DTI5126[EG]: Fundamentals Data Science

Assignment 2

Classification

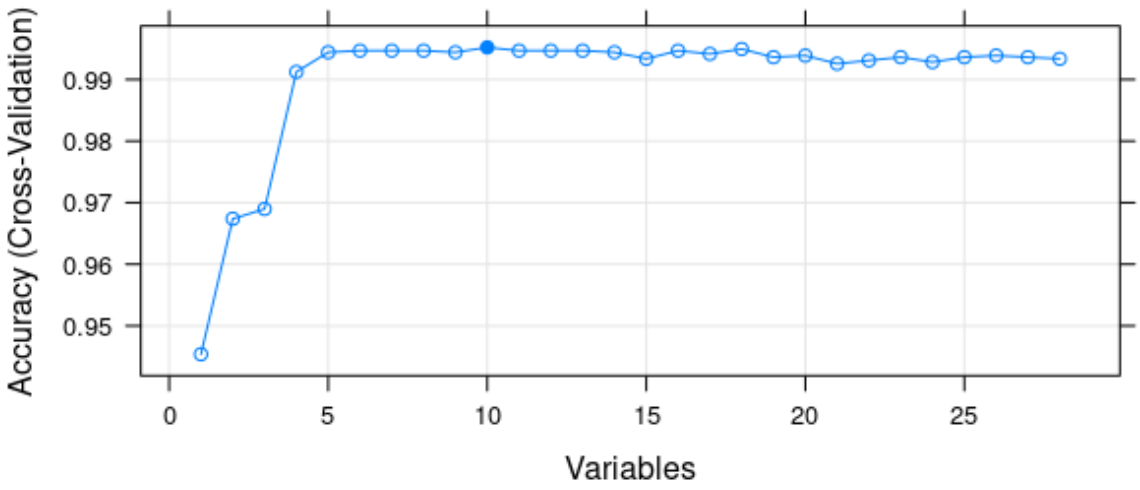
Part A: Decision Tree

a. For missing values there're several techniques, to use any technique we've to look at the data first to check which one is more suitable.

   1st: convert the '?' to 'NA's to be able to deal with them.
   2nd:

   - 'Age' column → only 1 value is missing → Drop this value 'row' (as it won't affect much since it's 1 value amidst over 3000 data point)
   - 'TBG' column → all of its values are NAs → Drop the whole column (as we don't have any information about it, and it's not applicable to infer its value from other columns)
   - Other columns that have missing values are numeric and chrs, we change them to numeric and factors then use **MICE** for imputation. 'Sex' column can be imputed by mice, method 'logreg' as it's binary. Numeric columns can be imputed by method 'norm' [1]

b. Attribute selection: one of the methods is RFE (Recursive Feature Elimination) [2] where it iterates over the data to evaluate all possible subsets of the attributes using random forest and then return to you the best combination/informative features.



In this graph, we can see that the accuracy for different number of features was nearly maximum with 10 features, however, it seems that

using just 5 features will be sufficient and yield high accuracy. Hence, we choose these 5 features only from the data; which are:

```
"TSH" "TSH_measured" "on_thyroxine" "FTI" "TT4"
```

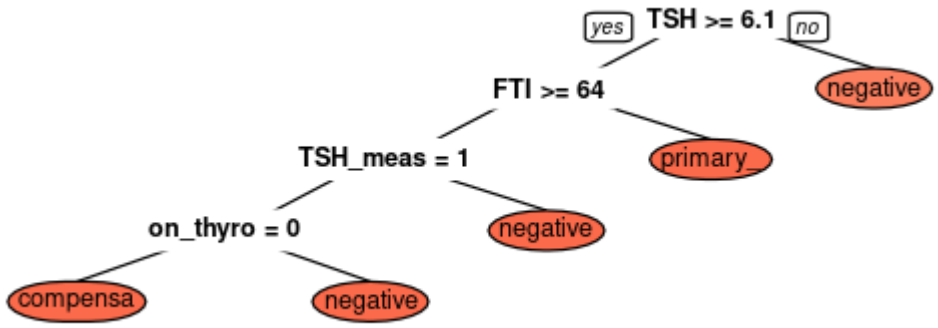
Importance of attribute selection: In data science, not all features comprise of the same importance, some features might be irrelevant to the data/problem so they might affect your problem negatively. Sometimes you find redundant information in two columns, e.g. when 2 columns have strong correlation (they're redundant, may mislead the model or lead to overfitting). Sometimes, we need to give more attention to some features over other features as these contribute more to the decision.
All in all, it's always advisable to perform feature selection to [3]:
-focus only on the features that can solve your problem.
-Save you from the curse of dimensionality (a lot of dimensions/features), thus decrease training time.
-Prevent overfitting and increase generalization of the model.

