Diabetes Project Report

Manuel Alexis Mena Nieves

6/23/2020

1. Introduction

This project report is about creating a model prediction system for the HarvardX Data Science professional certificate program, using the Indians Pima Diabetes Dataset, originally from the National Institute of Diabetes and Digestive and Kidney Diseases.

This dataset consists of eight medical predictor variables and one target variable, which shows if the patient has diabetes or not. Predictor variables includes the number of pregnancies the patient has had, their BMI, insulin level, age, skin thickness, glucose level, blood pressure and computed value called Diabetes Pedrigree Function.

The goal of the project is to diagnostically predict whether or not a patient has diabetes, based on certain diagnostic measurements included in the dataset using all the tools shown throughout the courses in this series. To accomplish this an exploratory analysis was done, in order to understand the data and summarize their main characteristics with tables and visual methods. After this, a machine learning model and an ensemble model was created to predict whether or not the patients in the dataset have diabetes.

2. Getting the data

The following code will be used to download the dataset. We begin by loading the tidyverse, caret, skimr and some useful machine learning libraries:

```r
# Load the libraries
repo <- "http://cran.us.r-project.org"
if(!require(tidyverse)) install.packages("tidyverse", repos = repo)
if(!require(caret)) install.packages("caret", repos = repo)
if(!require(skimr)) install.packages("skimr", repos = repo)
if(!require(rpart)) install.packages("rpart", repos = repo)
if(!require(randomForest)) install.packages("randomForest", repos = repo)
if(!require(gbm)) install.packages("gbm", repos = repo)
if(!require(kernlab)) install.packages("kernlab", repos = repo)
if(!require(gam)) install.packages("gam", repos = repo)
```

The dataset was uploaded into GitHub, thus we can access the data.

```r
# Read the file
url <- paste0("https://raw.githubusercontent.com/alexismenanieves/",
              "Diabetes_Project/master/dataset.txt")
dataset <- read.csv(url)
```

In order to analyze the dataset, we see the dimensions, variable types and a summary.

```r
# A first view of the data, dimensions and variables
dim(dataset)
```

```
## [1] 768   9
```

```r
as_tibble(dataset)
```

```
## # A tibble: 768 x 9
##    Pregnancies Glucose BloodPressure SkinThickness Insulin   BMI
##          <int>   <int>         <int>         <int>   <int> <dbl>
##  1           6     148            72            NA       0  NA
##  2           1      85            NA            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
##  7           3      NA            50            32      88  31
##  8          10     115             0             0       0  35.3
##  9           2     197            70            45     543  30.5
## 10           8     125            96             0       0   0
## # ... with 758 more rows, and 3 more variables: DiabetesPedigreeFunction <dbl>,
## #   Age <int>, Outcome <int>
```

```r
summary(dataset)
```

```
##   Pregnancies        Glucose      BloodPressure    SkinThickness
##  Min.   : 0.000   Min.   :  0.0   Min.   :  0.00   Min.   : 0.00
##  1st Qu.: 1.000   1st Qu.: 99.0   1st Qu.: 62.00   1st Qu.: 0.00
##  Median : 3.000   Median :117.0   Median : 72.00   Median :23.00
##  Mean   : 3.845   Mean   :120.9   Mean   : 69.11   Mean   :20.52
##  3rd Qu.: 6.000   3rd Qu.:140.0   3rd Qu.: 80.00   3rd Qu.:32.00
##  Max.   :17.000   Max.   :199.0   Max.   :122.00   Max.   :99.00
##                   NA's   :2       NA's   :1        NA's   :1
##     Insulin           BMI        DiabetesPedigreeFunction      Age
##  Min.   :  0.0   Min.   : 0.00   Min.   :0.0780           Min.   :21.00
##  1st Qu.:  0.0   1st Qu.:27.30   1st Qu.:0.2437           1st Qu.:24.00
##  Median : 30.5   Median :32.00   Median :0.3725           Median :29.00
##  Mean   : 79.8   Mean   :31.99   Mean   :0.4719           Mean   :33.24
##  3rd Qu.:127.2   3rd Qu.:36.60   3rd Qu.:0.6262           3rd Qu.:41.00
##  Max.   :846.0   Max.   :67.10   Max.   :2.4200           Max.   :81.00
##                  NA's   :1
##     Outcome
##  Min.   :0.000
##  1st Qu.:0.000
##  Median :0.000
##  Mean   :0.349
##  3rd Qu.:1.000
##  Max.   :1.000
##
```

We can see that the dataset has 768 observations and 9 variables. When we explore the dataset we see that all variables are numbers, that there are some NAs and that the last column is the outcome. In the summary we see that some variables have a zero value, like glucose, blood pressure, insulin, skin thickness or BMI. The median age of the patients is 33, and the median number of pregnancies is 3.

