DiabeticPrediction

Bita Parsa

26/11/2021

Diabitics Pridiction Project

Introduction

Machine learning plays an essential role in predicting the disease based on the symptoms. This data set was prepared by "National Institute of Diabetes and Digestive and Kidney Diseases" as part of the Pima Indians Diabetes Database. All patients here belong to the Pima Indian heritage (a subgroup of Native Americans) and are females aged 21 and above. In this project, we try to predict if the patient has diabetes or not.

Installing required packages

Required packages will be installed by following codes:

```r
if(!require(tidyverse)) install.packages("tidyverse", repos = "http://cran.us.r-project.org")
if(!require(caret)) install.packages("caret", repos = "http://cran.us.r-project.org")
if(!require(data.table)) install.packages("data.table", repos = "http://cran.us.r-project.org")
if(!require(matrixStats)) install.packages("matrixStats", repos = "http://cran.us.r-project.org")
if(!require(reshape2)) install.packages("reshape2", repos = "http://cran.us.r-project.org")
if(!require(dplyr)) install.packages("reshape2", repos = "http://cran.us.r-project.org")
if(!require(randomForest)) install.packages("reshape2", repos = "http://cran.us.r-project.org")
if(!require(gam)) install.packages("reshape2", repos = "http://cran.us.r-project.org")


library(tidyverse)
library(caret)
library(data.table)
library(matrixStats)
library(reshape2)
library(dplyr)
library(randomForest)
library(gam)
```

Downloading the Data

The following code can download the data:

```r
url <- "https://github.com/Bitakhparsa/Diabetics_prediction/raw/main/diabetes2.csv"
download.file(url, destfile = "./data.csv", method="auto")
diabetes <- read.csv("data.csv")
```

Data Exploration and visualization

We will explore data by some simple codes.

```
#Exploring Data

glimpse(diabetes)
```

```
## Rows: 768
## Columns: 9
## $ Pregnancies              <int> 6, 1, 8, 1, 0, 5, 3, 10, 2, 8, 4, 10, 10, 1, ~
## $ Glucose                  <int> 148, 85, 183, 89, 137, 116, 78, 115, 197, 125~
## $ BloodPressure            <int> 72, 66, 64, 66, 40, 74, 50, 0, 70, 96, 92, 74~
## $ SkinThickness            <int> 35, 29, 0, 23, 35, 0, 32, 0, 45, 0, 0, 0, 0, ~
## $ Insulin                  <int> 0, 0, 0, 94, 168, 0, 88, 0, 543, 0, 0, 0, 0, ~
## $ BMI                      <dbl> 33.6, 26.6, 23.3, 28.1, 43.1, 25.6, 31.0, 35.~
## $ DiabetesPedigreeFunction <dbl> 0.627, 0.351, 0.672, 0.167, 2.288, 0.201, 0.2~
## $ Age                      <int> 50, 31, 32, 21, 33, 30, 26, 29, 53, 54, 30, 3~
## $ Outcome                  <int> 1, 0, 1, 0, 1, 0, 1, 0, 1, 1, 0, 1, 0, 1, 1, ~
```

```
head(diabetes)
```

```
##   Pregnancies Glucose BloodPressure SkinThickness Insulin  BMI
## 1           6     148            72            35       0 33.6
## 2           1      85            66            29       0 26.6
## 3           8     183            64             0       0 23.3
## 4           1      89            66            23      94 28.1
## 5           0     137            40            35     168 43.1
## 6           5     116            74             0       0 25.6
##   DiabetesPedigreeFunction Age Outcome
## 1                    0.627  50       1
## 2                    0.351  31       0
## 3                    0.672  32       1
## 4                    0.167  21       0
## 5                    2.288  33       1
## 6                    0.201  30       0
```

The code shows we have 768 rows and 9 columns.