➔ Before going to cross-validation, data was split into train and test sets then cv was performed on train set.

c. Original Decision tree:
   - 10-fold train accuracies →

```
> model$resample
    Accuracy     Kappa Resample
1  0.9840426 0.8966276   Fold02
2  0.9841270 0.8967542   Fold01
3  0.9867021 0.9096023   Fold03
4  0.9920635 0.9475704   Fold06
5  0.9946809 0.9620394   Fold05
6  0.9894180 0.9320877   Fold04
7  0.9946950 0.9644708   Fold07
8  0.9920424 0.9475198   Fold10
9  0.9973545 0.9825065   Fold09
10 0.9920424 0.9474638   Fold08
> View(datatest)
```

d. Tree →



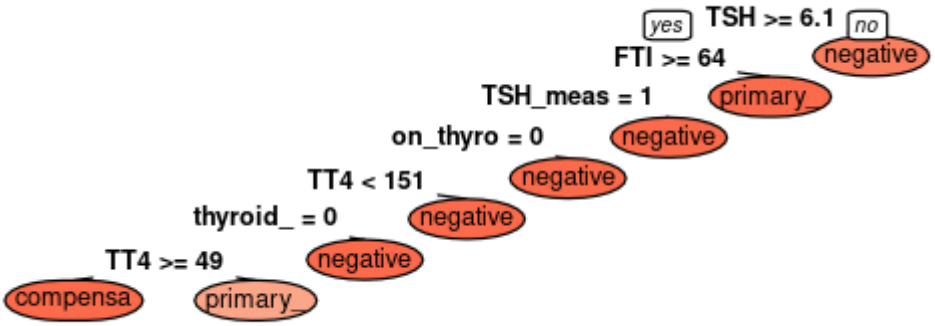
Interpret tree: it seems that 'TSH' is the first discriminator where all values of less than 6.1 are considered definitely as 'negative' class. Then, the tree starts to split based on other attributes, if 'TSH' >= 6.1, FTI comes as a 2nd hand for discrimination, if FTI < 64 then it's 'primary' class, however if it's >= 64 then we need more investigation, this is where other attributes like 'TSH_measurement' and 'on_thyroxine' come to action to complete the discrimination.

Evaluation of Tree: (confusion matrix)

```
Confusion Matrix and Statistics

          Reference
Prediction    1    2    3    4
         1  158   17    7    0
         2    0 2932    0    2
         3    0    6   73    0
         4    0    0    0    0

Overall Statistics

               Accuracy : 0.99
                 95% CI : (0.9859, 0.9931)
    No Information Rate : 0.9249
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.932

 Mcnemar's Test P-Value : NA

Statistics by Class:

                     Class: 1 Class: 2 Class: 3 Class: 4
Sensitivity           1.00000   0.9922  0.91250 0.000000
Specificity           0.99210   0.9917  0.99807 1.000000
Pos Pred Value        0.86813   0.9993  0.92405      NaN
Neg Pred Value        1.00000   0.9119  0.99775 0.999374
Prevalence            0.04945   0.9249  0.02504 0.000626
Detection Rate        0.04945   0.9177  0.02285 0.000000
Detection Prevalence  0.05696   0.9183  0.02473 0.000000
```

