# Diabetes Prediction Report

*Sairam Sasupathi*

*20 September 2016*

## Objective

To build a model that can predict whether a patient is diabetic or not.

## Introduction

The data we have acquired is from UCI Machine Learning Repository. Before we dive into making the model, we have to understand the data that we have.

```
summary(diabetes_train)
```

```
##   pregnancies     plasma.glucose  blood.pressure   triceps.skin.thickness
##  Min.   : 0.000   Min.   :  0.0   Min.   :  0.00   Min.   : 0.00
##  1st Qu.: 1.000   1st Qu.: 99.0   1st Qu.: 64.00   1st Qu.: 0.00
##  Median : 3.000   Median :117.0   Median : 72.00   Median :23.00
##  Mean   : 3.825   Mean   :120.5   Mean   : 68.89   Mean   :20.39
##  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
##     insulin            bmi       diabetes.pedigree      age
##  Min.   :  0.00   Min.   : 0.0   Min.   :0.078     Min.   :21.00
##  1st Qu.:  0.00   1st Qu.:27.0   1st Qu.:0.240     1st Qu.:24.00
##  Median : 37.00   Median :32.0   Median :0.376     Median :29.00
##  Mean   : 80.05   Mean   :31.9   Mean   :0.476     Mean   :33.11
##  3rd Qu.:128.00   3rd Qu.:36.5   3rd Qu.:0.637     3rd Qu.:40.00
##  Max.   :846.00   Max.   :67.1   Max.   :2.420     Max.   :81.00
##   diabetes
##  false:460
##  true :241
##
##
##
##
```

```
summary(diabetes_test)
```

```
##   pregnancies    plasma.glucose  blood.pressure  triceps.skin.thickness
##  Min.   : 0.00   Min.   : 65.0   Min.   : 0.00   Min.   : 0.00
##  1st Qu.: 1.00   1st Qu.:102.0   1st Qu.:62.00   1st Qu.: 0.00
##  Median : 3.00   Median :121.0   Median :74.00   Median :26.00
##  Mean   : 4.06   Mean   :124.6   Mean   :71.36   Mean   :22.04
##  3rd Qu.: 6.00   3rd Qu.:140.5   3rd Qu.:82.00   3rd Qu.:33.00
##  Max.   :13.00   Max.   :190.0   Max.   :94.00   Max.   :48.00
##     insulin            bmi        diabetes.pedigree      age
##  Min.   :  0.00   Min.   : 0.00   Min.   :0.1180    Min.   :21.00
##  1st Qu.:  0.00   1st Qu.:29.15   1st Qu.:0.2585    1st Qu.:24.00
##  Median :  0.00   Median :32.80   Median :0.3490    Median :32.00
##  Mean   : 77.16   Mean   :32.99   Mean   :0.4284    Mean   :34.57
##  3rd Qu.:123.50   3rd Qu.:37.15   3rd Qu.:0.5055    3rd Qu.:42.50
##  Max.   :510.00   Max.   :49.30   Max.   :1.1820    Max.   :66.00
##   diabetes
##  false:40
##  true :27
##
##
##
##
```

## Data Cleaning

The variables namely plasma.glucose, blood.pressure, triceps.skin.thickness, insulin, bmi, diabetes,pedigree have zero values which are not possible. We deduce that the data set has zero as entry for missing values. Hence they're to be replaced with NA to make better sense.

Zeros' of different variables are replaced with NAs in the below code.

```
for(i in 2:7)
  diabetes_train[diabetes_train[,i] == 0,i] = NA
for(i in 2:7)
  diabetes_test[diabetes_test[,i] == 0,i] = NA
```

## Feature Selection for modeling

The feature selection is done by setting a lesser complexity parameter to the decision tree algorithm and manually picking only the variables that are helpful. Also the variables with high number of NA like insulin are not to be considered as they may not help much. We cannot use "caret" package since it cannot handle the missing values. The CART algorithm is capable of handling missing values hence, we can use rpart package. J48 algorithm of the RWeka package is also another option. Hence the decision to use the CART, J48 over C5.0 and use "rpart", "J48" over "caret" package is made.

