HarvardX : Choose Your Own!

Pima Indian Women Diabetes Classification

Jordan Porcu

Contents

Introduction

Overview

This document is done within the framework of the *HarvardX PH125.9x Data Science : Capstone* certification. This project is a chosen subject : Diabetes Binary Classification.

The data set we are going to use can be found at https://www.kaggle.com/datasets/akshaydattatraykhare/diabetes-dataset.

In this project, as the subject implies, we will build a Machine learning model to predict if a woman has diabetes, based on physiological variables.

Since we face a binary classification problem, we will use two metrics to evaluate the models performance : **accuracy** & **f1-score**.

The Dataset

The data set is given by the National Institute of Diabetes and Digestive and Kidney Diseases. It displays physiological attributes of 21 years old and above women from Pima Indian community.

Variables are :

- **Pregnancies** : times a woman has been pregnant.
- **Glucose** : glucose level in blood.
- **BloodPressure** : blood pressure measurement.
- **SkinThickness** : thickness of their skin.
- **Insulin** : insulin level in blood.
- **BMI** : body mass index.
- **DiabetesPedigreeFunction** : diabetes percentage.
- **Age** : age of the woman
- **Outcome** : 1 if the woman has diabetes, 0 otherwise.

The first eight variables will be used to predict the ninth one.

The data set is available here : **diabetes**

Analysis

Once we downloaded the data set, we import it and we compute the first 5 lines :

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

Now that we have a view on what the data set looks like, we compute some useful statistics :

- data set dimensions

Rows	Columns
768	9

- variables statistics

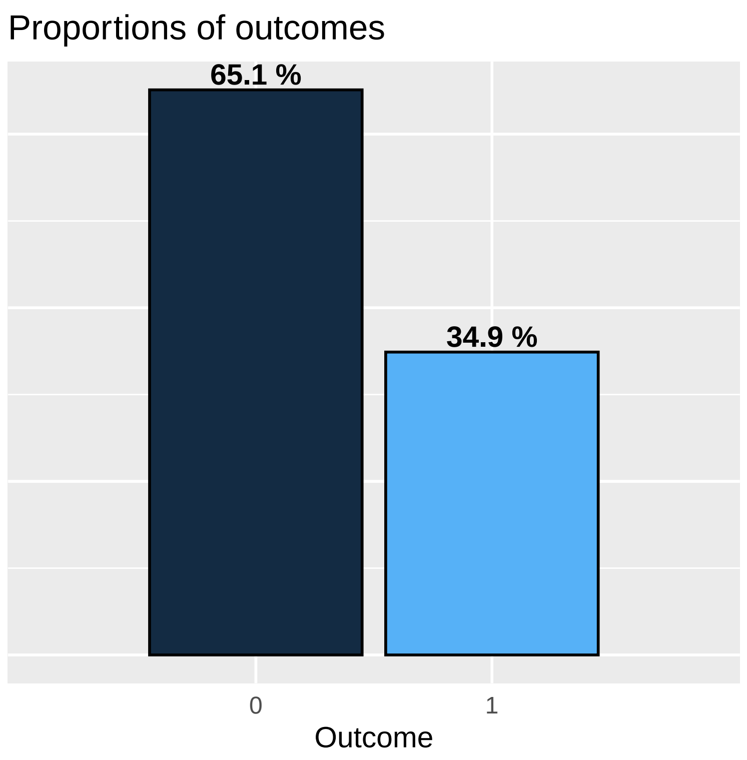
Pregnancies	Glucose	BloodPressure	SkinThickness
Min. : 0.000	Min. : 0.0	Min. : 0.00	Min. : 0.00
1st Qu.: 1.000	1st Qu.: 99.0	1st Qu.: 62.00	1st Qu.: 0.00
Median : 3.000	Median :117.0	Median : 72.00	Median :23.00
Mean : 3.845	Mean :120.9	Mean : 69.11	Mean :20.54
3rd Qu.: 6.000	3rd Qu.:140.2	3rd Qu.: 80.00	3rd Qu.:32.00
Max. :17.000	Max. :199.0	Max. :122.00	Max. :99.00