e. Different ways and their confusion metrices

- Different split strategy: **Gini**

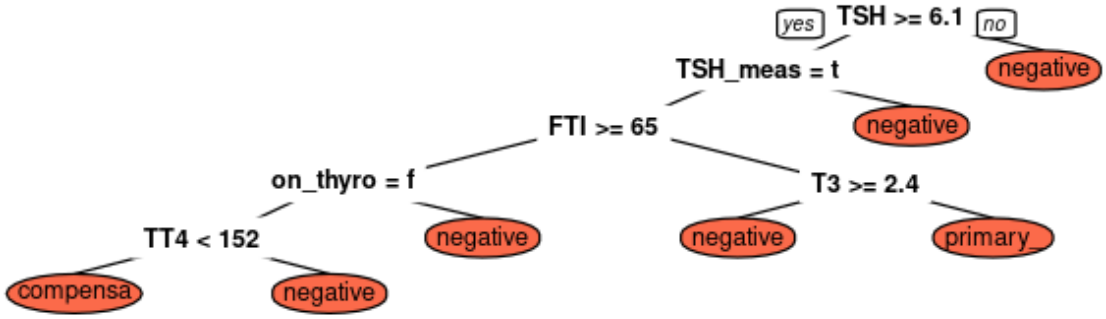



```
> model_gini$resample
    Accuracy     Kappa Resample
1  0.9900990 0.9359363   Fold02
2  0.9834437 0.8950952   Fold07
3  0.9966667 0.9764021   Fold01
4  0.9966887 0.9782138   Fold05
5  0.9933333 0.9537323   Fold06
6  0.9966887 0.9782264   Fold10
7  0.9933555 0.9537918   Fold04
8  0.9867550 0.9127041   Fold08
9  0.9767442 0.8502062   Fold09
10 1.0000000 1.0000000   Fold03
>
```

```
Confusion Matrix and Statistics

          Reference
Prediction    1    2    3    4
         1  157    1    0    0
         2    0 2947    0    2
         3    1    7   80    0
         4    0    0    0    0

Overall Statistics

               Accuracy : 0.9966
                 95% CI : (0.9938, 0.9983)
    No Information Rate : 0.9249
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.976

 Mcnemar's Test P-Value : NA

Statistics by Class:

                     Class: 1 Class: 2 Class: 3 Class: 4
Sensitivity           0.99367   0.9973  1.00000 0.000000
Specificity           0.99967   0.9917  0.99743 1.000000
Pos Pred Value        0.99367   0.9993  0.90909      NaN
Neg Pred Value        0.99967   0.9675  1.00000 0.999374
Prevalence            0.04945   0.9249  0.02504 0.000626
Detection Rate        0.04914   0.9224  0.02504 0.000000
Detection Prevalence  0.04945   0.9230  0.02754 0.000000
Balanced Accuracy     0.99667   0.9945  0.99872 0.500000
Warning messages:
```

Here we can observe that as the tree went into deeper nodes via gini, it had better accuracy.

- Pruning (rpart and tuning)