We can see through a table the number of subjects with diabetes or not:

```r
dataset %>% group_by(Outcome) %>%
  summarise(count = n()) %>% mutate(freq = count/sum(count))
```

```
## `summarise()` ungrouping output (override with `.groups` argument)
```

```
## # A tibble: 2 x 3
##   Outcome count  freq
##     <int> <int> <dbl>
## 1       0   500 0.651
## 2       1   268 0.349
```

What draws our attention is that the outcome is imbalanced, because only 34.8% of patients have diabetes.

Since the outcome is an integer let’s transform it into a factor and rename it as “Diabetes”.

```r
# Let's apply some changes on the outcome name and encoding
dataset$Outcome <- as.factor(ifelse(dataset$Outcome == 1,"Yes","No"))
names(dataset)[9]<- "Diabetes"
```

Let’s divide our dataset into a train set and a test set. The test set will be used as an unseen data to test the prediction power of our model.

```r
# Let's divide the dataset into a train and test set
set.seed(1979)
tt_index <- createDataPartition(dataset$Age, times = 1, p = 0.9, list = FALSE)
train_set <- dataset[tt_index,]
test_set <- dataset[-tt_index,]
```

3. Exploratory Analysis

We can display the structure of our train set.

```r
# See how many observations and variables are available
str(train_set)
```

```
## 'data.frame':    693 obs. of  9 variables:
##  $ Pregnancies             : int  6 1 8 1 0 5 3 10 2 8 ...
##  $ Glucose                 : int  148 85 183 89 137 116 NA 115 197 125 ...
##  $ BloodPressure           : int  72 NA 64 66 40 74 50 0 70 96 ...
##  $ SkinThickness           : int  NA 29 0 23 35 0 32 0 45 0 ...
##  $ Insulin                 : int  0 0 0 94 168 0 88 0 543 0 ...
##  $ BMI                     : num  NA 26.6 23.3 28.1 43.1 25.6 31 35.3 30.5 0 ...
##  $ DiabetesPedigreeFunction: num  0.627 0.351 0.672 0.167 2.288 ...
##  $ Age                     : int  50 31 32 21 33 30 26 29 53 54 ...
##  $ Diabetes                : Factor w/ 2 levels "No","Yes": 2 1 2 1 2 1 2 1 2 2 ...
```

```r
# Glimpse of mean, median and NA's
summary(train_set)
```

```
##   Pregnancies        Glucose      BloodPressure    SkinThickness
##  Min.   : 0.000   Min.   :  0.0   Min.   :  0.00   Min.   : 0.00
##  1st Qu.: 1.000   1st Qu.: 99.0   1st Qu.: 62.00   1st Qu.: 0.00
##  Median : 3.000   Median :117.0   Median : 72.00   Median :23.00
##  Mean   : 3.857   Mean   :120.4   Mean   : 69.18   Mean   :20.23
##  3rd Qu.: 6.000   3rd Qu.:139.0   3rd Qu.: 80.00   3rd Qu.:32.00
##  Max.   :17.000   Max.   :198.0   Max.   :122.00   Max.   :63.00
##                   NA's   :2       NA's   :1        NA's   :1
##     Insulin            BMI        DiabetesPedigreeFunction      Age
##  Min.   :  0.00   Min.   : 0.00   Min.   :0.0780           Min.   :21.00
##  1st Qu.:  0.00   1st Qu.:27.10   1st Qu.:0.2400           1st Qu.:24.00
##  Median : 36.00   Median :32.00   Median :0.3660           Median :29.00
##  Mean   : 79.29   Mean   :31.82   Mean   :0.4686           Mean   :33.17
##  3rd Qu.:128.00   3rd Qu.:36.33   3rd Qu.:0.6260           3rd Qu.:41.00
##  Max.   :846.00   Max.   :67.10   Max.   :2.4200           Max.   :81.00
##                   NA's   :1
##  Diabetes
##  No :457
##  Yes:236
##
##
##
##
##
```

```r
# Count how many zeros are in each variable
colSums(train_set[,-9] == 0, na.rm = TRUE)
```

