Classification of patients affected by diabetes

Anna Bertani & Tamara Rigo

Introduction

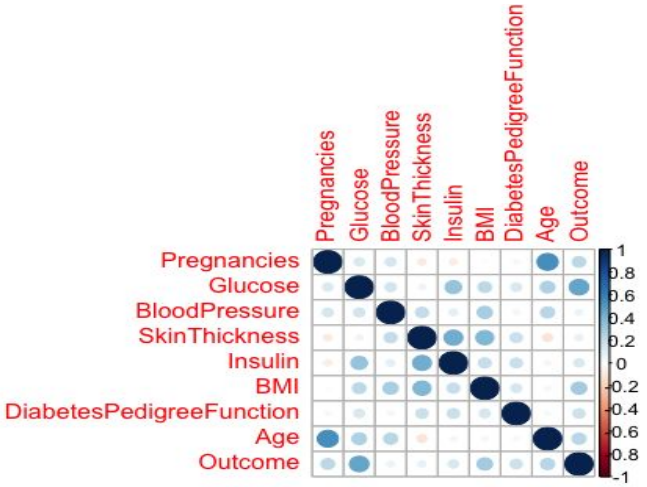
The aim of this project is to predict whether a female patient, given certain diagnostic measurements, has diabetes or not, through a series of classification models. Secondly, we would like to observe which among the variables in the dataset are the most important in order to better predict the Outcome (our response variable).In order to do that, we will use a dataset originally from the National Institute of Diabetes and Digestive and Kidney Disease (available at:https://www.kaggle.com/uciml/pima-indians-diabetes-database). The dataset presents several medical predictor variables and one target variable, called Outcome. Predictor variables include the number of pregnancies the patient has had, their BMI, insulin level, age and so on. It is important to underline that these data refer only to females at least 21 years old of Pima Indian heritage.

```
library(corrplot)
library(tidyr)
library(dplyr)
library(tmvnsim)
library(psych)
library(boot)
library(caret)
library(tidyverse)
library(glmnet)
library(class)
library(e1071)
library(tree)
library(randomForest)

knitr::opts_chunk$set(echo = TRUE) #settings for the plots
df <- read.csv("diabetes.csv", header = T) #readCSVany(is.na(df)) #check if
there are any NAs. No NAs found!
```

By performing *cor(df)* we can also see how there are some correlations among predictors, some stronger and some weaker. Focusing on the linear correlation between the outcome and the predictors, we can notice that the glucose predictor is the one with the highest linear correlation (0.46), followed by BMI, Age and Pregnancies. However, all the predictors are positively correlated to the Outcome, which underlines that all of them contribute to having diabetes to some extent.

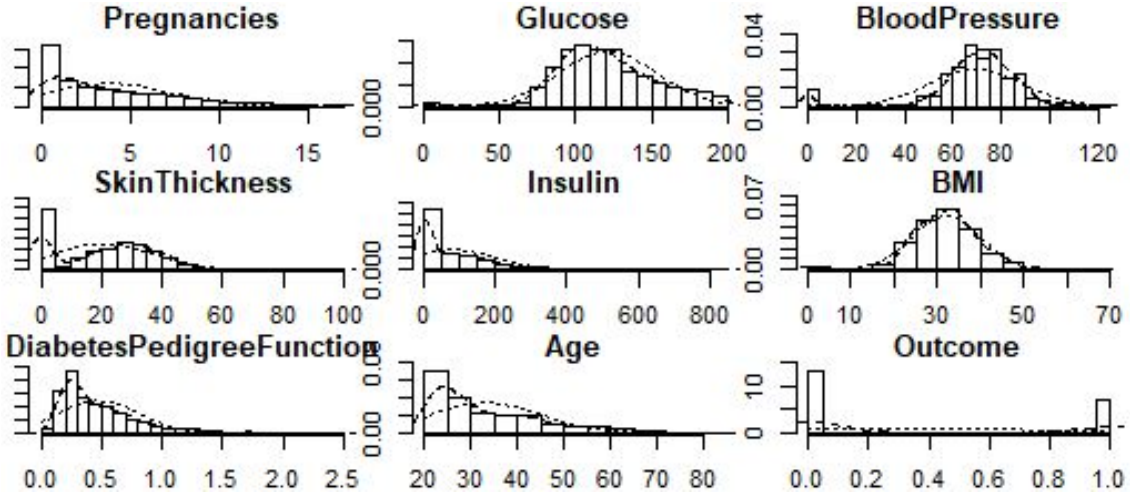
```
cor(df) #correlation among predictors
```

```
df_cor <- corrplot(cor(df)) #visualize correlation among predictors
```