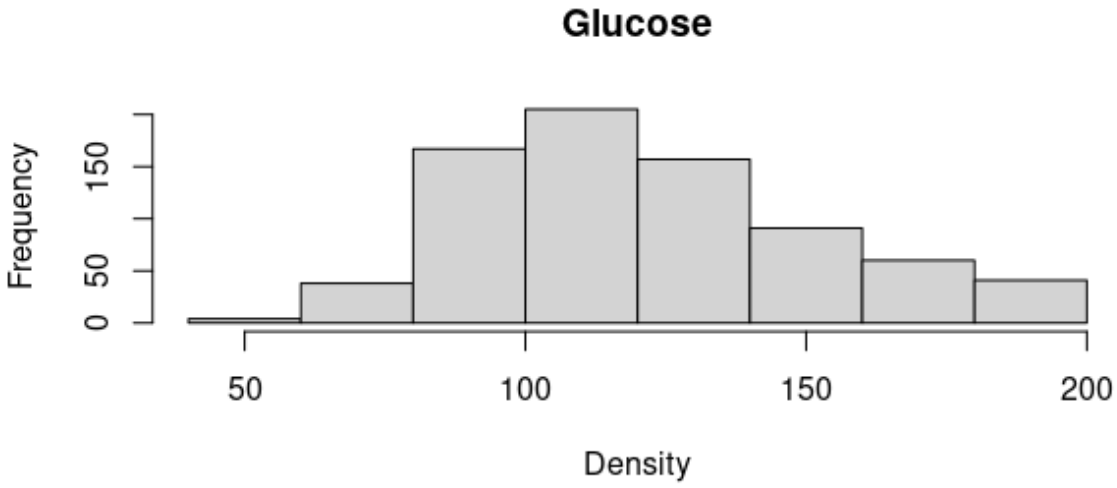
```
> confusionMatrix( as.factor(unclass(datatest$class)), as.factor(pred3)
Confusion Matrix and Statistics

          Reference
Prediction   1   2   3   4
         1  35   2   1   0
         2   1 694   2   1
         3   0   0  19   0
         4   0   0   0   0

Overall Statistics

               Accuracy : 0.9907
                 95% CI : (0.981, 0.9963)
    No Information Rate : 0.9219
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.9359

 Mcnemar's Test P-Value : NA

Statistics by Class:

                     Class: 1 Class: 2 Class: 3 Class: 4
Sensitivity           0.97222   0.9971  0.86364 0.000000
Specificity           0.99583   0.9322  1.00000 1.000000
Pos Pred Value        0.92105   0.9943  1.00000      NaN
Neg Pred Value        0.99861   0.9649  0.99592 0.998675
Prevalence            0.04768   0.9219  0.02914 0.001325
Detection Rate        0.04636   0.9192  0.02517 0.000000
Detection Prevalence  0.05033   0.9245  0.02517 0.000000
Balanced Accuracy     0.98402   0.9647  0.93182 0.500000
```

However, here with using rpart/other way for pruning, the accuracy didn't improve much.

Part B: Support Vector Machine

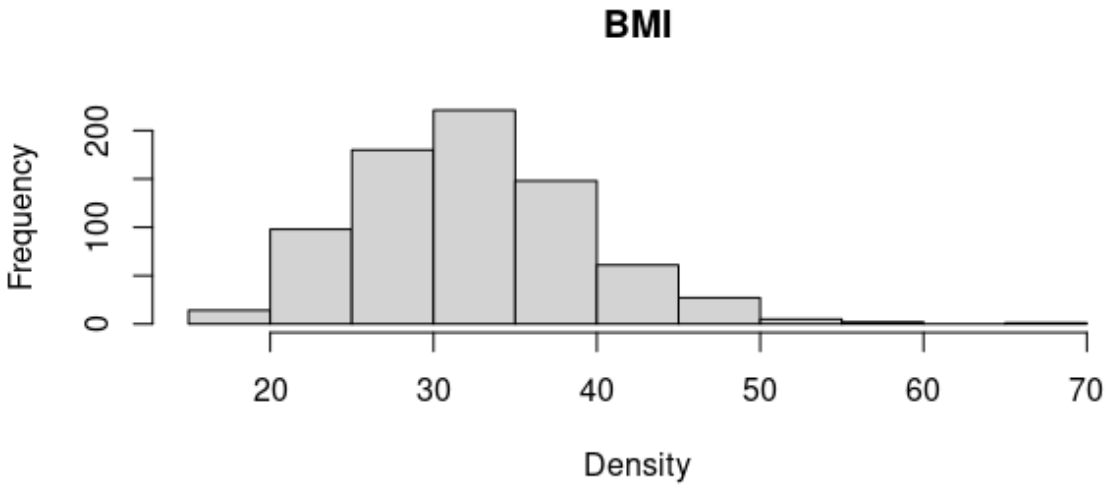
a. Imputation: using the central tendency means that we need to look at the distribution of each column, and based on the distribution we choose whether to impute by mean or median

- 'Glucose'