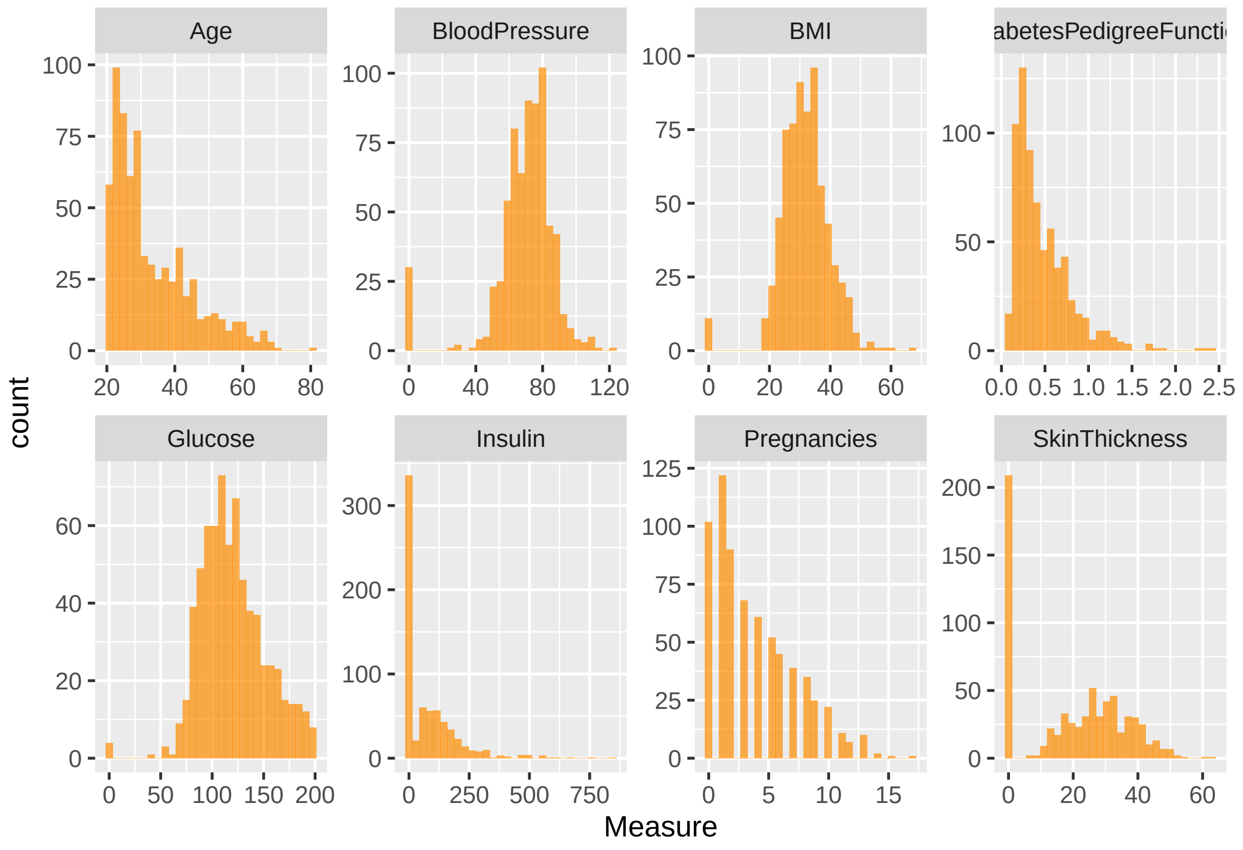
```
##              Pregnancies                  Glucose            BloodPressure
##                      102                        4                       30
##            SkinThickness                  Insulin                      BMI
##                      209                      335                       11
## DiabetesPedigreeFunction                      Age
##                        0                        0
```

An histogram plot for each predictor can show us the data distribution and the outliers.

```r
train_set %>% gather(key = "Variable", value = "Measure", -Diabetes) %>%
  ggplot(aes(Measure)) + geom_histogram(alpha = 0.7, fill = "darkorange") +
  facet_wrap(~Variable, ncol = 4, scales = "free")
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```

```
## Warning: Removed 5 rows containing non-finite values (stat_bin).
```



We can see that in blood pressure, BMI, glucose, insulin and skin thickness the zero values are outliers because of the distribution shape. We can not say the same for the Diabetes Pedigree Function. The decision taken was to convert the zero values into NAs.