Feature selection is done using the less complexity parameter setting. The features that are contributing more to the top of the trees are selected. The features or the variables contributing to the top 10 nodes are selected.

Before we do that, lets load the required packages. This step cannot always be decided before hand, so this step will is updated during the various stages of the program depending on the requirement. We will be using the below packages for different actions. + "rpart" is used for decision tree algorithm CART, + "RWeka" is used for decision tree algorithm J48, + "caret" is used for the function of confusion matrix that it provides, + "rattle" is used for the visualisation of the models it provides.

```
library(rpart)
library(RWeka)
library(caret)
```

```
## Loading required package: lattice
## Loading required package: ggplot2
```

```
library(rattle)
```

```
## Rattle: A free graphical interface for data mining with R.
## Version 4.1.0 Copyright (c) 2006-2015 Togaware Pty Ltd.
## Type 'rattle()' to shake, rattle, and roll your data.
```

We will set a seed value so that we will get the same output everytime we run as there are some functions within rpart, RWeka which will use random digit generation during the Cross Validation of the models.

```
set.seed(16027)
```

Lets do the feature selection by creating a tree with very less complexity parameter. The reason being to find out what will be features that will affect the model at the top of the tree. Hence, we can use those features. The various parameters that are passed into the rpart function being used will be explained now.

1. 'rpart.control' is a function which is used to fine tune the model creation according to our requirement. Parameters used in 'rpart.control' function:

- 'xval' is the number of Cross Validations
- 'cp' is the permissible Complexity Parameter
- 'usesurrogate' is used to define how to use surrogates in the splitting process.
- 'surrogatestyle' is used to control the selection of a best surrogate.

2. 'rpart' function is used to create the model using the features sent to it and for a training dataset. Parameters used in 'rpart' function:

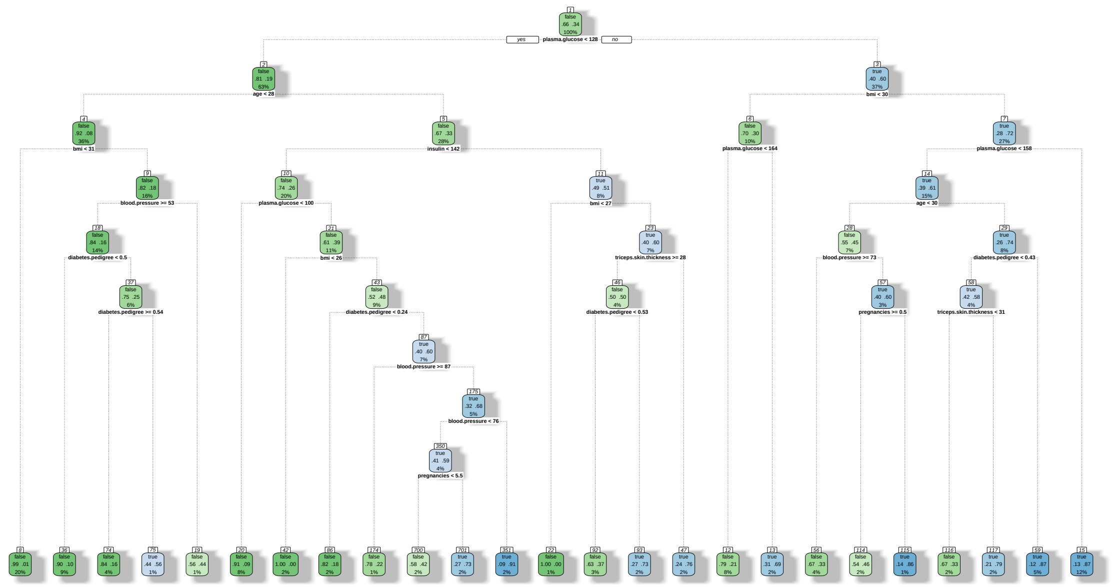
- 'formula' needs to have the feature that needs to be predicted on the lhs, followed by a ~ sign and other features that need to be considered for the model creation in the rhs. If we want to consider all the other features in the rhs, '.' will suffice.
- 'data' should have the data to be used for training the model.
- 'control' should have the rpart.control parameters that we want to fine tune our model with. If we don't give this, the function will use the default controls for the model.