Check the distribution of the variables:

```
multi.hist(df) #there are more healthy subjects than diabetics
```



Classification Methods

First of all, we should transform our response variable into a categorical one.

```
df$Outcome <- as.factor(df$Outcome)
```

Then we should divide our dataset into training and testing sets,this is an important step since we will use these sets throughout this project.

```
set.seed(25)
train <- sample(nrow(df), nrow(df)/2)
test <- df[-train,]
```

Logistic regression

We start our analysis with a logistic regression.

```
outcome_test <- df$Outcome[-train]
standardized_x <- scale(df[,-9])
train_set <- standardized_x[train,]
```

We perform logistic regression on the train set.

```
log_reg <- glm(df$Outcome ~ ., data = df, subset = train, family = binomial)
summary(log_reg)
```

```
##
## Call:
## glm(formula = df$Outcome ~ ., family = binomial, data = df, subset =
train)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.3878  -0.7294  -0.4280   0.7872   2.5777
##
## Coefficients:
##                            Estimate Std. Error z value Pr(>|z|)
## (Intercept)              -7.9512183  0.9944335  -7.996 1.29e-15 ***
## Pregnancies               0.0903976  0.0429948   2.103   0.0355 *
## Glucose                   0.0281398  0.0050593   5.562 2.67e-08 ***
## BloodPressure            -0.0165862  0.0072590  -2.285   0.0223 *
## SkinThickness             0.0055863  0.0102956   0.543   0.5874
## Insulin                  -0.0007482  0.0013756  -0.544   0.5865
## BMI                       0.0962577  0.0218846   4.398 1.09e-05 ***
## DiabetesPedigreeFunction  0.7938784  0.3963141   2.003   0.0452 *
## Age                       0.0308267  0.0126843   2.430   0.0151 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 497.98  on 383  degrees of freedom
## Residual deviance: 372.76  on 375  degrees of freedom
## AIC: 390.76
##
## Number of Fisher Scoring iterations: 5
```

```
glm.probs <- predict(log_reg, type="response", newdata = test)
glm.pred <- rep("0", nrow(test))
glm.pred[glm.probs > 0.5] <- "1"
table(glm.pred, test$Outcome) #confusion matrix
```

```
##
## glm.pred   0   1
##        0 217  53
##        1  34  80
```

```
err.logreg <- mean(glm.pred != test$Outcome) #error rate 0.23
```

As we can observe, some variables seem to be more important in predicting the outcome of having diabetes: glucose, BMI, age and pregnancies.

K-Fold Cross-Validation

We decided to also perform a K-fold CV, an approach similar to the LOOCV, but potentially less computational expensive. Indeed, this involves randomly dividing the set of observations in K folds of approximately equal size. The first fold is as the validation set, and the method is fit on the remaining k - 1folds. In this case, the accuracy is 78%

```
set.seed(123)
train.control <- trainControl(method = "cv", number = 10)#Define training
control
model <- train(Outcome ~., data = df, method = "glm",
               trControl = train.control)# Train the model
# Summarize the results
print(model) #Accuracy 0.78

## Generalized Linear Model
##
## 768 samples
##   8 predictor
##   2 classes: '0', '1'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 691, 691, 691, 692, 692, 691, ...
## Resampling results:
##
##   Accuracy   Kappa
##   0.7760082  0.4782385
```