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

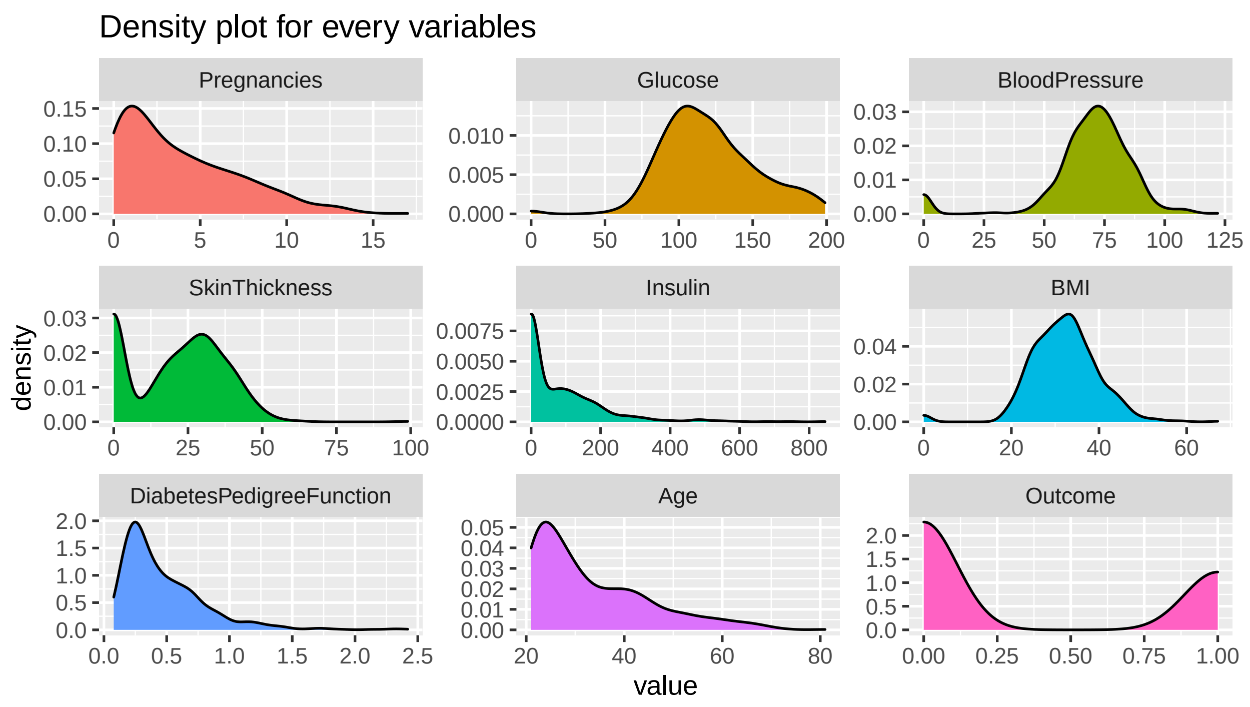
- structure of the data set

```
## 'data.frame':    768 obs. of  9 variables:
##  $ Pregnancies             : num  6 1 8 1 0 5 3 10 2 8 ...
##  $ Glucose                 : num  148 85 183 89 137 116 78 115 197 125 ...
##  $ BloodPressure           : num  72 66 64 66 40 74 50 0 70 96 ...
##  $ SkinThickness           : num  35 29 0 23 35 0 32 0 45 0 ...
##  $ Insulin                 : num  0 0 0 94 168 0 88 0 543 0 ...
##  $ BMI                     : num  33.6 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                     : num  50 31 32 21 33 30 26 29 53 54 ...
##  $ Outcome                 : num  1 0 1 0 1 0 1 0 1 1 ...
```