3. 'predict' function uses the model created using any algorithm and applies it in the data set for testing. Parameters used in 'predict' function:

- first argument to the function will be the model generated using any function.
- 'newdata' should have the data to be used for testing the model generated.
- 'type' should have the type of the variable. In this case, it's class.

4. 'fancyRpartPlot' will print the model that is created using the 'rpart' function.

```
cntrl = rpart.control(xval = 10, cp = 0.001, surrogatestyle = 1, usesurrogate = 2)
model_NA = rpart(formula = diabetes ~ . , data = diabetes_train, control = cntrl)
pred_model_NA = predict(model_NA, newdata = diabetes_test, type = "class")
fancyRpartPlot(model_NA)
```



Rattle 2016−Dec−15 09:57:25 saiviki

plasma.glucose, bmi, age and insulin are the features that are considered as they are in the top of the tree. Also only these features are helping in dividing the data above 10%. Insulin will not be considered as the number of NAs in the Insulin column is very high.

Lets check what are the characteristics of the model that we've generated. 'confusionMatrix' function of 'caret' package is used to find the difference between the predicted values and the original data. The parameters to pass to this function are the predicted model and the test dataset used during the prediction.

```
costMatrix = matrix(c(0,150,50,0),nrow = 2)
confusionMatrix(pred_model_NA,diabetes_test$diabetes)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction false true
##      false    32   10
##      true      8   17
##
##                Accuracy : 0.7313
##                  95% CI : (0.609, 0.8324)
##     No Information Rate : 0.597
##     P-Value [Acc > NIR] : 0.01549
##
##                   Kappa : 0.4349
##  Mcnemar's Test P-Value : 0.81366
##
##             Sensitivity : 0.8000
##             Specificity : 0.6296
##          Pos Pred Value : 0.7619
##          Neg Pred Value : 0.6800
##              Prevalence : 0.5970
##          Detection Rate : 0.4776
##    Detection Prevalence : 0.6269
##       Balanced Accuracy : 0.7148
##
##        'Positive' Class : false
##
```

```
conf$table[2]*150 +  conf$table[3] *50
```

```
## [1] 1700
```

We're getting an accuracy of 73.53%. Kappa value for the model is 44.01%. Sensitivity and the Specificity are 80.49% and 62.96%. The Cost assosciated with the model is 1700.

