# Bagging

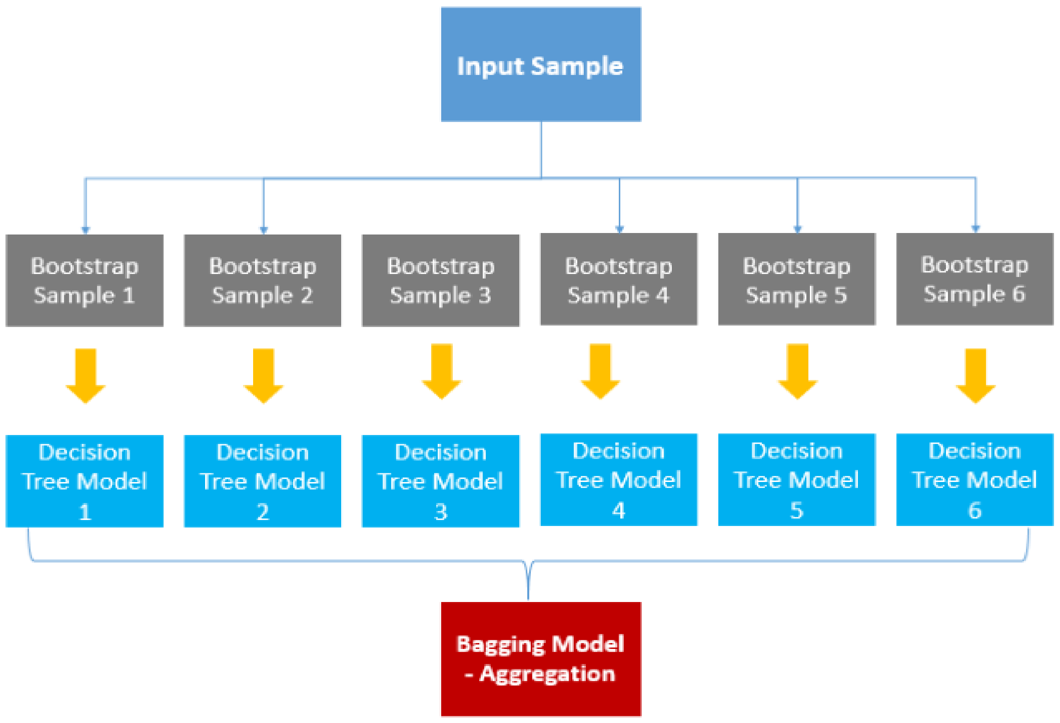
Bagging is a powerful method to improve the performance of simple models and reduce overfitting of more complex models. The principle is very easy to understand, instead of fitting the model on one sample of the population, several models are fitted on different samples (with replacement) of the population. Then, these models are aggregated by using their average, weighted average or a voting system (mainly for classification).

Bagging meaning **B**ootstrap **Agg**regation. Bootstrapping is a process of selecting samples from original sample (or population) and using these samples for estimating various statistics or model accuracy.

Bootstrap samples are created to estimate and validate models for improved accuracy, reduced variance and bias, and improved stability of a model. Once bootstrap samples are created, model classifier is used for training or building a model and then selecting model based on popularity votes. In classification model, a label with maximum votes will assigned to the observations. Average value is used in case of a regression model.

## Bagging: Overview

Bagging is an ensembling process – where a model is trained on each of the bootstrap samples and the final model is an aggregated model of the all sample models. For a numeric target variable /regression problem the predicted outcome is an average of all the models and in the classification problems, the predicted class is defined based on plurality.



In R, “**adabag**” and “**ipred**” packages allow us to develop bagging-based models for both classification and regression scenarios.
Tree Building process used on Bagging is based on CART algorithm and **adabag and ipred** use **rpart** algorithm**.**

## Bagging for a Classification Scenario

Predictive Modelling has varied and large number of real life examples and significant percentage of those business scenarios are Binary Predictive Modelling. One of the widely used application of Predictive Modelling is Credit Risk Scoring. Main objective in Credit Risk Scoring is to predictive defaulter based on applicant or customer attributes. Overview to Credit Underwriting and Analytics .