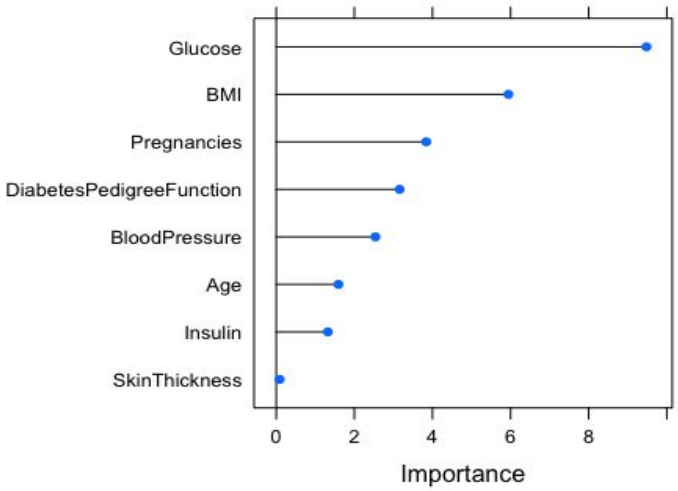
Repeated K-FOLD Cross Validation

An alternative approach is to repeat the K-Fold CV several times and report the mean performance across all the folds and all the repeats, called the repeated K-Fold CV. Also in this case, the accuracy is 0.78%. We are going to show an importance plot of the variables. As we could imagine, the most important predictors are Glucose and BMI, which are also the ones with the highest correlation with the variable Outcome.

```
set.seed(21)
control <- trainControl(method="repeatedcv", number=10, repeats=3)
log_imp <- train(Outcome ~., data=df, method="glm", family =
binomial,trControl=control)
print(log_imp) # Accuracy 0.78

## Generalized Linear Model
##
## 768 samples
##   8 predictor
```

```
##   2 classes: '0', '1'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold, repeated 3 times)
## Summary of sample sizes: 691, 691, 691, 692, 691, 691, ...
## Resampling results:
##
##   Accuracy  Kappa
##   0.773838  0.4755393
```

```
importance<- varImp(log_imp, scale=FALSE)
plot(importance)
```



Penalized logistic regression

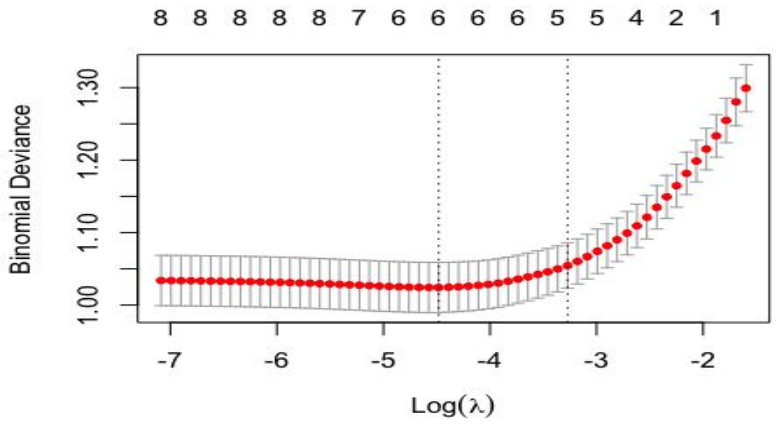
Here below, we will perform a method in order to investigate if there is a reduced set of variables that can lead to more accurate results. We start with the penalized logistic regression on the lasso model, which imposes a penalty to the logistic model for having too many variables. This results in shrinking the coefficients of the less contributive variables towards zero.

```
train.data  <- df[train, ]
x <- model.matrix(Outcome ~., train.data)[,-1]
y <- ifelse(train.data$Outcome == "1", 1, 0)
set.seed(1)
cv.lasso <- cv.glmnet(x, y, alpha = 1, family = "binomial") #alpha = 1 lasso
#Now we fit the model  on the training data
model <- glmnet(x, y, alpha = 1, family = "binomial",
                lambda = cv.lasso$lambda.min)
# Display regression coefficients
coef(model)
```

```
## 9 x 1 sparse Matrix of class "dgCMatrix"
##                                      s0
## (Intercept)              -7.15365783
```

```
## Pregnancies                 0.07310779
## Glucose                     0.02477842
## BloodPressure              -0.01028674
## SkinThickness               .
## Insulin                     .
## BMI                         0.08398178
## DiabetesPedigreeFunction    0.60647850
## Age                         0.02519127
```

```
# Make predictions on the test data
x.test <- model.matrix(Outcome ~., test)[,-1] #
probabilities <- model %>% predict(newx = x.test)
predicted.classes <- ifelse(probabilities > 0.5, "1", "0")
# Model accuracy
observed.classes <- test$Outcome
mean(predicted.classes == observed.classes)#0.76

err.penalized <- mean(predicted.classes != observed.classes)#error rate 0.24

set.seed(123)
cv.lasso <- cv.glmnet(x, y, alpha = 1, family = "binomial")
plot(cv.lasso)
```