```
# Convert zeros to NA
train_set[,c(2:6)] <- apply(train_set[,c(2:6)], 2, function(x) {ifelse(x==0, NA, x)} )
```

Let's see how correlated are the variables using the "cor" function:

```
train_set[,-9] %>%
  rename(DPF = DiabetesPedigreeFunction,
         SkThick = SkinThickness,
         BloodPress = BloodPressure) %>%
  cor(use = "complete.obs") %>% format(digits = 2)
```

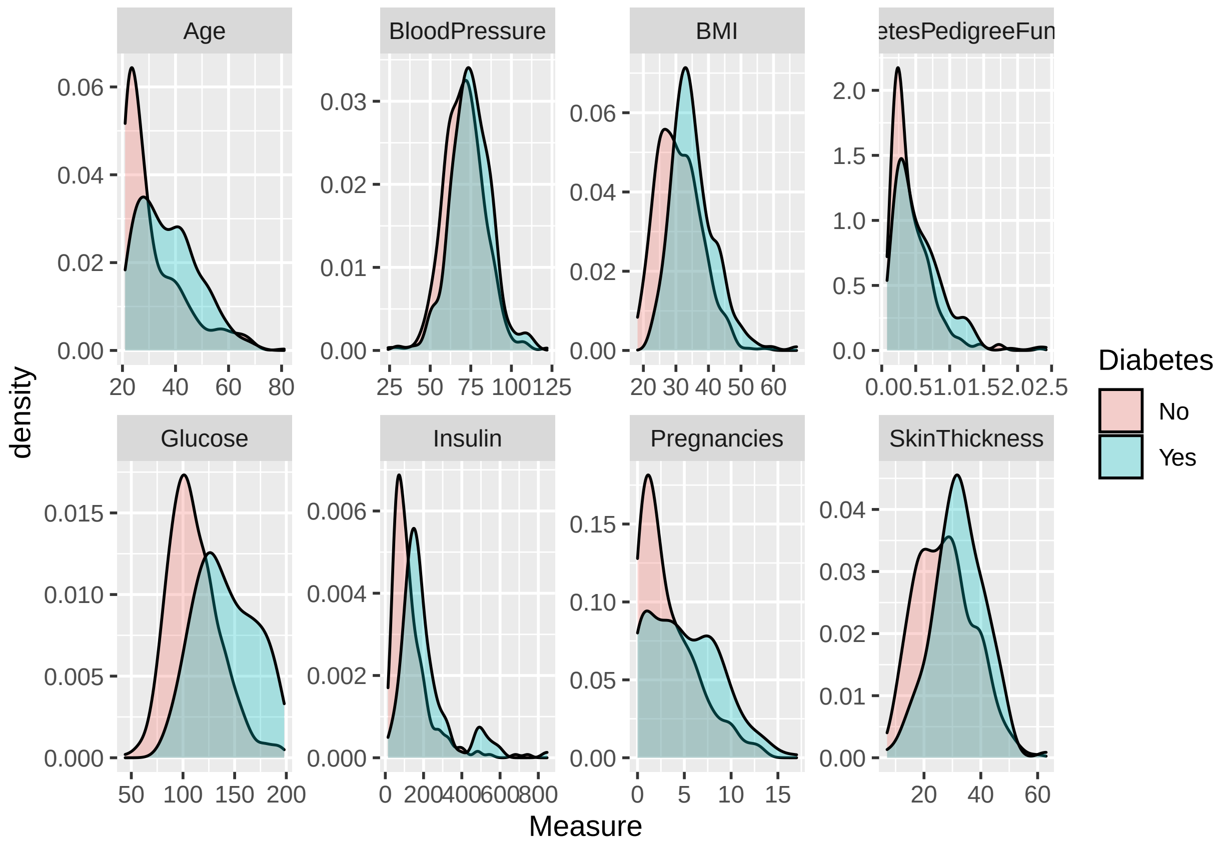
```
##               Pregnancies Glucose  BloodPress SkThick  Insulin  BMI      DPF
## Pregnancies " 1.000"     " 0.181" " 0.205"   " 0.094" " 0.065" "-0.043" " 0.017"
## Glucose     " 0.181"     " 1.000" " 0.184"   " 0.177" " 0.574" " 0.181" " 0.135"
## BloodPress  " 0.205"     " 0.184" " 1.000"   " 0.240" " 0.079" " 0.307" "-0.029"
## SkThick     " 0.094"     " 0.177" " 0.240"   " 1.000" " 0.158" " 0.655" " 0.152"
## Insulin     " 0.065"     " 0.574" " 0.079"   " 0.158" " 1.000" " 0.211" " 0.121"
## BMI         "-0.043"     " 0.181" " 0.307"   " 0.655" " 0.211" " 1.000" " 0.166"
## DPF         " 0.017"     " 0.135" "-0.029"   " 0.152" " 0.121" " 0.166" " 1.000"
## Age         " 0.677"     " 0.326" " 0.283"   " 0.177" " 0.212" " 0.059" " 0.104"
##               Age
## Pregnancies " 0.677"
## Glucose     " 0.326"
## BloodPress  " 0.283"
## SkThick     " 0.177"
## Insulin     " 0.212"
## BMI         " 0.059"
## DPF         " 0.104"
## Age         " 1.000"
```

We can see that pregnancies and age are correlated which is expected.

Now we can explore further, by using the density plot for each variable and stratifying by the outcome.