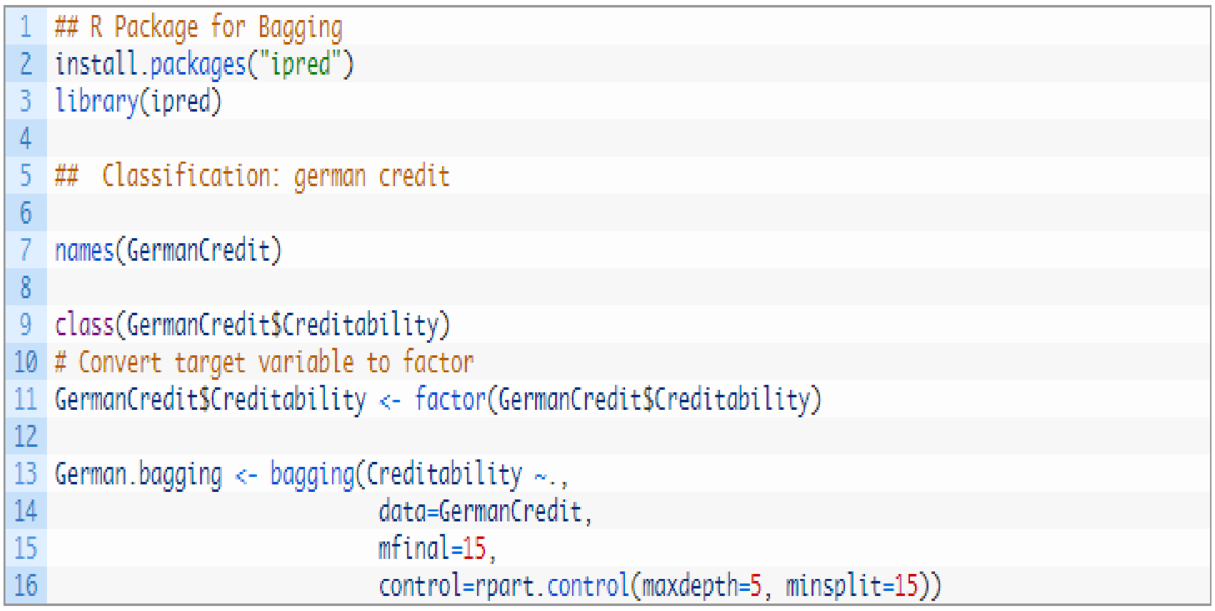
We are taking German Credit data as an example to build a Predictive Model using Bagging Classifier.

### Bagging using R for Classification

We have used **ipred** package and it has **bagging** function for building predictive model using Bagging classifier.

Bagging using R for Classification

We have used **ipred** package and it has **bagging** function for building predictive model using Bagging classifier.



```r
## R Package for Bagging
install.packages("ipred")
library(ipred)

##  Classification: german credit

names(GermanCredit)

class(GermanCredit$Creditability)
# Convert target variable to factor
GermanCredit$Creditability <- factor(GermanCredit$Creditability)

German.bagging <- bagging(Creditability ~.,
                          data=GermanCredit,
                          mfinal=15,
                          control=rpart.control(maxdepth=5, minsplit=15))
```

Now, we want to score the current dataset and get a predicted class variable/level for each of the observations. We can use **predict** function for the same.

```r
# score the dataset
GermanCredit$pred.class <- predict(German.bagging,
                                  GermanCredit)
```

Now , we want to understand accuracy of the classification and a few other performance measures.A summary on model performance statistics and calculations. Also if you are interested in R code on Model Performance.

```r
# Load Library or packages
library(e1071)
library(caret)
# Create Confusion Matrix
confusionMatrix(data=factor(GermanCredit$pred.class),
                reference=factor(GermanCredit$Creditability),
                positive='1')
```

```
Confusion Matrix and Statistics

          Reference
Prediction   0   1
         0 144  35
         1 156 665

               Accuracy : 0.809
                 95% CI : (0.7832, 0.8329)
    No Information Rate : 0.7
    P-Value [Acc > NIR] : 2.857e-15

                  Kappa : 0.486
 Mcnemar's Test P-Value : < 2.2e-16

            Sensitivity : 0.9500
            Specificity : 0.4800
         Pos Pred Value : 0.8100
         Neg Pred Value : 0.8045
             Prevalence : 0.7000
         Detection Rate : 0.6650
   Detection Prevalence : 0.8210
      Balanced Accuracy : 0.7150

       'Positive' Class : 1
```

Bootstrap aggregating, also called bagging, is a machine learning ensemble meta-algorithm designed to improve the stability and accuracy of machine learning algorithms used in statistical classification and regression. It also reduces variance and helps to avoid overfitting. Although it is usually applied to decision tree methods, it can be used with any type of method. Bagging is a special case of the model averaging approach. The out-of-bag estimate uses the observations which are left out in a boot- strap sample to estimate the misclassification error at almost no additional computational costs.