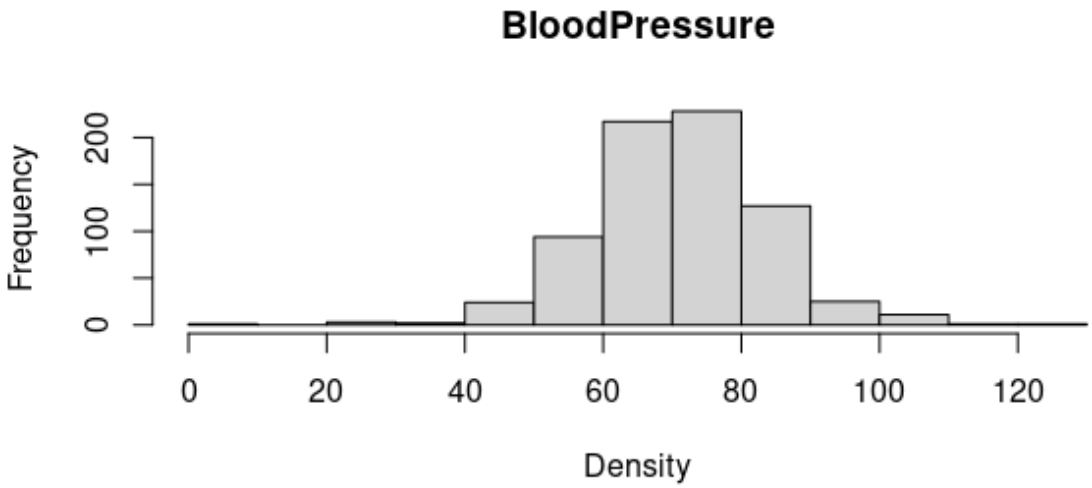
It seems as a normal distribution → then we impute it by Mean

- 'BMI'



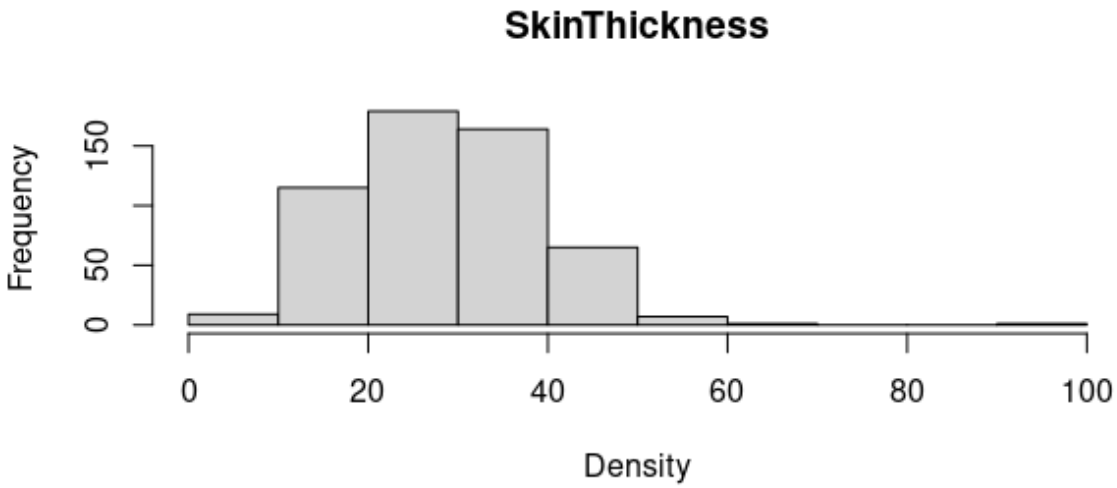
BMI seems to have more outliers and more skewed → then we impute it by Median.

- ‘BloodPressure’