```
train_set %>% gather(key = "Variable", value = "Measure", -Diabetes) %>%
  ggplot(aes(Measure, fill = Diabetes)) + geom_density(alpha = 0.3) +
  facet_wrap(~Variable,ncol = 4, scales = "free")
```

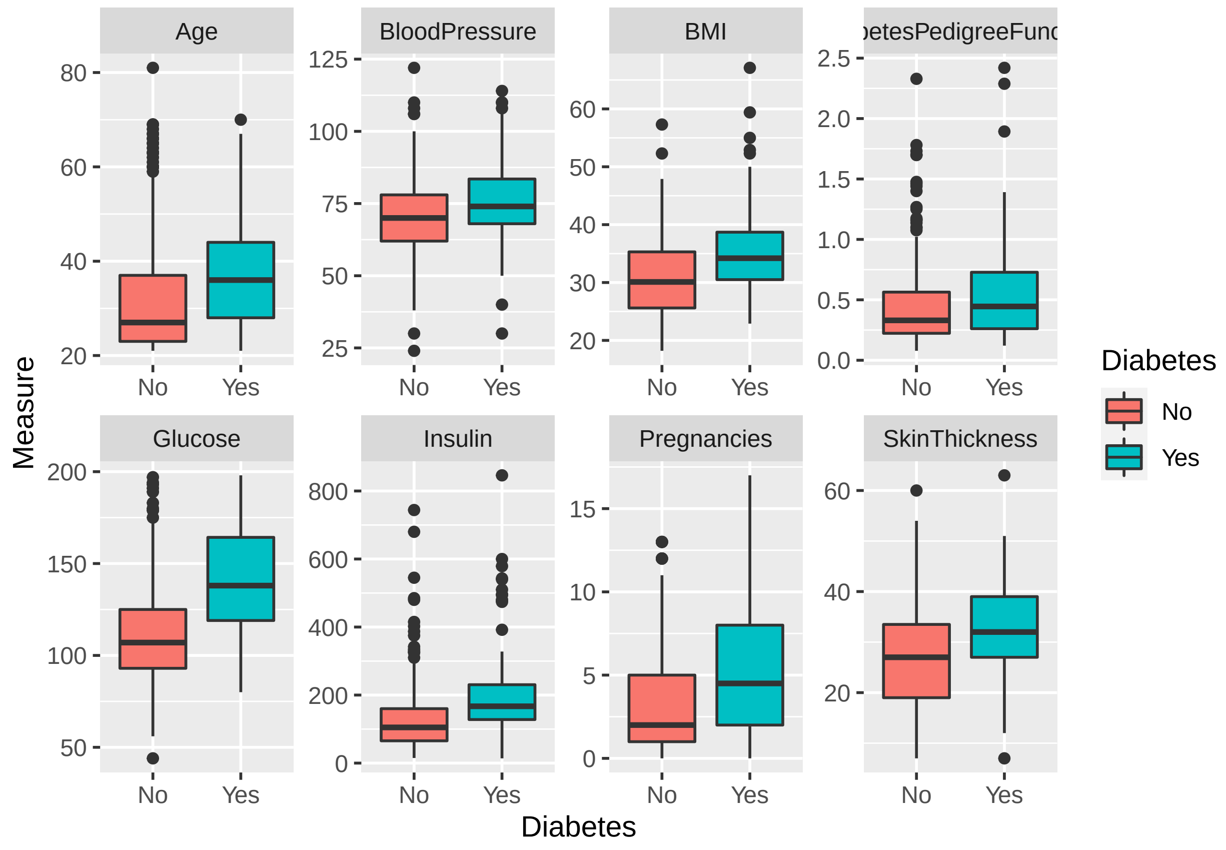
```
## Warning: Removed 594 rows containing non-finite values (stat_density).
```



We see somehow a mean shift between diabetes and no diabetes in age, BMI, glucose, insulin and skin thickness predictors. Let's see a similar plot but using a box-plot.

```
train_set %>% gather(key = "Variable", value = "Measure", -Diabetes) %>%
  ggplot(aes(Diabetes, Measure, fill = Diabetes)) + geom_boxplot() +
  facet_wrap(~Variable,ncol = 4, scales = "free")
```

```
## Warning: Removed 594 rows containing non-finite values (stat_boxplot).
```



We can see clearly here that glucose has a better mean shift than insulin, so that predictor cold be an important predictor for diabetes.

4. Preprocessing the data

Before creating a model, we have to preprocess our data. Since we want to tune our model we split our train dataset into a new train dataset and a validation dataset.