The plot shows the cross-validation error according to the log of lambda. We find the value of lambda that gives the simplest model and then we use lambda.min and lambda.1se as the best lambda to see which regression coefficients show.

```
cv.lasso$lambda.min #0.011

cv.lasso$lambda.1se #0.038

coef(cv.lasso, cv.lasso$lambda.min)

## 9 x 1 sparse Matrix of class "dgCMatrix"
##                                      1
## (Intercept)               -7.088630865
```

```
## Pregnancies                0.071618749
## Glucose                    0.024584847
## BloodPressure             -0.009748631
## SkinThickness              .
## Insulin                    .
## BMI                        0.082597312
## DiabetesPedigreeFunction   0.590209427
## Age                        0.024671961
```

```
coef(cv.lasso, cv.lasso$lambda.1se)
```

```
## 9 x 1 sparse Matrix of class "dgCMatrix"
##                                      1
## (Intercept)               -5.61299675
## Pregnancies                0.03804530
## Glucose                    0.02053318
## BloodPressure              .
## SkinThickness              .
## Insulin                    .
## BMI                        0.05387276
## DiabetesPedigreeFunction   0.21456458
## Age                        0.01378908
```

```
# Final model with lambda.min
lasso.model <- glmnet(x, y, alpha = 1, family = "binomial",
                      lambda = cv.lasso$lambda.min)
# Make prediction on test data
x.test <- model.matrix(Outcome ~., test)[,-1]
probabilities <- lasso.model %>% predict(newx = x.test)
predicted.classes <- ifelse(probabilities > 0.5, "1", "0")
# Model accuracy
observed.classes <- test$Outcome
mean(predicted.classes == observed.classes)#0.76
```

```
## [1] 0.7630208
```

```
err.lambdamin <- mean(predicted.classes != observed.classes) #error rate 0.24

# Final model with lambda.1se
lasso.model <- glmnet(x, y, alpha = 1, family = "binomial",
                      lambda = cv.lasso$lambda.1se)
# Make prediction on test data
x.test <- model.matrix(Outcome ~., test)[,-1]
probabilities <- lasso.model %>% predict(newx = x.test)
predicted.classes <- ifelse(probabilities > 0.5, "1", "0")
# Model accuracy rate
observed.classes <- test$Outcome
mean(predicted.classes == observed.classes)#0.74
```

```
err.lambda1se <- mean(predicted.classes != observed.classes)  #error rate
0.26
```

From the output above, the variables SkinThickness and Insulin have a coefficient equal to zero, using lamda.min, while using lambda.1se, we obtain also BloodPressure as variables setted to 0. We compute the final lasso model using lasso.min and then assess the model accuracy against the test data, the same will be performed using lambda.1se. The accuracy obtained with lambda.min is slightly higher than the one with lambda.1se, demonstrating that imposing a penalty to more variables can lead to obtaining a model less accurate.

kNN

KNN is a non-parametric method which takes some advantages in cases of non-linear boundaries, even though it does not indicate which predictors are actually useful and which are not. First of all, we check mean and variance in order to see if it is necessary to standardize the variables to have zero mean and unit variance , since KNN is based on the distance. From the summary at the beginning of the analysis we have seen that the mean is inhomogeneous among the variables, we must check if this also applies to the variance.

```
apply(df, MARGIN=2, FUN=var)  #we need to standardize
```

```
##               Pregnancies                   Glucose             BloodPressure
##              1.135406e+01              1.022248e+03              3.746473e+02
##             SkinThickness                   Insulin                       BMI
##              2.544732e+02              1.328118e+04              6.215998e+01
## DiabetesPedigreeFunction                       Age                   Outcome
##              1.097786e-01              1.383030e+02              2.274826e-01
```

```
standardized.df<- as.data.frame(standardized_x)
```