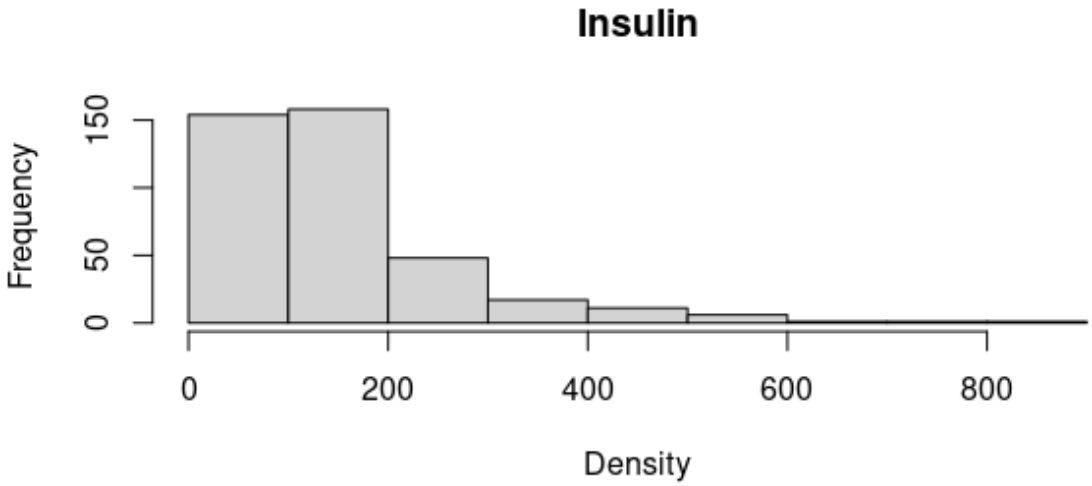
Pressure might be more of symmetric/normal one → impute by Mean.

- ‘SkinThickness’



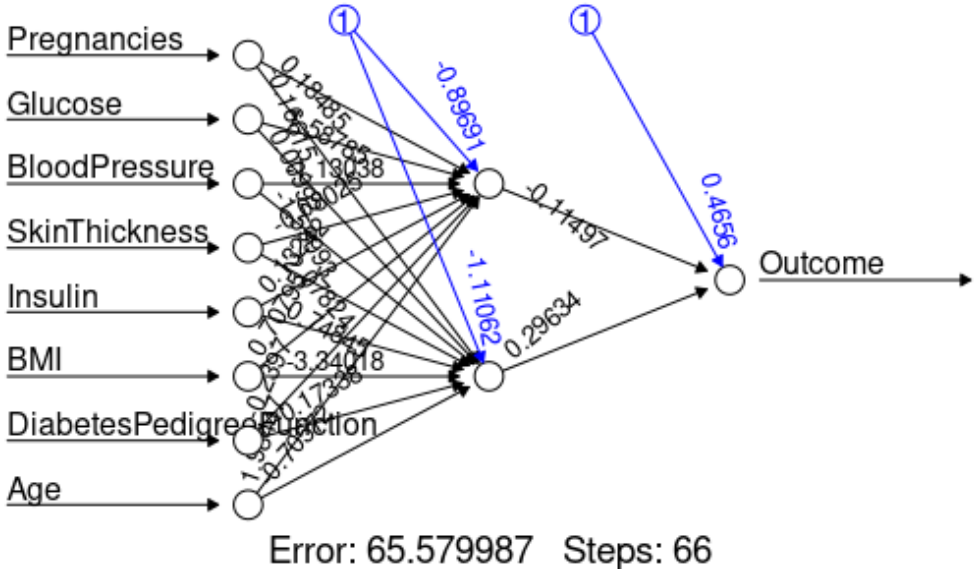
Skin is more likely to be skewed → impute by Median**.**

- ‘Insulin’