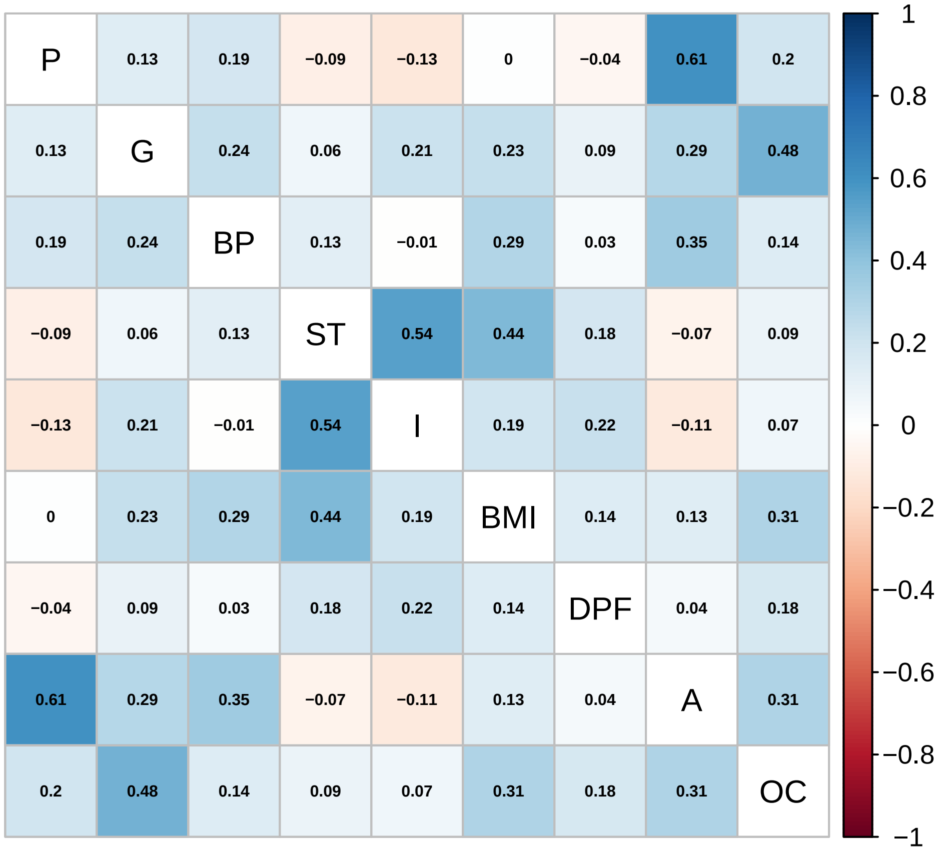
- proportion of Outcome values



- variables density plot



- correlation between variables



P	0.13	0.19	−0.09	−0.13	0	−0.04	0.61	0.2
0.13	G	0.24	0.06	0.21	0.23	0.09	0.29	0.48
0.19	0.24	BP	0.13	−0.01	0.29	0.03	0.35	0.14
−0.09	0.06	0.13	ST	0.54	0.44	0.18	−0.07	0.09
−0.13	0.21	−0.01	0.54	I	0.19	0.22	−0.11	0.07
0	0.23	0.29	0.44	0.19	BMI	0.14	0.13	0.31
−0.04	0.09	0.03	0.18	0.22	0.14	DPF	0.04	0.18
0.61	0.29	0.35	−0.07	−0.11	0.13	0.04	A	0.31
0.2	0.48	0.14	0.09	0.07	0.31	0.18	0.31	OC

- any NaN in the set ?

```
sum(is.na(diabetes))
```

```
## [1] 0
```

From these elements, we can extract some information :

- All variables are numerical, and the only issue we will (easily) deal with is the fact that "Outcome" is numerical instead of factor.
- Data set is clean (0 NaN).
- Approximately, a third of the patients are sick.
- It seems that there is no aberration in variables distribution.
- "Glucose,"BloodPressure" and "BMI" are close to a normal distribution (extreme values of "Glucose" tend to contradict this information).
- There is no negative correlation between variables.
- Correlation between variables and "Outcome" fluctuate from 0.07 (no correlation) to 0.48 (half correlated)

As a conclusion, we may say the data set is clean, no problem with data, and the lack of pure correlation will make this model building interesting.

Model building

Before we start our construction, we need to separate our set in 3 parts :

- the train set : to train our models
- the test set : to test our models performances
- the validation : to use it for the final model

We manage to do this with this piece of code :

```r
set.seed(1)
X <- diabetes %>% select(-Outcome)
y <- as.factor(diabetes$Outcome)

## Validation set
test_validation <- createDataPartition(y, times = 1, p = 0.9, list = FALSE)
validation <- diabetes[-test_validation, ]
test_index <- createDataPartition(y, times = 1, p = 0.8, list = FALSE)

## Train & Test set
train <- diabetes[test_index, ]
test <- diabetes[-test_index, ]

# Outcome as factor
train$Outcome <- as.factor(train$Outcome)
test$Outcome <- as.factor(test$Outcome)
validation$Outcome <- as.factor(validation$Outcome)
```

set	length
train	615
test	153
validation	76

Then, we can try our first model.