Now we implement the KNN model in the most classical way, using the library class. We also use the simplest approach to find the number of clusters, selecting k by calculating k=sqrt(n).

```
set.seed(123)
train.sd <- standardized.df[train, ]
test.sd <- standardized.df[-train, ]

train.y <- df$Outcome[train]
test.y <- df$Outcome[-train]
ideal.k<-round(sqrt(dim(train.sd)[1])) #k=20
knn.pred <- knn(train.sd, test.sd, train.y, k=ideal.k)

table(knn.pred, test.y) #confusion matrix
```

```
##         test.y
## knn.pred   0   1
##        0 229  69
##        1  22  64
```

```
err.knn<-mean(test.y != knn.pred) # error rate 0.24
```

A more complex method consists in performing hyperparameter tuning to select the best value for k throughout a 10 k-fold cv.

```
standardized.df$outcome <- 1
standardized.df$outcome[df$Outcome == 0] <- 0
```

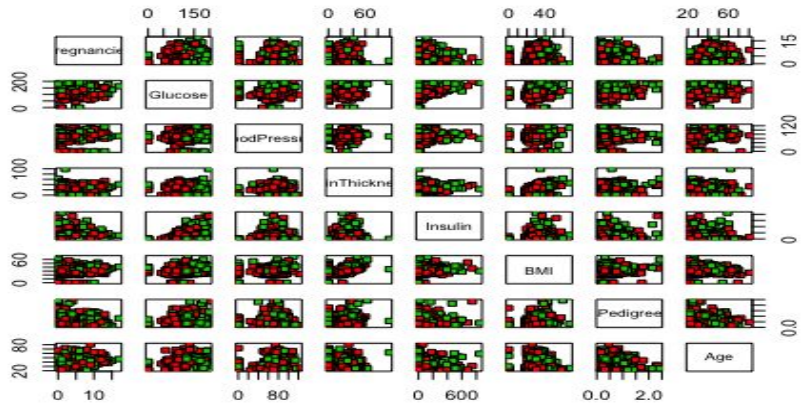
```
standardized.df$outcome<-as.factor(standardized.df$outcome)
set.seed(123)
shuffled.df <- standardized.df[sample(nrow(standardized.df)),] #reshuffle the
data
train_control<- trainControl(method = "cv", number = 10)
knn_kcv<- train(outcome ~ ., method = "knn", tuneGrid = expand.grid(k =
1:20),
                trControl = train_control, metric = "Accuracy", data =
shuffled.df)

knn_kcv[["bestTune"]][["k"]] #best value for k

## [1] 20
```

Also according to this CV setting, the best value for k is 20, so we have definitely found the best knn model. Although the error rate is acceptable, the results, as it can be seen in the confusion matrix, are not so brilliant. In particular, compared to the logistic regression, this model fails to well classify the items labelled as 1.

SVM

Support vector machines support different types of kernels, which define the way we measure similarity between two observations. Firstly, we have to decide what type of kernel to use:

```
pairs(df[-9], pch = 22, bg = c("red", "green3")[unclass(df$Outcome)])
#radial kernel seems the most suitable
```