The column descriptions have been provided by Kaggle as follows:

Pregnancies: Number of times pregnant

Glucose: Plasma glucose concentration a 2 hours in an oral glucose tolerance test

BloodPressure: Diastolic blood pressure (mm Hg)

SkinThickness: Triceps skinfold thickness (mm)

Insulin: 2-Hour serum insulin (mu U/ml)

BMI: Body mass index (weight in kg/(height in m)^2)

DiabetesPedigreeFunction: Diabetes pedigree function

Age: Age (years)

Outcome: Class variable (0 or 1), 1 shows that the patient has diabetes and 0 shows she is healthy.

Here we can check if we have any NA entries in the data.

```r
any(is.na(diabetes))
```

```
## [1] FALSE
```

Zero entry does not make sense for some of the columns like BMI, Insulin, SkinThickness, BloodPressure and Glucose, so we can estimate them with the median of the column.

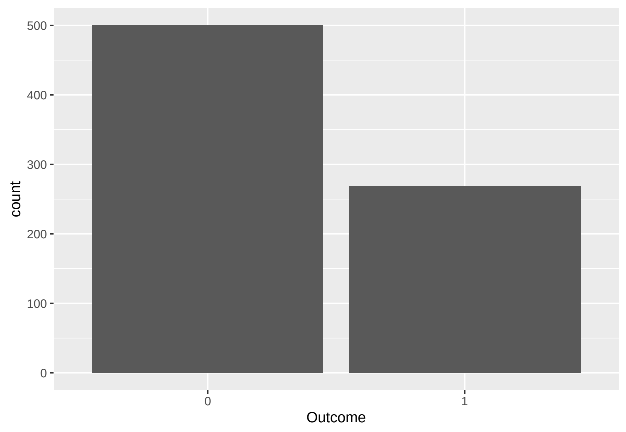
```r
diabetes_new <- diabetes %>%
  mutate(Outcome = factor(Outcome),
         BMI = ifelse((BMI==0),
                      median(BMI, na.rm = TRUE), BMI),
         Insulin = ifelse((Insulin==0),
                          median(Insulin, na.rm = TRUE), Insulin),
         SkinThickness = ifelse((SkinThickness==0),
                                median(SkinThickness, na.rm = TRUE), SkinThickness),
         BloodPressure = ifelse((BloodPressure==0),
                                median(BloodPressure, na.rm = TRUE), BloodPressure),
         Glucose = ifelse((Glucose==0),
                          median(Glucose, na.rm = TRUE), Glucose)
         )
```

The following codes show that 268 of the outcomes are 1, and 500 are 0.

```r
diabetes_new %>% group_by(Outcome) %>% summarise(n=n())
```

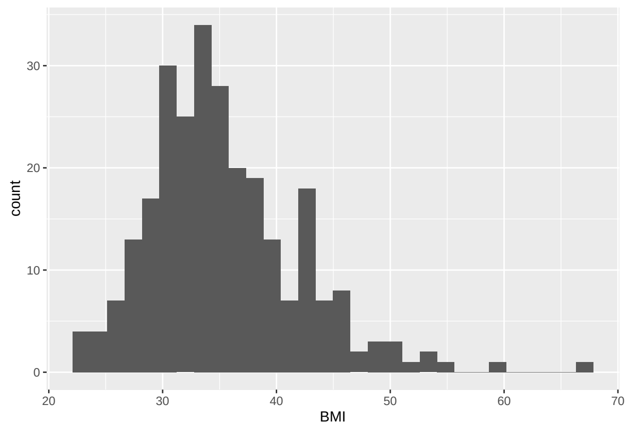
```
## # A tibble: 2 x 2
##   Outcome     n
##   <fct>   <int>
## 1 0         500
## 2 1         268
```

```r
diabetes_new %>% ggplot(aes(Outcome)) + geom_bar()
```