```
# Set seed
set.seed(1979)
# Divide the train set into a new train set and a validation set
tv_index <- createDataPartition(train_set$Age, times = 1, p = 0.8, list = FALSE)
validation_set <- train_set[-tv_index,]
train_set <- train_set[tv_index,]
```

We have to replace the missing values, so we impute them with the median value. We can normalize values so it ranges between 0 and 1 by using range. This method in caret is called preProcess.

```
preProcess_data <- preProcess(train_set, method = c("medianImpute","range"))
train_set <- predict(preProcess_data, newdata = train_set)
```

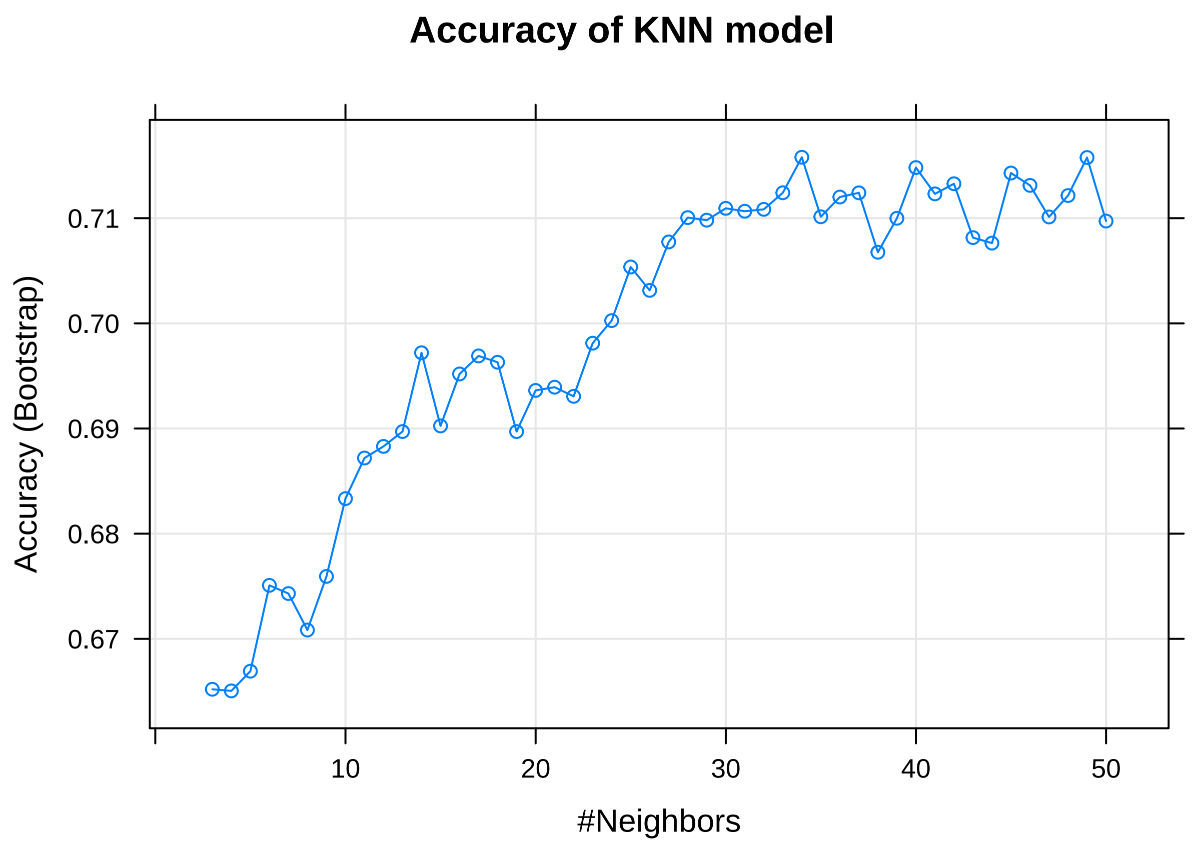
5. Creating a simple model

Our first model can be the KNN, it's simple to understand and we can tune it faster. Because the class imbalance in the dataset, we can use the parameter "sampling" and as our dataset is not so big, we use the argument "up". We will tune our model with neighbors from 3 to 50.