First model

This initial model will be a "glm" one. In fact, what matters is the metrics of the model. To compute the accuracy and the f1-score on a first prediction with this code :

```r
# fitting
fit <- train(Outcome~.,
             data=train,
             method="glm")

# predict
pred <- predict(fit,test)

# confusion matrix
cm <- confusionMatrix(pred,test$Outcome,mode="everything",positive="1")

# accuracy
cm[3]$overall[1]
```

```
##  Accuracy
## 0.7712418
```

```
# f1-score
cm[4]$byClass[7]
```

```
##        F1
## 0.6601942
```

For a first model, without any optimization, results are pretty positive. Now we have the methodology to fit and predict with a model, we can create a *function* to get a tibble with metrics of the models, on the train and the test set.

```
model_result <- function(method){

  # cross-validation control
  ctrl <- trainControl(method = "repeatedcv",
                       number = 10,
                       repeats = 3)

  # fitting
  fit <- train(Outcome~.,
               data=train,
               trControl = ctrl,
               method=method)

  # --- TEST ---
  # predict
  pred_test <- predict(fit,test)

  # confusion matrix
  cm_test <- confusionMatrix(pred_test,test$Outcome,mode="everything",positive="1")

  # accuracy
  accuracy_test <- round(cm_test[3]$overall[1],3)
  # f1-score
  f1_score_test <- round(cm_test[4]$byClass[7],3)

  # --- TRAIN ---
  # predict
  pred_train <- predict(fit,train)

  # confusion matrix
  cm_train <- confusionMatrix(pred_train,train$Outcome,mode="everything",positive="1")

  # accuracy
  accuracy_train <- round(cm_train[3]$overall[1],3)
  # f1-score
  f1_score_train <- round(cm_train[4]$byClass[7],3)


  # tibble with results
  result <- tibble("model"=method,
                   "train_acc"=accuracy_train,
                   "test_acc"=accuracy_test,
                   "acc_diff"=round(abs(accuracy_train-accuracy_test),5),
                   "train_f1"=f1_score_train,
                   "test_f1"=f1_score_test,
                    "f1_diff"=round(abs(f1_score_train-f1_score_test),5)
                    )
  return(result)
}
```

Finding the best model

To find the best model, we will try different ones on a list. Among the most popular binary classification models, 5 were selected from the caret package :

1. svmLinear : Support Vector Machines with Linear Kernel
2. rf : Random Forest
3. glm : Generalized Linear Model (first try)
4. glmboost : Boosted Generalized Linear Model (tunable glm)
5. lda : Linear Discriminant Analysis

We proceed to study the metrics they return (this step can take few minutes to achieve):

```
##       model train_acc test_acc acc_diff train_f1 test_f1 f1_diff ratio
## 3       glm     0.771    0.771    0.000    0.628   0.660   0.032  3.86
## 5       lda     0.771    0.778    0.007    0.630   0.667   0.037  3.76
## 4  glmboost     0.769    0.765    0.004    0.628   0.640   0.012  3.75
## 1 svmLinear     0.774    0.784    0.010    0.627   0.673   0.046  3.73
## 2        rf     1.000    0.752    0.248    1.000   0.612   0.388  1.68
```

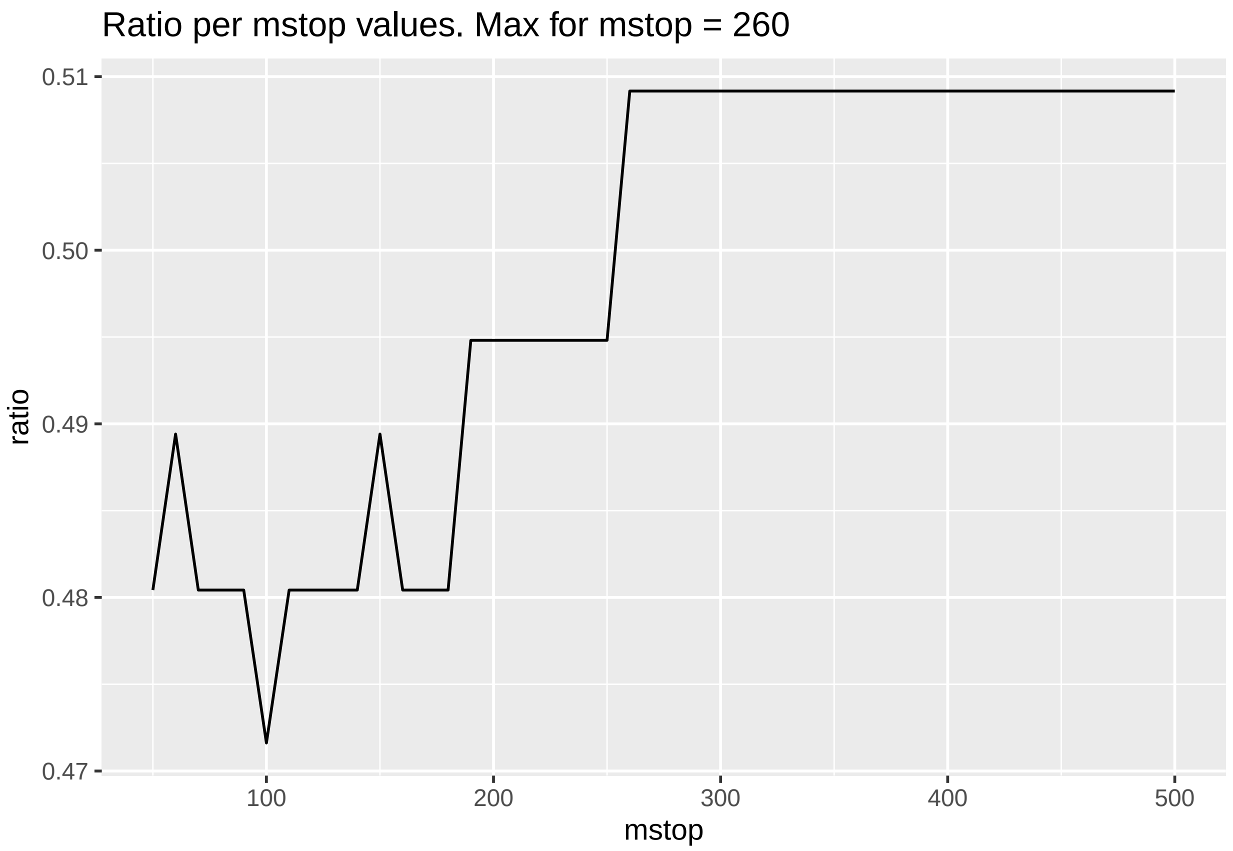
To understand this results table, the "diff" columns are the differences between the train and test values of the metric. It allows use to prevent an over fitting case. Moreover, the "ratio" columns is a homemade variable that gives us a view on which model is the best, according to the accuracy rate and the ratio.