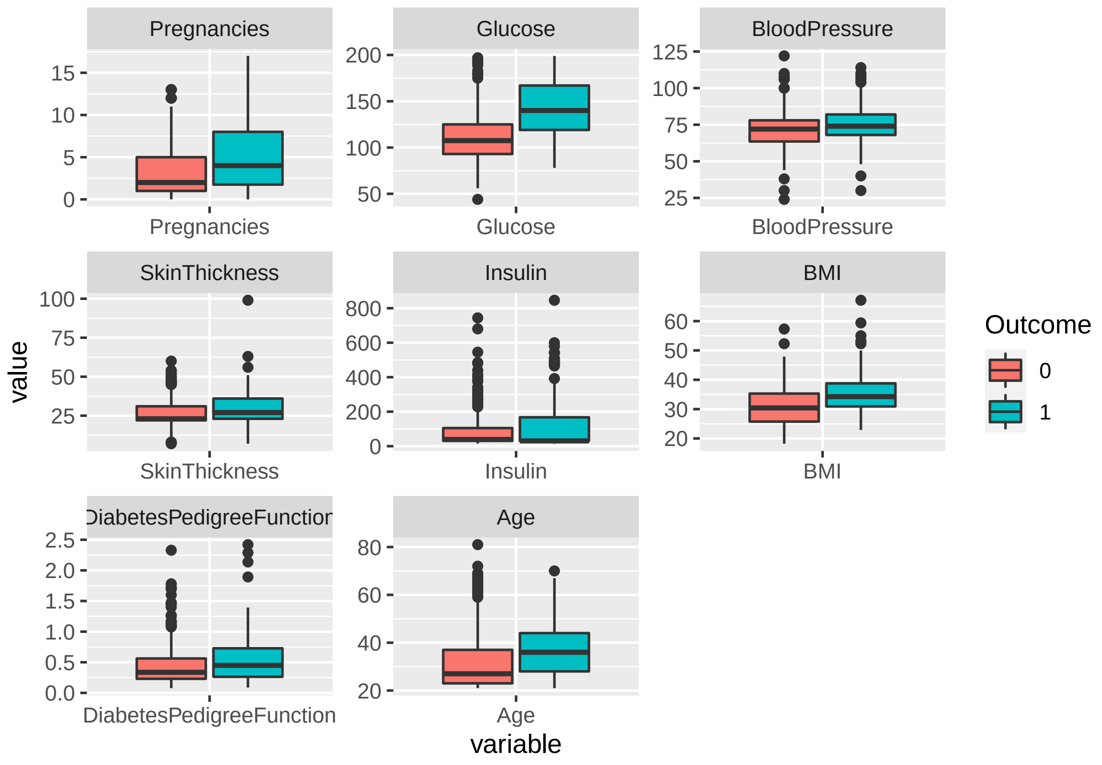
The below graph shows that more than 30 of the patient has a BMI of around 34.

```r
diabetes_new %>%
  filter(Outcome == "1") %>%
  ggplot(aes(BMI)) +
  geom_histogram(bins = 30)
```



Now we can see the distribution of each predictor stratified by the outcome.

```r
melt(diabetes_new , id="Outcome") %>%
                        ggplot(aes(x=variable, y=value)) +
                        geom_boxplot(aes(fill=Outcome))+
                        facet_wrap( ~ variable, scales="free")
```



The predictors save in a matrix and the outcome save in a vector:

```
y <- diabetes_new$Outcome
x <- diabetes_new %>% select(-Outcome)
class(y)
```

```
## [1] "factor"
```

```
class(x)
```

```
## [1] "data.frame"
```