An optimal value for the cost and the gamma parameters can be found by trying different values in 10-fold CV setting:

```
set.seed(123)
output.tune <- tune.svm( Outcome  ~ ., data = df[train, ], kernel = "radial",
                         gamma = 10^(-6:-1), cost = 10^(-1:2))
summary(output.tune)

##
## Parameter tuning of 'svm':
```

```
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  gamma cost
##  0.001  100
##
## - best performance: 0.247166
##
## - Detailed performance results:
##    gamma  cost     error dispersion
## 1  1e-06   0.1 0.3510796 0.07146202
## 2  1e-05   0.1 0.3510796 0.07146202
## 3  1e-04   0.1 0.3510796 0.07146202
## 4  1e-03   0.1 0.3510796 0.07146202
## 5  1e-02   0.1 0.3510796 0.07146202
## 6  1e-01   0.1 0.3146424 0.07993732
## 7  1e-06   1.0 0.3510796 0.07146202
## 8  1e-05   1.0 0.3510796 0.07146202
## 9  1e-04   1.0 0.3510796 0.07146202
## 10 1e-03   1.0 0.3537112 0.07049608
## 11 1e-02   1.0 0.2601889 0.06657761
## 12 1e-01   1.0 0.2679487 0.06387404
## 13 1e-06  10.0 0.3510796 0.07146202
## 14 1e-05  10.0 0.3510796 0.07146202
## 15 1e-04  10.0 0.3537112 0.07049608
## 16 1e-03  10.0 0.2575574 0.06257506
## 17 1e-02  10.0 0.2574899 0.05768010
## 18 1e-01  10.0 0.2993252 0.07596295
## 19 1e-06 100.0 0.3510796 0.07146202
## 20 1e-05 100.0 0.3537112 0.07049608
## 21 1e-04 100.0 0.2575574 0.06257506
## 22 1e-03 100.0 0.2471660 0.05634181
## 23 1e-02 100.0 0.2678138 0.05781085
## 24 1e-01 100.0 0.3334683 0.09346436
```

The lowest error (0.25) is obtained with gamma=1e-03 and cost=100 .We use this model:

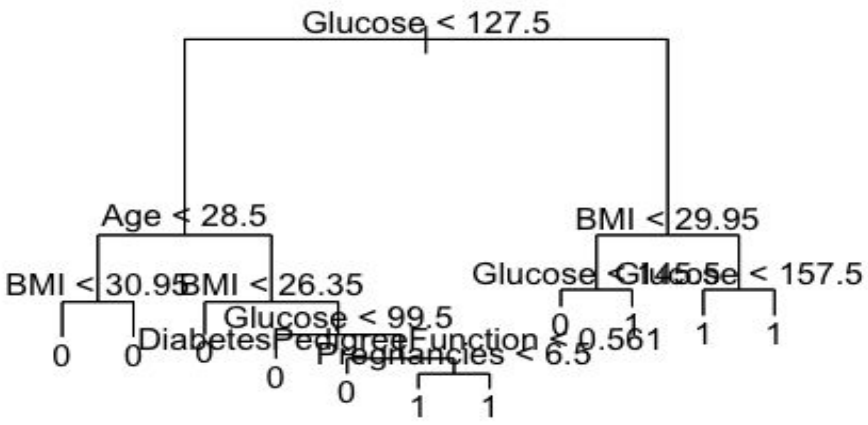
```
set.seed(123)
svmfit <- svm(Outcome ~ ., data=df[train,], kernel="radial", cost=100,
gamma=0.001,
              scale=FALSE)
#Confusion matrix:
svm.pred<- predict(svmfit, newdata=test)
table(svm.pred, test$Outcome)
## svm.pred   0   1
##        0 194  73
##        1  57  60

err.svm <- mean(svm.pred != test$Outcome) #error rate 0.34
```

Also, the svm model fails to well classify the items labeled as 1, slightly worse than the knn model. Moreover the test error is higher than those of the previous model.

Decision Tree

Decision trees have some interesting advantages. Indeed, trees are very easy to explain to people, they replicate human decision making, and this is the reason why they are more comprehensible but, on the other side, they are less robust than other techniques. A classifier performance is not properly evaluated until we estimate the test error.

```
set.seed(1)
tree_class <- tree(Outcome ~ ., data=df)
plot(tree_class)
text(tree_class, pretty=0)
```




```
set.seed(1)
tree_train <- tree(Outcome ~ ., data= df ,subset=train)
pred_tree <- predict(tree_train, test, type = "class")
table(pred_tree, outcome_test)
```

```
##           outcome_test
## pred_tree   0   1
##         0 191  53
##         1  60  80
```

```
mean(pred_tree == outcome_test) #accuracy 0.70

err.tree <-mean(pred_tree != outcome_test) #error rate 0.30
```

The most important variables are Glucose, BMI and Age, as we have also seen previously. Although previsions are less biased towards a single class, the accuracy of this model is 0.70, which is not satisfying as the first models we have performed.