As a conclusion, except "rf" (Random Forest), they all have results on the same range. Since "glm" and "lda" don't have values we can tune, we need to choose between "glmboost" and "svmLinear". Let's tune them both to see which one will prevail.

Tuning glmBoost

With the glmBoost method, two values need to be set : *mstop* and *prune*. Prune is a “yes/no” value. By setting it to “yes”, mstop will be automatically selected. But since we want to manually tune this up, we will set it as “no”.

About the *mstop* value, we will set a range from 50 to 500, by steps of 10. We create a new metric : ratio = accuracy * f1-score. This permits to find the best compromise between accuracy and f1-score by plotting ratio per mstop value. The highest the ratio, the better.



We have the mstop value that maximize the ratio, we can now plot the result for an optimized glmboost model :

```
##      model accuracy f1_score
## 1 glmBoost    0.771     0.66
```

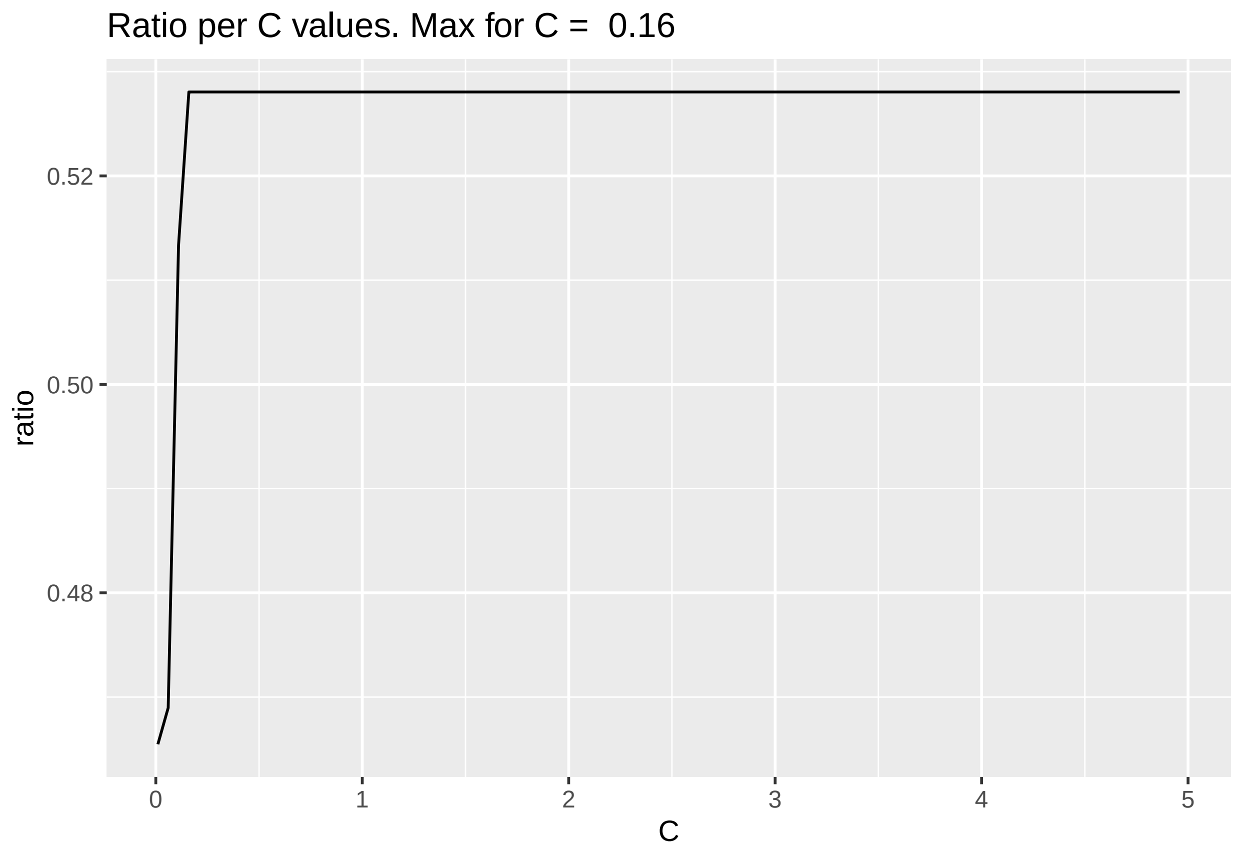
model	accuracy	f1_score
glmBoost	0.771	0.66