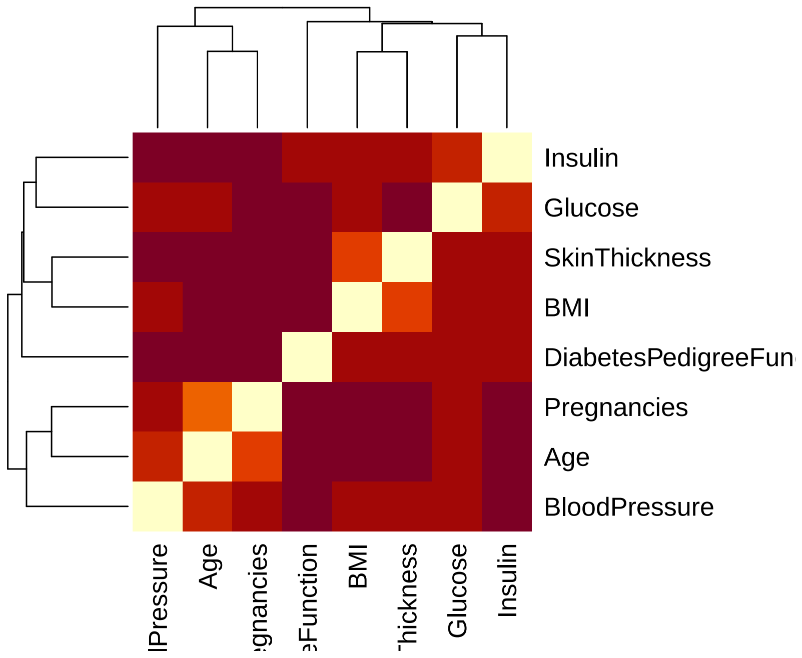
```
x<- as.matrix(x)
```

Now we can scale the matrix.

```
# Scaling the matrix x

x_centered <- sweep(x,2,colMeans(x))
x_scaled <- sweep(x_centered,2,colSds(x), FUN="/")
```

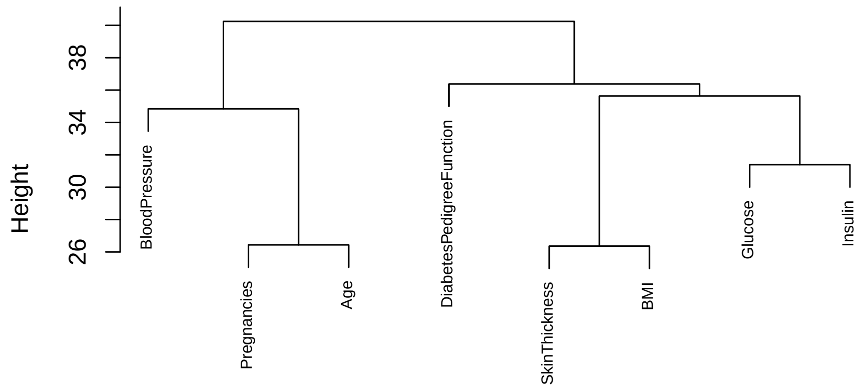
Heatmap is a strong tool in visualization; it shows clustering or pattern in the data. Here we use the heatmap function for discovering clusters.

```
#Heatmap

d <- dist(t(x_scaled))
heatmap(as.matrix(d))
```



Hierarchical cluster groups similar objects. We can find the distance between two things from the below graph. Find the first location from top to bottom that the object split into two different groups. For example, the distance between BloodPressure and Age is about 35.

```r
#Hierarchical clustering
h <- hclust(d)

plot(h, cex = 0.65, main = "", xlab = "")
```



hclust (*, "complete")

Also, we can split the objects into 3 groups as below.

```r
groups <- cutree(h,k=3)

split(names(groups), groups)
```

```
## $`1`
## [1] "Pregnancies"   "BloodPressure" "Age"
##
## $`2`
## [1] "Glucose"       "SkinThickness" "Insulin"       "BMI"
##
## $`3`
## [1] "DiabetesPedigreeFunction"
```

Predict the data

We can use many machine learning algorithms to predict the data; here, we want to use four of them to predict if a patient has diabetes or not.

Creating Data set

Data is divided into Diabetes_data and validation; we use validation data set only for the final model to avoid overtraining.

```
#Create Diabetes_data, validation, test set and train set

set.seed(1, sample.kind = "Rounding")
index <- createDataPartition(diabetes_new$Outcome, times = 1, p = 0.2, list = FALSE)

Diabetes_data <- diabetes_new %>% slice(-index)
validation <- diabetes_new %>% slice(index)
```

And then Diabetes_data is divided into train_set and test_set.

```
set.seed(1, sample.kind = "Rounding")
ind <- createDataPartition(Diabetes_data$Outcome, times = 1, p = 0.2, list = FALSE)

train_set <- Diabetes_data %>% slice(-ind)
test_set <- Diabetes_data %>% slice(ind)
```