```
# Train knn on train set for a first glimpse of accuracy
model_KNN <- train(Diabetes ~.,
                   data = train_set,
                   trControl = trainControl(sampling = "up"),
                   tuneGrid = data.frame(k = seq(3,50,1)),
                   method = "knn")
```

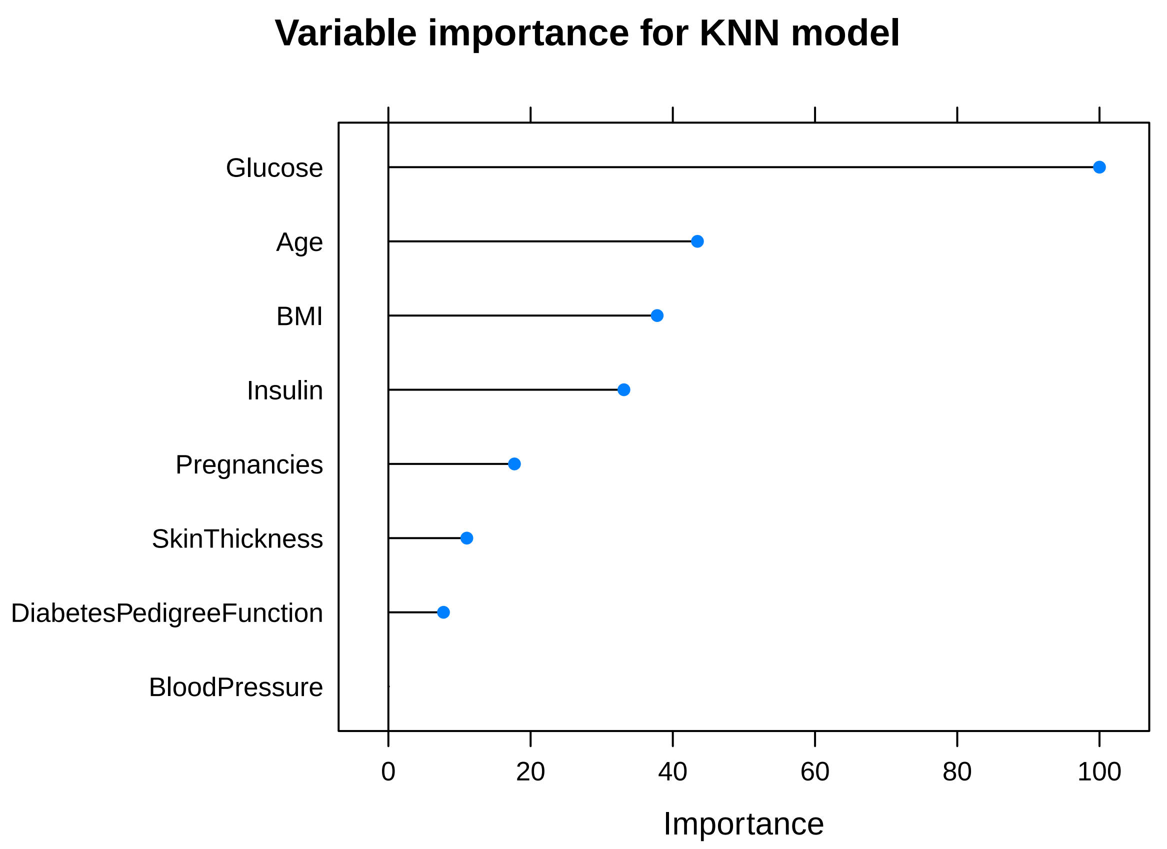
Let's plot the accuracy for each neighbor k in our model:

```
# Plot accuracy
plot(model_KNN, main = "Accuracy of KNN model")
```



The best accuracy obtained is over 0.71. Let's plot the variable importance for this model:

```
# Plot variable importance
plot(varImp(model_KNN), main = "Variable importance for KNN model")
```



We can see that glucose, age, BMI and insulin are the top four variables in our model. We can calculate the max accuracy and k in the model:

```
model_KNN$bestTune
```

```
##     k
## 32 34
```

```
max(model_KNN$results$Accuracy)
```

```
## [1] 0.7157949
```

5. Ensemble Model

We can create an ensemble model using multiple algorithms and keeping the best methods. The models selected are a non-parametric method (KNN), a classification tree (rpart), a random forest (rf), a generative additive model (gamLoess), a gradient boosting method (gbm) and a support vector machine (svmLinear). For each model we define a tuning parameter, except for gamLoess.