Bagging

The decision trees suffer, in general, from high variance. This means that if we split the training into two parts at random, and fit a decision tree to both halves, the result that we get can be quite different. Bagging is another procedure for reducing the variance of statistical learning method and it is simply a special case of a random forest with m = p. Here we performed Bagging with all our predictors, which led us to an OOB estimate of error rate: 27.34%.

```
bagging <- randomForest(Outcome~., data = df, subset = train, mtry= 8,
importance=TRUE)
bagging
```

```
##
## Call:
##  randomForest(formula = Outcome ~ ., data = df, mtry = 8, importance =
TRUE,      subset = train)
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 8
##
##         OOB estimate of  error rate: 27.34%
## Confusion matrix:
##     0  1 class.error
## 0 200 49   0.1967871
## 1  56 79   0.4148148
```

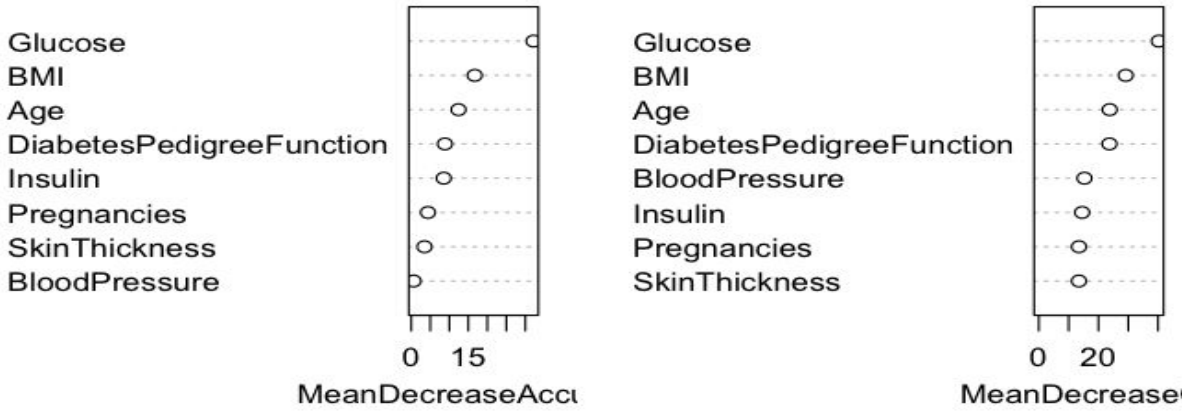
```
err.bagging <- 1 - (200+79)/384 #error rate 0.27
```

Random Forest

```
set.seed(1)
randomf <- randomForest(Outcome ~ ., data= df, subset= train, importance=
TRUE)
randomf # here the OOB error is slightly lower than bagging. 26.04%
## Call:
##  randomForest(formula = Outcome ~ ., data = df, importance = TRUE,
subset = train)
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 2
##
##         OOB estimate of  error rate: 26.04%
## Confusion matrix:
##     0  1 class.error
## 0 210 39   0.1566265
## 1  61 74   0.4518519
```

```
err.rf <- 1-(210+74)/384 # error rate 0.26
```

```
varImpPlot(randomf)
```



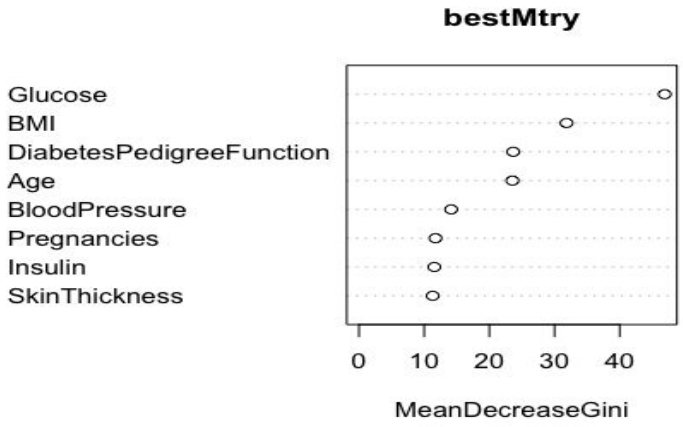
Then we perform a Random Forest and we show a variable importance plot, which displays that the most important variables are Glucose, BMI and Age, which is a result consistent with the one obtained by performing the logistic regression. The random forest is slightly better than the bagging by looking at the OOB estimate of error rate. Now, we are going to perform the hyperparameter tuning in order to obtain the best m.

```
set.seed(1)
bestMtry <- tuneRF(train_set, df[train,9], stepFactor = 1.5, doBest=T, plot=
F)

## mtry = 2  OOB error = 28.65%
## Searching left ...
## Searching right ...
## mtry = 3     OOB error = 26.82%
## 0.06363636 0.05
## mtry = 4     OOB error = 25.78%
## 0.03883495 0.05
```