The following code shows the number of rows for each data set:

```
nrow(Diabetes_data)
```

```
## [1] 614
```

```
nrow(validation)
```

```
## [1] 154
```

```
nrow(train_set)
```

```
## [1] 491
```

```
nrow(test_set)
```

```
## [1] 123
```

No.1 Logistic regression model

Logistic regression is a method used for modelling the probability of the data with a certain class or event existing like pass/fail, healthy/sick, win/lose. If we define the outcome Y as 1 for sick and 0 for healthy and X matrix of predictors, we have :

```
                Pr(Y=1 | X=x)
```

Since we have a binary outcome, we use binomial for family.

```
#Logistic regression model

set.seed(1, sample.kind = "Rounding")

train_glm <- train(Outcome ~ ., data = train_set, method = "glm", family="binomial")
glm_preds <- predict(train_glm, newdata = test_set)
cf_glm <- confusionMatrix(glm_preds,test_set$Outcome)
```

Now we can create a result table that shows accuracy, Sensitivity and Specificity of prediction for each method.

```
#Creating a result table
results <- tibble(method ="Logistic regression model",
                  accuracy =cf_glm$overall["Accuracy"],
                  Sensitivity =cf_glm$byClass[1] ,
                  Specificity = cf_glm$byClass[2])
```

No.2 K-nearest neighbors model

Another method used for modelling the data with a certain class, is K-nearest neighbours. In this method, we assume that similar things are near each other. This method is very simple and easy but slows for big data set, but our data is not big.

```
#K-nearest neighbors model

set.seed(1, sample.kind = "Rounding")

train_knn <- train(Outcome ~ ., method = "knn",
                   data=train_set,
                   tuneGrid = data.frame(k = seq(5, 90, 2)) )
train_knn$bestTune
```

```
##     k
## 24 51
```

```
knn_preds <- predict(train_knn, test_set)

cf_knn <- confusionMatrix(data=knn_preds, test_set$Outcome)
```

The best parameter that maximizes the estimated accuracy can find as below:

```r
train_knn$bestTune
```

```
##     k
## 24 51
```

The result is added to the table.

```r
#adding result in the table
results <- bind_rows(results,data_frame(method="K-nearest neighbors model",
                                       accuracy =cf_knn$overall["Accuracy"],
                                       Sensitivity =cf_knn$byClass[1] ,
                                       Specificity = cf_knn$byClass[2] ))
results %>% knitr::kable()
```

method	accuracy	Sensitivity	Specificity
Logistic regression model	0.7398374	0.8625	0.5116279
K-nearest neighbors model	0.7073171	0.8625	0.4186047

No.3 Random forest model

Random forests model goal is to improve accuracy by averaging multiple decision trees. Each tree in the random forest shows a class prediction, and the class with the most votes becomes the model's prediction.

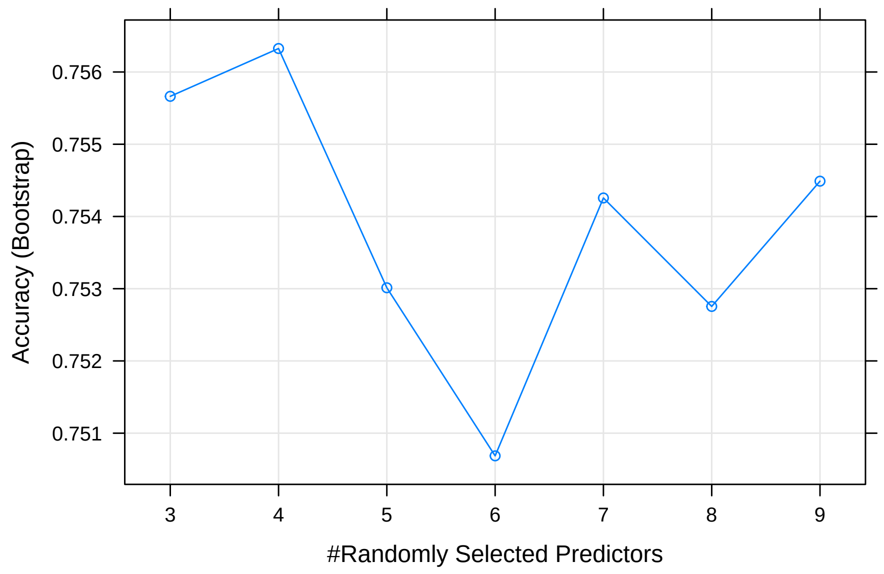
```r
#Random forest model
set.seed(1, sample.kind = "Rounding")
train_rf <- train(Outcome ~ ., data=train_set,
                  method="rf",
                  importance = TRUE,
                  nodesize = 1,
                  tuneGrid = data.frame(mtry=seq(3, 9, 1)))

rf_preds <- predict(train_rf, test_set)

cf_rf <- confusionMatrix(data=rf_preds, test_set$Outcome)
```