The accuracy is average, and the f1-score is not really higher the other models we tried. We will tune the svlLinear model to see if we get better results.

Tuning svmLinear

The svmLiner tuning will be done the same way glmboost was. The only exception is that there is only one value to tune, instead of two : C. A range of 0.01 to 2 with a step of 0.02 is set.

We, as previously, plot ratio (defined as accuracy*f1-score aswell) per C values.



With this C value, we have these results for svmLinear :

```
##       model accuracy f1_score
## 1 svmLinear    0.784    0.673
```

Accuracy and F1-score are a little better than the glmboost model. We will analyze the results ine the next part.

Results

Model finding results :

```
##       model train_acc test_acc acc_diff train_f1 test_f1 f1_diff ratio
## 3       glm     0.771    0.771    0.000    0.628   0.660   0.032  3.86
## 5       lda     0.771    0.778    0.007    0.630   0.667   0.037  3.76
## 4  glmboost     0.769    0.765    0.004    0.628   0.640   0.012  3.75
## 1 svmLinear     0.774    0.784    0.010    0.627   0.673   0.046  3.73
## 2        rf     1.000    0.752    0.248    1.000   0.612   0.388  1.68
```

Tuning results :

```
##
##
## |model     | accuracy| f1_score|
## |:---------|--------:|--------:|
## |glmBoost  |    0.771|    0.660|
## |svmLinear |    0.784|    0.673|
```

Among all the models we've tried, svmLinear tends to be the best. However, we can see that, unlike glmBoost that receive a growth in its metrics through tuning, svmLinear remains the same. Now we can recreate a svmLinear model from scratch to get all the information we can extract on this model performance.

```
##
##
## |Model     | Accuracy| F1.score| Precision| Recall| Sensitivity| Specificity|
## |:---------|--------:|--------:|---------:|------:|-----------:|-----------:|
## |svmLinear |    0.789|    0.667|     0.727|  0.615|       0.615|        0.88|
```

Results on the validation set are clearly better. Let's analyze it metric per metric :

- accuracy : we saw this one earlier, it is higher than what we got first, making this model stronger.
- F1-score : it remains almost the same, so we can't conclude it gets better.
- Precision & Recall : in fact, F1-score is a sort of mean of precision and recall, that's why f1-score is worth studying.
- Sensitivity : is at a good level, but since it is defined as "true positive rate" and we face a medical case, it remains a bit low.
- specificity : this one is really high, it is defined as "true false rate" which is convenient in this case.

To understand the last two better, we can translate, for this project, the specificity as the "proportion of really sick people among all the people who were diagnosed with diabetes" and specificity as "proportion of people that are really healthy among all the ones who were diagnosed negative". Now we see why a high sensibility is mandatory in a medical study : we cannot let people get misdiagnosed. Specificity is important as well because we don't want healthy people to take a difficult treatment because they also were misdiagnosed.