## Description

Fits the Bagging algorithm proposed by Breiman in 1996 using classification trees as single classifiers.

## Usage

```
bagging(formula, data, mfinal = 100, control, par=FALSE,...)
```

## Arguments

| | |
|---|---|
| `formula` | a formula, as in the `lm` function. |
| `data` | a data frame in which to interpret the variables named in the `formula` |
| `mfinal` | an integer, the number of iterations for which boosting is run or the number of trees to use. Defaults to `mfinal=100` iterations. |
| `control` | options that control details of the rpart algorithm. See rpart.control for more details. |
| `par` | if `TRUE`, the cross validation process is runned in parallel. If `FALSE` (by default), the function runs without parallelization. |

# 1. Decision Tree

## 1.5 Decision Tree in R

```
# classification decision tree
library (tree)
```

```
## Warning: package 'tree' was built under R version 3.3.3
```

```
library ( ISLR)
```

```
## Warning: package 'ISLR' was built under R version 3.3.3
```

```
attach ( Carseats )
High= ifelse (Sales <=8 ," No"," Yes ")
Carseats = data.frame ( Carseats, High)
tree.carseats = tree( High~.-Sales , Carseats )
summary ( tree.carseats )
```

```
##
## Classification tree:
## tree(formula = High ~ . - Sales, data = Carseats)
## Variables actually used in tree construction:
## [1] "ShelveLoc"   "Price"       "Income"      "CompPrice"   "Population"
## [6] "Advertising" "Age"         "US"
## Number of terminal nodes:  27
## Residual mean deviance:  0.4575 = 170.7 / 373
## Misclassification error rate: 0.09 = 36 / 400
```

```
# tree plot
plot( tree.carseats )
text( tree.carseats , pretty =0)
```

```
# train and test
set.seed (2)
train = sample(1: nrow( Carseats ), 200)
Carseats.test= Carseats [- train ,]
High.test=High [- train ]
tree.carseats = tree( High~.-Sales ,Carseats ,subset = train )
tree.pred= predict (tree.carseats , Carseats.test , type ="class")
table (tree.pred , High.test )
```