We can use the plot to see if the Random forest has converged or we need more trees, and the plot shows that it converged.

```r
train_rf$bestTune
```

```
##   mtry
## 2    4
```

```r
plot(train_rf)
```



The below code shows the importance of the various variables, and the most important feature is Glucose.

```r
varImp(train_rf)
```

```
## rf variable importance
##
##                          Importance
## Glucose                      100.00
## Age                           41.27
## BMI                           39.14
## Pregnancies                   32.51
## DiabetesPedigreeFunction      20.03
## SkinThickness                 13.63
## Insulin                       13.14
## BloodPressure                  0.00
```

The result is added to the table.

```r
#adding result in the table
results <- bind_rows(results,data_frame(method="Random forest model",
                                        accuracy =cf_rf$overall["Accuracy"],
                                        Sensitivity =cf_rf$byClass[1] ,
                                        Specificity = cf_rf$byClass[2] ) )
results %>% knitr::kable()
```

method	accuracy	Sensitivity	Specificity
Logistic regression model	0.7398374	0.8625	0.5116279
K-nearest neighbors model	0.7073171	0.8625	0.4186047
Random forest model	0.7317073	0.8250	0.5581395

Selecting final model

Overall accuracy is a good measure to evaluate a method, but it is not enough; depending on the context of the data, some of the errors are more dangerous or costly.
Sensitivity and specificity define as below:

sensitivity = TP/(TP+FN)
specificity = TN/(TN+FP)

	Actual positive	Actual negative
Predicted positive	True positive(TP)	False positive(FP)
Predicted negative	False negative(FN)	True negative(TN)

Sensitivity: the ability of a test to correctly identify patients with a disease.
Specificity: the ability of a test to correctly identify people without the disease.
In this case, Sensitivity is more important for us; It is clear that failing to predict a sick person (true negative), is worse than predicting a healthy person as a sick person (false positive). The result shows that Logistic regression is the best model, since the accuracy and sensitivity is higher than the other models.

Testing the final model

Now selected model(Logistic regression) is tested on the validation set.

```
#Final model
glm_preds_final <- predict(train_glm, newdata= validation)

cf_glm_final <- confusionMatrix(data=glm_preds_final,validation$Outcome)


#adding result in the table
results <- bind_rows(results,data_frame(method="Final model",
                                        accuracy =cf_glm_final$overall["Accuracy"],
                                       Sensitivity =cf_glm_final$byClass[1] ,
                                       Specificity = cf_glm_final$byClass[2] ) )
results %>% knitr::kable()
```

method	accuracy	Sensitivity	Specificity
Logistic regression model	0.7398374	0.8625	0.5116279
K-nearest neighbors model	0.7073171	0.8625	0.4186047
Random forest model	0.7317073	0.8250	0.5581395
Final model	0.7792208	0.8900	0.5740741

Conclusion

The result table shows that the logistic regression model is more accurate than the other models. Since the sensitivity is about 0.89 %, predicting sick people can be more accurate. The model specificity is not good but better than the others, so it is inaccurate to predict if someone is healthy.