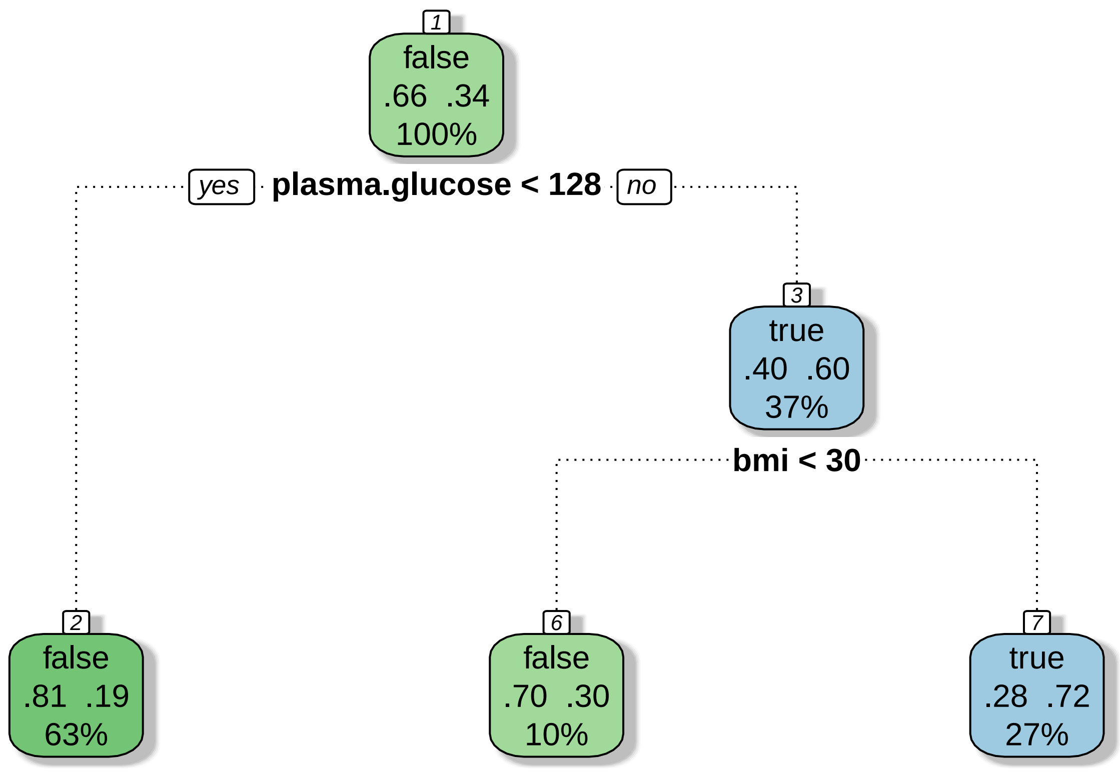
## Model Generation

As we've decided the features that need to be used, we can go ahead with creating the actual model generation by varying the parameters in the different functions that we can use. The functions "rpart" and "J48" are to be used and the model control for these functions will be changed to find the optimum model.

### rpart function

Lets try using the 'rpart' function. The rpart function can be fine tuned by using the Complexity parameter in the rpart.control function. We'll use loops to check the cost for different Complexity parameters. Iterating the decision tree from complexity parameter of 0.0010 to 0.200 with a increment of 0.0005.

```
for (i in seq(from = 0.0010, to = 0.2000, by = 0.0005)) {
  cnt_i = rpart.control(xval = 10, cp = i, surrogatestyle = 1, usesurrogate = 1)
  model_N = rpart(formula = diabetes ~ plasma.glucose + bmi + age + insulin,
                  data = diabetes_train, control = cnt_i)
  pred_model_N = predict(model_N, newdata = diabetes_test, type="class")
  j = confusionMatrix(pred_model_N, diabetes_test$diabetes)
  new_cost = j$table[2]*150 +  j$table[3] *50
# If the value of the new_cost is lesser than the previous cost, we will swap
# the complexity parameter value, cost, confusion matrix and the
# final model with the new variables.
  if (new_cost < cost)
  {
    cp_val = i
    cost = new_cost
    conf = j
    final_model = model_N
  }
  else NULL
}
```



Rattle 2016−Dec−15 09:57:31 saiviki

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction false true
##      false    35   12
##      true      5   15
##
##                Accuracy : 0.7463
##                  95% CI : (0.6251, 0.8447)
##     No Information Rate : 0.597
##     P-Value [Acc > NIR] : 0.007732
##
##                   Kappa : 0.4495
##  Mcnemar's Test P-Value : 0.145610
##
##             Sensitivity : 0.8750
##             Specificity : 0.5556
##          Pos Pred Value : 0.7447
##          Neg Pred Value : 0.7500
##              Prevalence : 0.5970
##          Detection Rate : 0.5224
##    Detection Prevalence : 0.7015
##       Balanced Accuracy : 0.7153
##
##        'Positive' Class : false
##

## [1] 0.021
## [1] 1350
```

## J48 function

Second, we'll try using the 'J48' function from 'RWeka' package. The J48 function can be fine tuned by using the either n-fold validation or Pruning Confidence at a time in the Weka_Control function.

## 10-Fold Validation

First, we'll check the 'J48' model for 10-fold validation with default complexity parameter.

