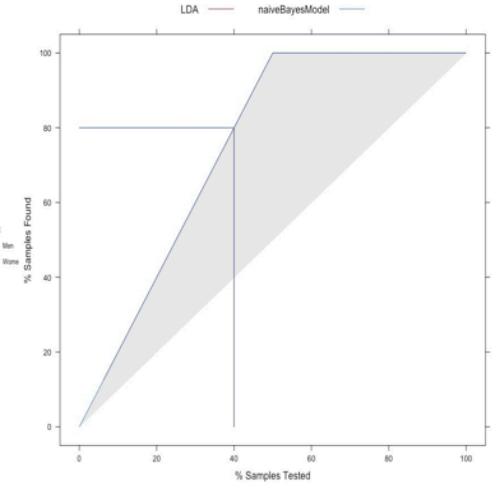
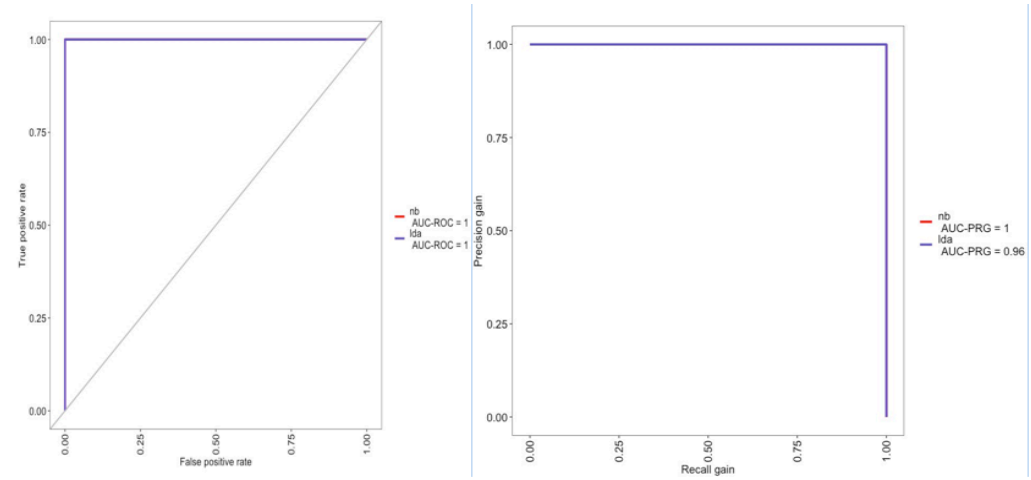


Fig 11. Lift chart for naïve Bayes and LDA models on testing datasets



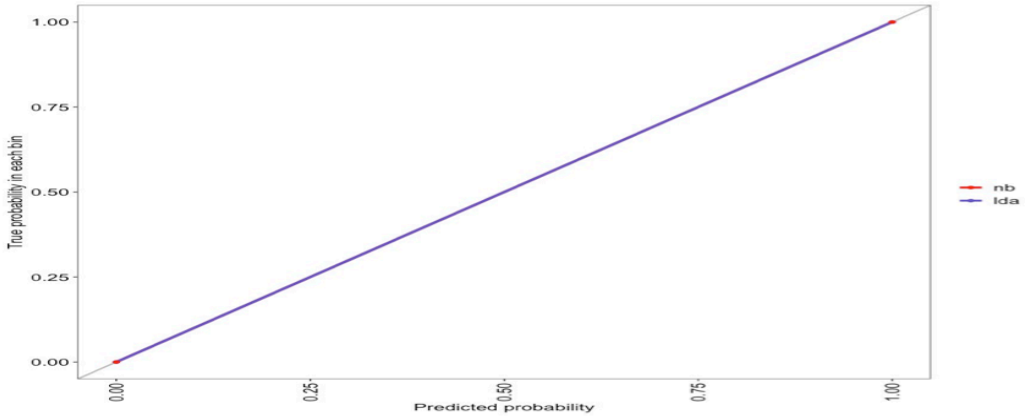


Fig 12: ROC Curve, Recall gain Curve and Calibration Curve for naïve Bayes and LDA models