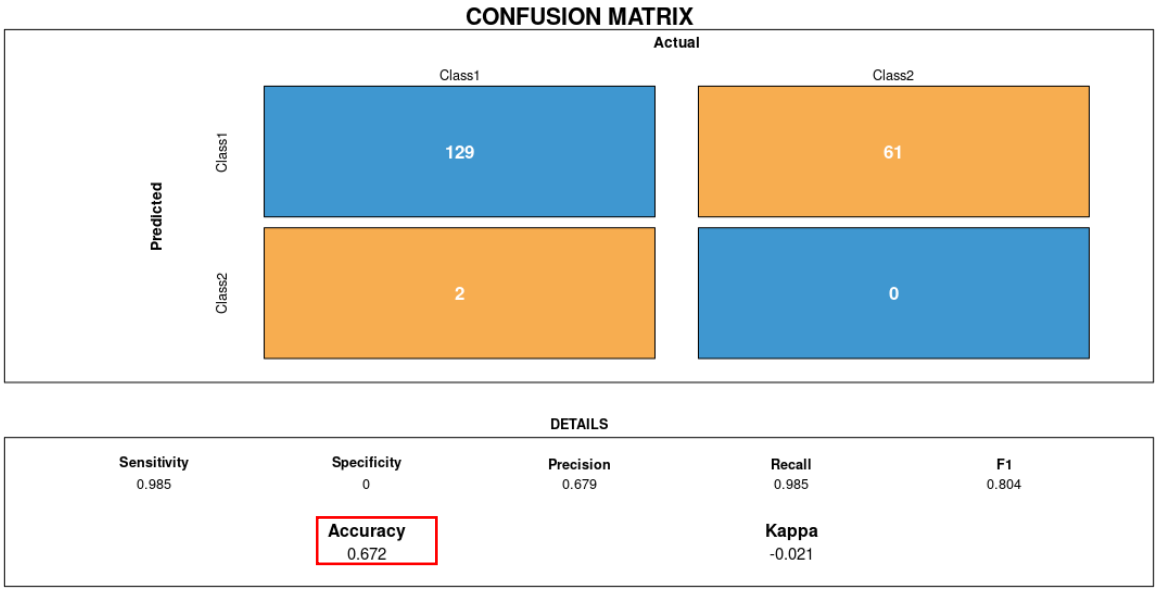


Definitely Insulin is skewed → impute by Median.

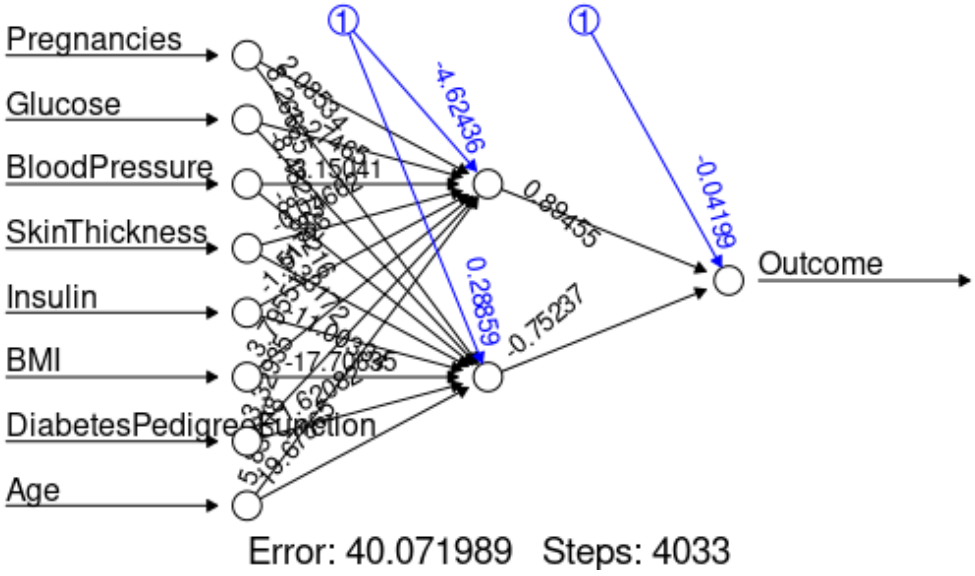
b. Splitting the data into train and test by ratio of 75% - 25%
Then, training on data without any hyperparameter tuning.