```
model_J48 = J48(formula = diabetes ~ plasma.glucose + bmi + age,
           control = Weka_control(N = 10, R = T),
           data = diabetes_train)
predict_model_J48 = predict(model_J48, newdata = diabetes_test)
confusionMatrix(predict_model_J48, diabetes_test$diabetes)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction false true
##      false    28    5
##      true     12   22
##
##                Accuracy : 0.7463
##                  95% CI : (0.6251, 0.8447)
##     No Information Rate : 0.597
##     P-Value [Acc > NIR] : 0.007732
##
##                   Kappa : 0.494
##  Mcnemar's Test P-Value : 0.145610
##
##             Sensitivity : 0.7000
##             Specificity : 0.8148
##          Pos Pred Value : 0.8485
##          Neg Pred Value : 0.6471
##              Prevalence : 0.5970
##          Detection Rate : 0.4179
##    Detection Prevalence : 0.4925
##       Balanced Accuracy : 0.7574
##
##        'Positive' Class : false
##
```

```
n$table[2]*150 +  n$table[3] *50
```

```
## [1] 2050
```

## Looping over different Complexity Parameters

Next, We'll use loops to check the cost for different Complexity parameters. We will iterating the decision tree from complexity parameter of 0.0010 to 0.200 with a increment of 0.0005.

```
for (i in seq(from = 0.0010, to = 0.2000, by = 0.0005)) {
  model_J48 = J48(formula = diabetes ~ plasma.glucose + bmi + age,
           control = Weka_control(C = i),
           data = diabetes_train)
  predict_model_J48 = predict(model_J48, newdata = diabetes_test)
  j1 = confusionMatrix(predict_model_J48, diabetes_test$diabetes)
  new_cost = j1$table[2]*150 +  j1$table[3] *50
# If the value of the new_cost is lesser than the previous cost, we will swap
# the complexity parameter value, cost, confusion matrix and the
# final model with the new variables.
  if (new_cost < cost1)
  {
    cp_val1 = i
    cost1 = new_cost
    conf1 = j
    final_model1 = model_J48
  }
  else NULL
}
```

```
## J48 pruned tree
## ------------------
##
## plasma.glucose <= 127: false (429.0/82.0)
## plasma.glucose > 127
## |   bmi <= 29.9: false (70.0/21.0)
## |   bmi > 29.9: true (187.0/52.0)
##
## Number of Leaves  :  3
##
## Size of the tree :   5
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction false true
##      false    33    9
##      true      7   18
##
##                Accuracy : 0.7612
##                  95% CI : (0.6414, 0.8569)
##     No Information Rate : 0.597
##     P-Value [Acc > NIR] : 0.003593
##
##                   Kappa : 0.4977
##  Mcnemar's Test P-Value : 0.802587
##
##             Sensitivity : 0.8250
##             Specificity : 0.6667
##          Pos Pred Value : 0.7857
##          Neg Pred Value : 0.7200
##              Prevalence : 0.5970
##          Detection Rate : 0.4925
##    Detection Prevalence : 0.6269
##       Balanced Accuracy : 0.7458
##
##        'Positive' Class : false
##
```

```
## [1] 0.001
```

```
## [1] 1350
```

## Conclusion

The final model from both the rpart and the J48 prediction algorithms are fetching the cost of 1350. The model from both are one and the same. Hence, we can use any of the model created by the rpart or the J48.