```
models <- c("knn","rpart","rf","gamLoess","gbm","svmLinear")
knn_tg = data.frame(k = seq(3,50,1))
rpart_tg = data.frame(cp = seq(0,0.1,len = 50))
rf_tg = data.frame(mtry = seq(1,8,1))
gbm_tg = expand.grid(n.trees = c(100, 1000, 10000),
                     interaction.depth = c(1,3,5),
                     shrinkage = 0.001,
                     n.minobsinnode = 1)
svmLinear_tg = data.frame(C = c(0.01, 0.05, 0.1, seq(0.25, 2, 0.25)))
```

We use the lapply function to train each model, sampling up our data to address the imbalance and in the case of the gbm method we use the argument verbose as false to avoid the calculation text output given in each iteration.

```
suppressMessages(fits <- lapply(models, function(model){
  print(model)
  if(model == "gbm"){
    train(Diabetes ~ .,
          data = train_set,
          trControl = trainControl(sampling = "up"),
          tuneGrid = gbm_tg,
          method = model,
          verbose = FALSE)
  } else {
    train(Diabetes ~ .,
          data = train_set,
          trControl = trainControl(sampling = "up"),
          tuneGrid = switch(model,
                            "knn" = knn_tg,
                            "rpart" = rpart_tg,
                            "rf" = rf_tg,
                            "gamLoess" = NULL,
                            "svmLinear" = svmLinear_tg),
        method = model)
  }
}))
```

```
## [1] "knn"
## [1] "rpart"
## [1] "rf"
## [1] "gamLoess"
## [1] "gbm"
## [1] "svmLinear"
```

Now we have a list for each model, so we give them their corresponding name.

```
names(fits) <- models
```

We use sapply to create a matrix of predictions for the validation test. For each column we will have a “Yes” or “No” outcome.

```
pred <- sapply(fits, function(object)
  predict(object, newdata = validation_set))
dim(pred)
```

```
## [1] 77  6
```

Now we can compute accuracy for each model on the validation set:

```
acc <- colMeans(pred == validation_set$Diabetes)
```

```
## Warning in `==.default`(pred, validation_set$Diabetes): longer object length is
## not a multiple of shorter object length

## Warning in is.na(e1) | is.na(e2): longer object length is not a multiple of
## shorter object length
```

```
acc
```

```
##       knn     rpart        rf  gamLoess       gbm svmLinear
## 0.3506494 0.3246753 0.4025974 0.2857143 0.5064935 0.3116883
```

We can proceed to calculate the mean accuracy. It turns out to be 0.758, which is an improvement.

```
mean(acc)
```

```
## [1] 0.3636364
```

We build an ensemble prediction by majority vote, using the validation set.

```
votes <- rowMeans(pred == "Yes")
y_hat <- ifelse(votes > 0.5, "Yes", "No")
mean(y_hat == validation_set$Diabetes)
```

```
## Warning in `==.default`(y_hat, validation_set$Diabetes): longer object length is
## not a multiple of shorter object length

## Warning in is.na(e1) | is.na(e2): longer object length is not a multiple of
## shorter object length

## [1] 0.3405797
```

We now remove the methods that do not perform well and then we re do the ensemble. The index selected will be every method accuracy that is better than the mean accuracy calculated.

```
ind <- acc >= mean(acc)
best_votes <- rowMeans(pred[,ind]=="Yes")
best_y_hat <- ifelse(best_votes > 0.5, "Yes", "No")
mean(best_y_hat == validation_set$Diabetes)
```

```
## Warning in `==.default`(best_y_hat, validation_set$Diabetes): longer object
## length is not a multiple of shorter object length

## Warning in is.na(e1) | is.na(e2): longer object length is not a multiple of
## shorter object length

## [1] 0.5
```

We can see that the best accuracy of this ensemble model is 0.782, which is better than our first attempt with the KNN algorithm. The methods who performed well were "rf", "gamLoess", "gbm" and "svmLinear".

5. Evaluation on the test set

```r
test_set <- predict(preProcess_data, newdata = test_set)
test_pred <- sapply(fits, function(object)
  predict(object, newdata = test_set))
test_votes <- rowMeans(test_pred[,ind]=="Yes")
test_y_hat <- ifelse(test_votes > 0.5, "Yes", "No")
mean(test_y_hat == test_set$Diabetes)
```

```
## [1] 0.6933333
```

6. Conclusions

- It is important to make an exploratory data analysis, we could detect the high quantity of zero values, the outliers and the imbalance in our data.
- The plots were useful to understand the data distribution and supported the decision of converting the zero values into NAs. We also could see that some variables had more importance than others, like glucose, age, BMI and Insulin.
- In our first approach, using the KNN algorithm we could get an accuracy of 0.702 on the validation set, which is our starting point to improve the accuracy in subsequent models.
- Using the ensemble model we could get an accuracy of 0.782, which is an improvement over the first model we tried. The algorithms selected for
- On the test set, considering it an unseen data we could get an accuracy 0.773.