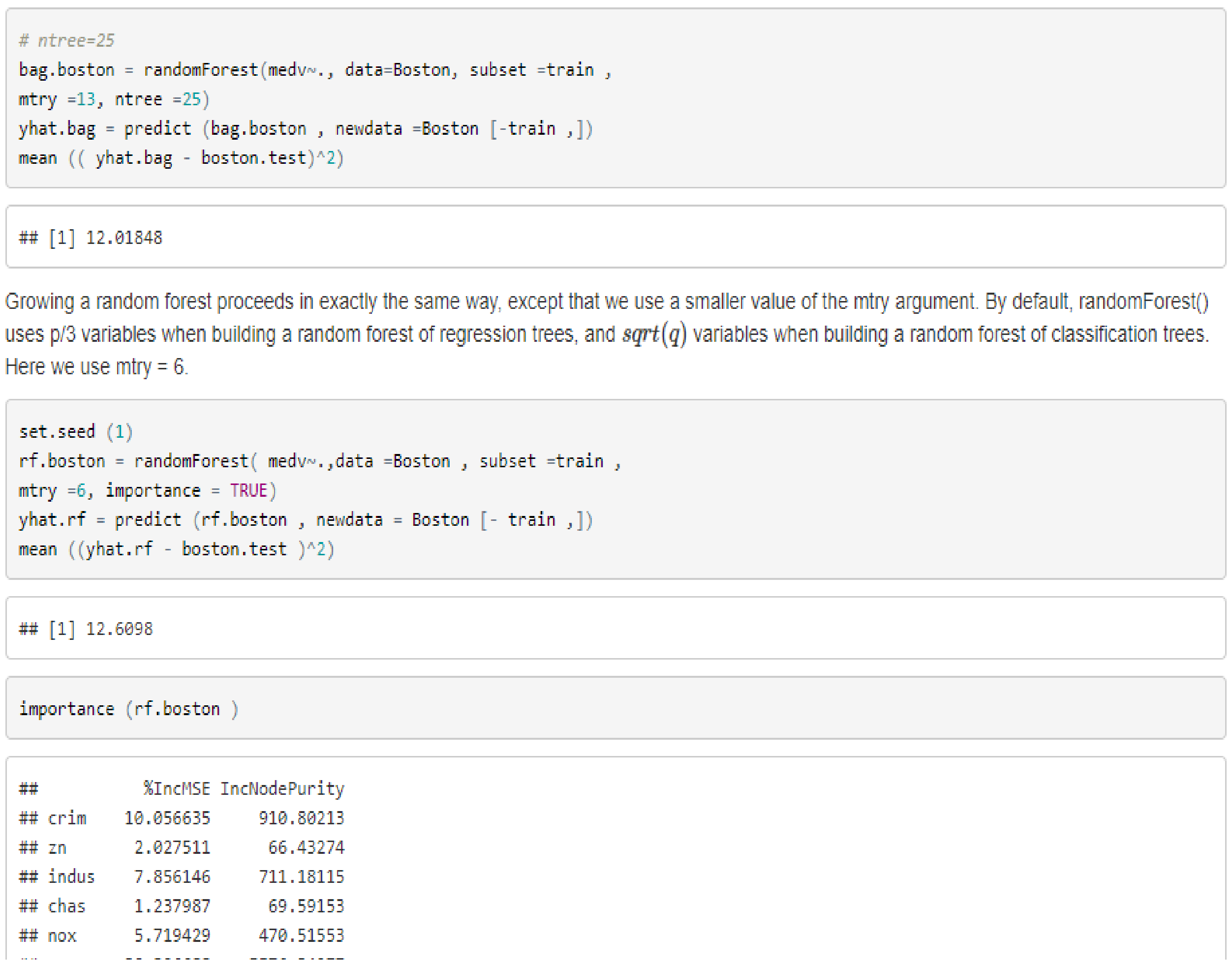
```
##           High.test
## tree.pred  No  Yes
##       No   86   27
##       Yes  30   57
```

Random forests or random decision forests are an ensemble learning method for classification, regression and other tasks, that operate by constructing a multitude of decision trees at training time and outputting the class that is the mode of the classes (classification) or mean prediction (regression) of the individual trees. Random decision forests correct for decision trees' habit of overfitting to their training set.

Bagging is simply a special case of a random forest with m = p. Therefore, the randomForest() function can be used to perform both random forests and bagging.

```
set.seed (1)

# Bagging
bag.boston = randomForest(medv~., data=Boston, subset =train, mtry =13, importance = TRUE)
boston.test = Boston [-train ,"medv"]
yhat.bag = predict (bag.boston , newdata =Boston[-train ,])
plot( yhat.bag , boston.test)
abline (0 ,1)
```



```
# ntree=25
bag.boston = randomForest(medv~., data=Boston, subset =train ,
mtry =13, ntree =25)
yhat.bag = predict (bag.boston , newdata =Boston [-train ,])
mean (( yhat.bag - boston.test)^2)
```

```
## [1] 12.01848
```

Growing a random forest proceeds in exactly the same way, except that we use a smaller value of the mtry argument. By default, randomForest() uses p/3 variables when building a random forest of regression trees, and $sqrt(q)$ variables when building a random forest of classification trees. Here we use mtry = 6.

```
set.seed (1)
rf.boston = randomForest( medv~.,data =Boston , subset =train ,
mtry =6, importance = TRUE)
yhat.rf = predict (rf.boston , newdata = Boston [- train ,])
mean ((yhat.rf - boston.test )^2)
```

```
## [1] 12.6098
```

```
importance (rf.boston )
```

```
##           %IncMSE IncNodePurity
## crim    10.056635     910.80213
## zn       2.027511      66.43274
## indus    7.856146     711.18115
## chas     1.237987      69.59153
## nox      5.719429     470.51553
```

The standard decision tree model, CART for classification and regression trees, build only one single tree, which is then used to predict the outcome of new observations. The output of this strategy is very unstable and the tree structure might be severally affected by a small change in the training data set.