The best m is 4 and in this way the OOB is even lower (25%).

```
bestMtry
## Call:
##  randomForest(x = x, y = y, mtry = res[which.min(res[, 2]), 1])
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 4
##
##         OOB estimate of  error rate: 25%
## Confusion matrix:
##     0  1 class.error
## 0 207 42   0.1686747
## 1  54 81   0.4000000

err.tuneRF <- 1- (207+81)/384 # error rate 0.25

bestMtry["mtry"] # the optimal number for m.
```

```
## [1] 4

varImpPlot(bestMtry)
```



Observing this plot, after performing the hyperparameter tuning, we can observe that the main variables are still Glucose and BMI, followed by DiabetesPedigreeFunction and Age, which are instead reversed in the previous plot.

Boosting

Boosting is another approach for improving the predictions resulting from a decision tree but it works in a different way than the previous two methods. Trees are grown sequentially and not with bootstrapping. We performed hyperparameter tuning in order to find the optimal number of iteration, which is 50, and then we perform Boosting looking for the best value of shrinking among the traditional values (0.1, 0.01, 0.001) and then we are going to test it.

```
trainControl <-trainControl(method="cv", number=5)
set.seed(1)
boost <- train(Outcome~., data= df[train,],
method="gbm",distribution="bernoulli",

trControl=trainControl,verbose=F,tuneGrid=data.frame(.n.trees=50,.shrinkage=c
(0.1, 0.01,0.001),
.interaction.depth=1, .n.minobsinnode=10))
boost$bestTune #the best tuning found

##   n.trees interaction.depth shrinkage n.minobsinnode
## 3      50                 1       0.1             10

set.seed(1)
pred_boost <- predict(boost, test, n.trees= 50)
postResample(pred_boost, outcome_test)#80% Accuracy

##  Accuracy     Kappa
## 0.7942708 0.5193002

err.boosting <- mean(pred_boost != outcome_test) #error rate 0.20
```

According to these results, boosting performs better than random forest and bagging.

Models Comparison

In order to make a comparison among all these models, it can be useful to see all the test errors in ascending order:

Model	TestError
Boosting	0.2057292
Logistic Regression	0.2265625
kNN	0.2369792
Penalized	0.2395833
Random Forest	0.2604167
Bagging	0.2734375
Tree	0.2942708
SVM	0.3385417

Conclusions

In this project, we have performed different classification models in order to predict whether a female patient is affected by diabetes or not. Among all, boosting seems to be the best model that classifies correctly the outcome with an accuracy of 80% and it can be used as the basis for prediction of the incidence of diabetes among Pima Indian women. The second best is the logistic regression with an accuracy of 78%. What we have observed is that a penalized logistic regression has an accuracy equal to 0.76, not improving the previous logistic regression. This can be due to the fact that our dataset is composed of only 8 predictors and all of them are positively correlated to the Outcome, as demonstrated by the corrplot. At the same time, we also found out which are the most important indicators to predict if a particular subject is likely to have diabetes or not.

For instance, Glucose and BMI are the most important predictors that are more than the others and this result is evident in all the *importance plots* but also in the analysis of the logistic regression and decision tree. Instead, the predictors less important are the SkinThickness and Insulin, as we could observe by looking at the penalized logistic regression and some of the importance variables. We should also bear in mind that, although our accuracy is not particularly high, considering that we have used a medical dataset, it can be assumed that the samples observed are very restricted and probably not obtained from randomized samplings.