Here is another metric used to clearly see how the prediction went during the use of the validation set, we call it "the confusion matrix" :

```
##           Reference
## Prediction  0  1
##          0 44 10
##          1  6 16
```

Conclusion

In conclusion, as a machine learning point of view, this final model can be useful. The metrics we saw are high enough to consider this model good. But, since we are studying a medical case, we cannot use this model in real life.

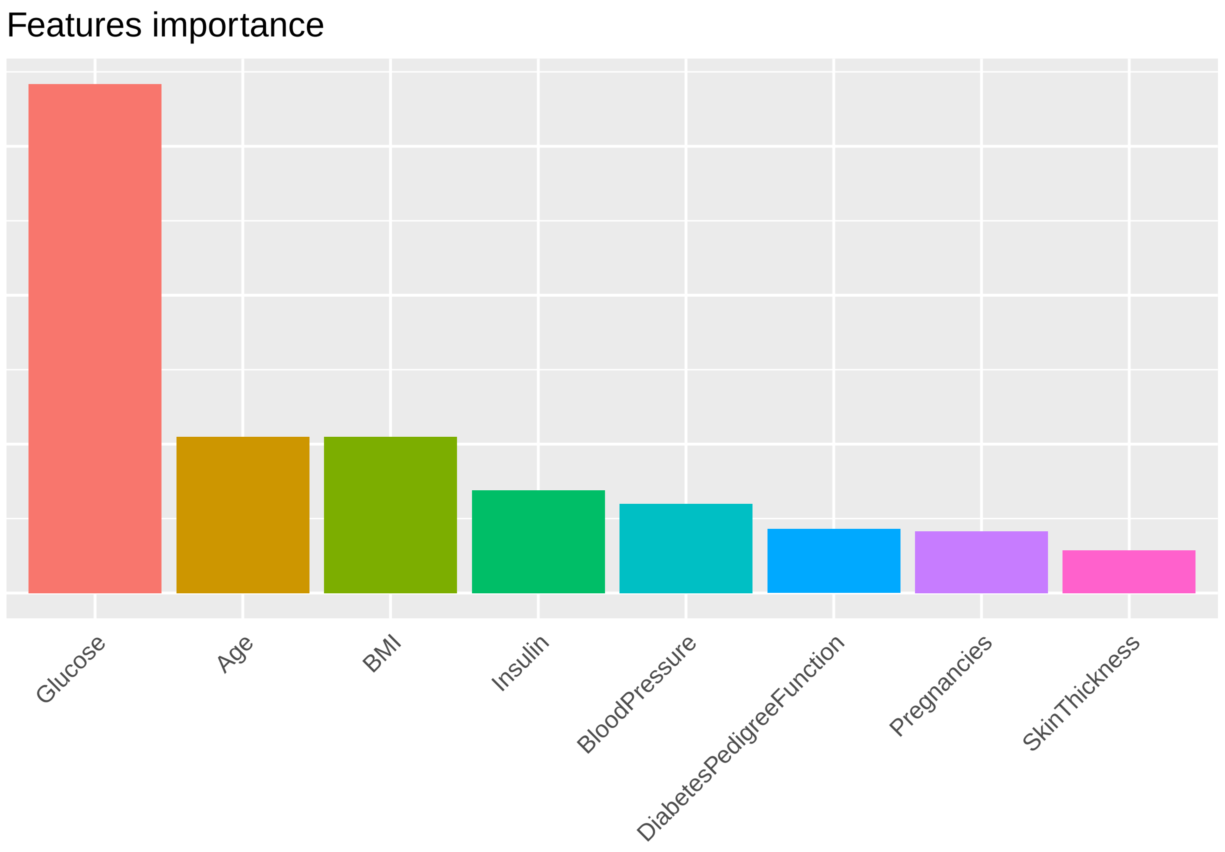
We can explain this with few points :

- the data set is usable, but for a really precise project, we would need more data
- the people targeted by this data set is really niche since it only concerns women from a specific area
- as we saw, variables are not really correlated to the outcome, which makes it harder for our machine learning to get higher results