There are different powerful alternatives to the classical CART algorithm, including **bagging**, **Random Forest** and **boosting**.

**Bagging** stands for bootstrap aggregating. It consists of building multiple different decision tree models from a single training data set by repeatedly using multiple bootstrapped subsets of the data and averaging the models. Here, each tree is built independently to the others.

**Random Forest** algorithm, is one of the most commonly used and the most powerful machine learning techniques. It is a special type of bagging applied to decision trees.
Compared to the standard CART model, the random forest provides a strong improvement, which consists of applying bagging to the data and bootstrap sampling to the predictor variables at each split. This means that at each splitting step of the tree algorithm, a random sample of n predictors is chosen as split candidates from the full set of the predictors.
Random forest can be used for both classification (predicting a categorical variable) and regression (predicting a continuous variable).

## Loading required R packages

- tidyverse for easy data manipulation and visualization
- caret for easy machine learning workflow
- randomForest for computing random forest algorithm

```
library(tidyverse)
library(caret)
library(randomForest)
```

## Classification

## Example of data set

Data set: PimaIndiansDiabetes2 [in mlbench package], introduced in Chapter @ref(classification-in-r), for predicting the probability of being diabetes positive based on multiple clinical variables.

Randomly split the data into training set (80% for building a predictive model) and test set (20% for evaluating the model). Make sure to set seed for reproducibility.

```
# Load the data and remove NAs
data("PimaIndiansDiabetes2", package = "mlbench")
PimaIndiansDiabetes2 <- na.omit(PimaIndiansDiabetes2)
# Inspect the data
sample_n(PimaIndiansDiabetes2, 3)
# Split the data into training and test set
set.seed(123)
training.samples <- PimaIndiansDiabetes2$diabetes %>%
  createDataPartition(p = 0.8, list = FALSE)
train.data  <- PimaIndiansDiabetes2[training.samples, ]
test.data <- PimaIndiansDiabetes2[-training.samples, ]
```

## Computing random forest classifier

We'll use the caret workflow, which invokes the randomforest() function [randomForest package], to automatically select the optimal number (mtry) of predictor variables randomly sampled as candidates at each split, and fit the final best random forest model that explains the best our data.

We'll use the following arguments in the function train():

- trControl, to set up 10-fold cross validation

```
# Fit the model on the training set
set.seed(123)
model <- train(
  diabetes ~., data = train.data, method = "rf",
  trControl = trainControl("cv", number = 10),
  importance = TRUE
  )
# Best tuning parameter
model$bestTune
```

```
##   mtry
## 3    8
```

```
# Final model
model$finalModel
```

```
## 
## Call:
##  randomForest(x = x, y = y, mtry = param$mtry, importance = TRUE) 
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 8
## 
##         OOB estimate of  error rate: 22%
## Confusion matrix:
##     neg pos class.error
## neg 185  25       0.119
## pos  44  60       0.423
```

```
# Make predictions on the test data
predicted.classes <- model %>% predict(test.data)
head(predicted.classes)
```

```
## [1] neg pos neg neg pos neg
## Levels: neg pos
```

```
# Compute model accuracy rate
mean(predicted.classes == test.data$diabetes)
```

```
## [1] 0.808
```

By default, 500 trees are trained. The optimal number of variables sampled at each split is 8.

Each bagged tree makes use of around two-thirds of the observations. The remaining one-third of the observations no used to fit a given bagged tree are referred to as the out-of-bag (OOB) observations (James et al. 2014).

For a given tree, the out-of-bag (OOB) error is the model error in predicting the data left out of the training set for that tree (P. Bruce and Bruce 2017). OOB is a very straightforward way to estimate the test error of a bagged model, without the need to perform cross-validation or the validation set approach.

In our example, the OBB estimate of error rate is 24%.

✓ The prediction accuracy on new test data is 79%, which is good.