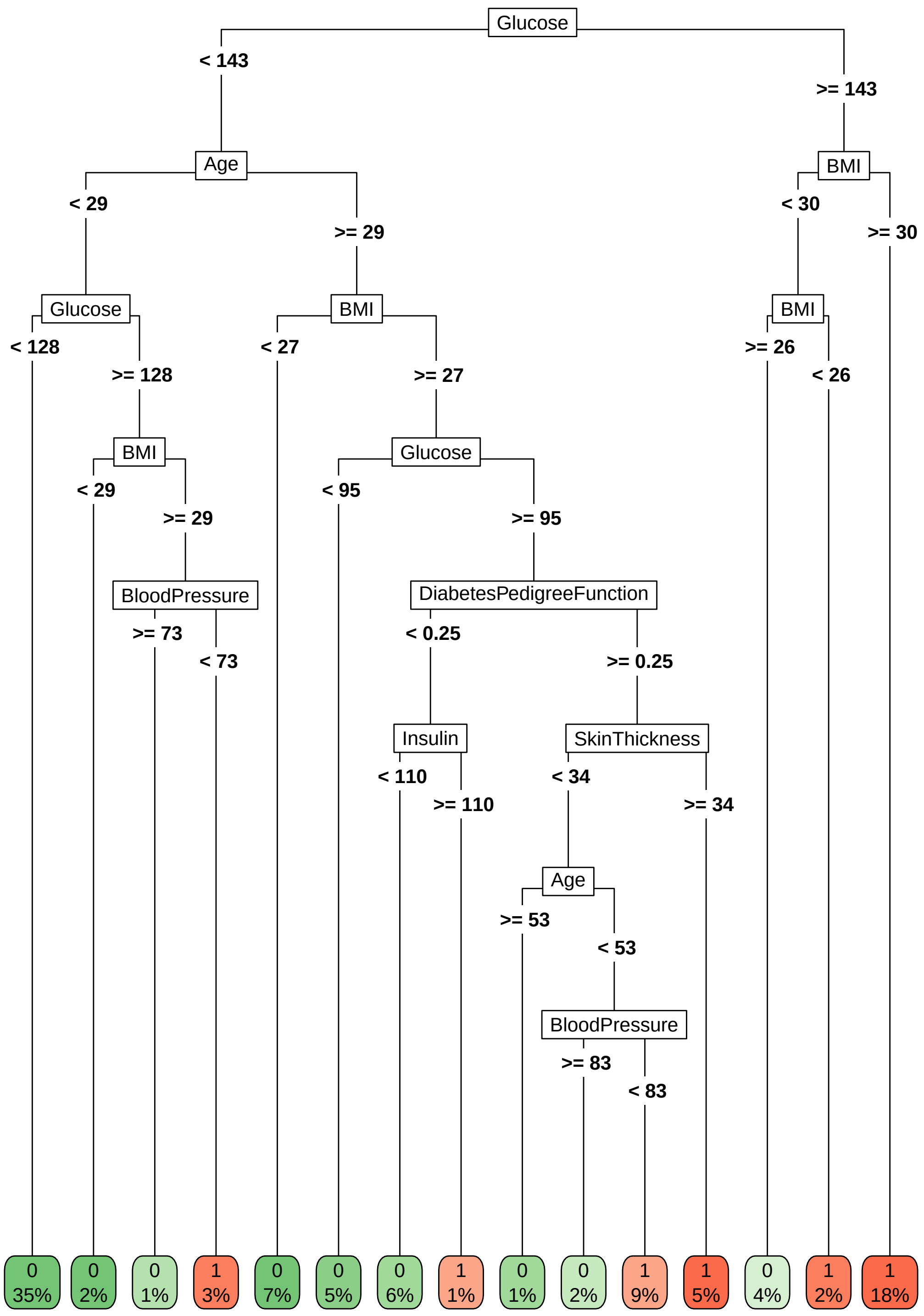
Finally, to have a better prediction model, we could have used Deep Learning and Neural Networks, but it is for an other level of data science and it is get outside of the machine learning field.

Bonus : decision tree

Since data scientists need to make predicting models, but also readable reports, I chose to use an other model to have a more straightforward view on prediction : the decision tree. We fit it on train data set to get the features importance the models use. Then, we can plot a tree that explains how the model predicts class.



Decision tree


Glucose
< 143
>= 143
Age
< 29
>= 29
BMI
< 30
>= 30
Glucose
< 128
>= 128
BMI
< 27
>= 27
BMI
>= 26
< 26
BMI
< 29
>= 29
Glucose
< 95
>= 95
BloodPressure
>= 73
< 73
DiabetesPedigreeFunction
< 0.25
>= 0.25
Insulin
< 110
>= 110
SkinThickness
< 34
>= 34
Age
>= 53
< 53
BloodPressure
>= 83
< 83
0 35%
0 2%
0 1%
1 3%
0 7%
0 5%
0 6%
1 1%
0 1%
0 2%
1 9%
1 5%
0 4%
1 2%
1 18%