## Variable importance

The importance of each variable can be printed using the function `importance()` [randomForest package]:

```
importance(model$finalModel)
```

```
##            neg    pos MeanDecreaseAccuracy MeanDecreaseGini
## pregnant 11.57  0.318                10.36             8.86
## glucose  38.93 28.437                46.17            53.30
## pressure -1.94  0.846                -1.06             8.09
## triceps   6.19  3.249                 6.85             9.92
## insulin   8.65 -2.037                 6.01            12.43
## mass      7.71  2.299                 7.57            14.58
## pedigree  6.57  1.083                 5.66            14.50
## age       9.51 12.310                15.75            16.76
```

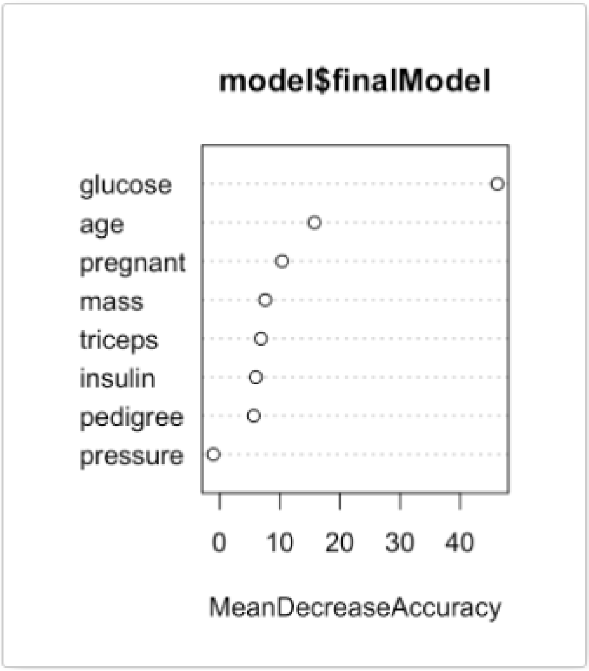
The result shows:

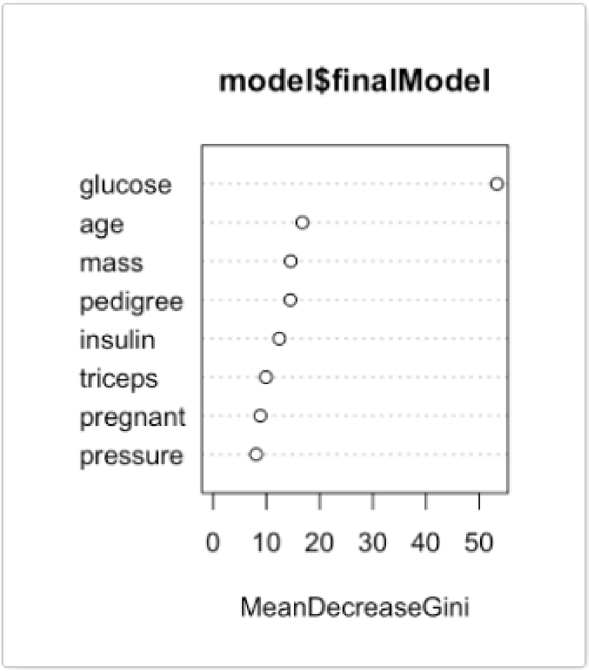
- `MeanDecreaseAccuracy`, which is the average decrease of model accuracy in predicting the outcome of the out-of-bag samples when a specific variable is excluded from the model.
- `MeanDecreaseGini`, which is the average decrease in node impurity that results from splits over that variable. The Gini impurity index is only used for classification problem. In the regression the node impurity is measured by training set RSS. These measures, calculated using the training set, are less reliable than a measure calculated on out-of-bag data. See Chapter @ref(decision-tree-models) for node impurity measures (Gini index and RSS).

✓ Note that, by default (argument importance = FALSE), randomForest only calculates the Gini impurity index. However, computing the model accuracy by variable (argument importance = TRUE) requires supplementary computations which might be time consuming in the situations, where thousands of models (trees) are being fitted.

Variables importance measures can be plotted using the function `varImpPlot()` [randomForest package]:

```
# Plot MeanDecreaseAccuracy
varImpPlot(model$finalModel, type = 1)
# Plot MeanDecreaseGini
varImpPlot(model$finalModel, type = 2)
```





The results show that across all of the trees considered in the random forest, the glucose and age variables are the two most important variables.

The function `varImp()` [in caret] displays the importance of variables in percentage:

```
varImp(model)
```

```
## rf variable importance
##
##          Importance
## glucose       100.0
## age            33.5
## pregnant       19.0
## mass           16.2
## triceps        15.4
## pedigree       12.8
## insulin        11.2
## pressure        0.0
```