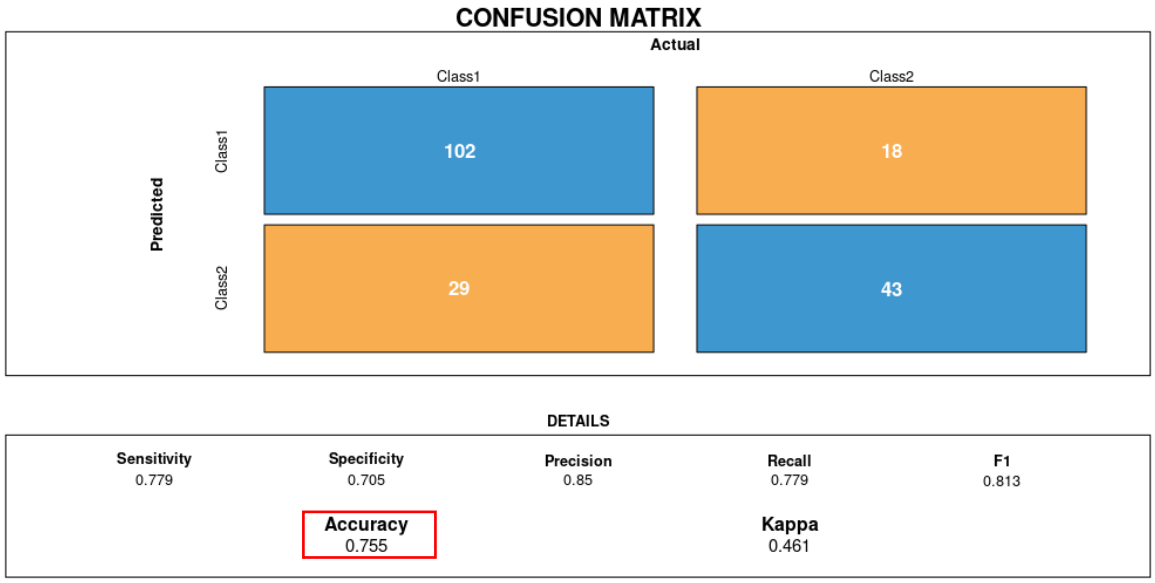


Confusion Matrix → **Accuracy: 67.2 %**



c. **Scaling** by min-max scale, gives us this NN which yields a confusion matrix as follows, **Accuracy: 75.5 %**

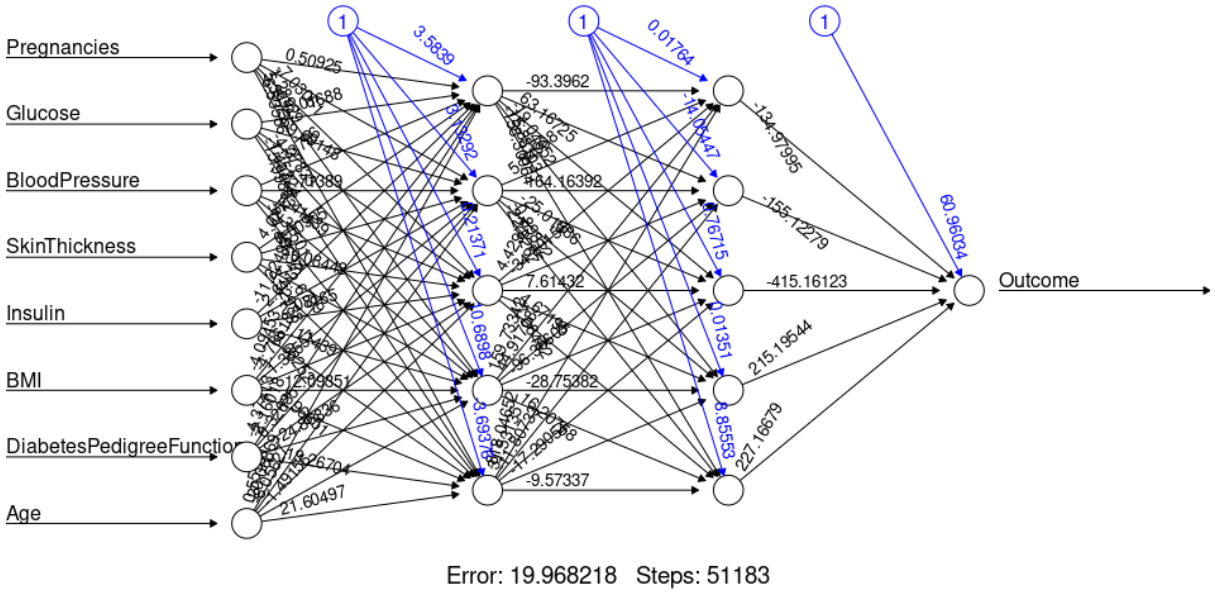


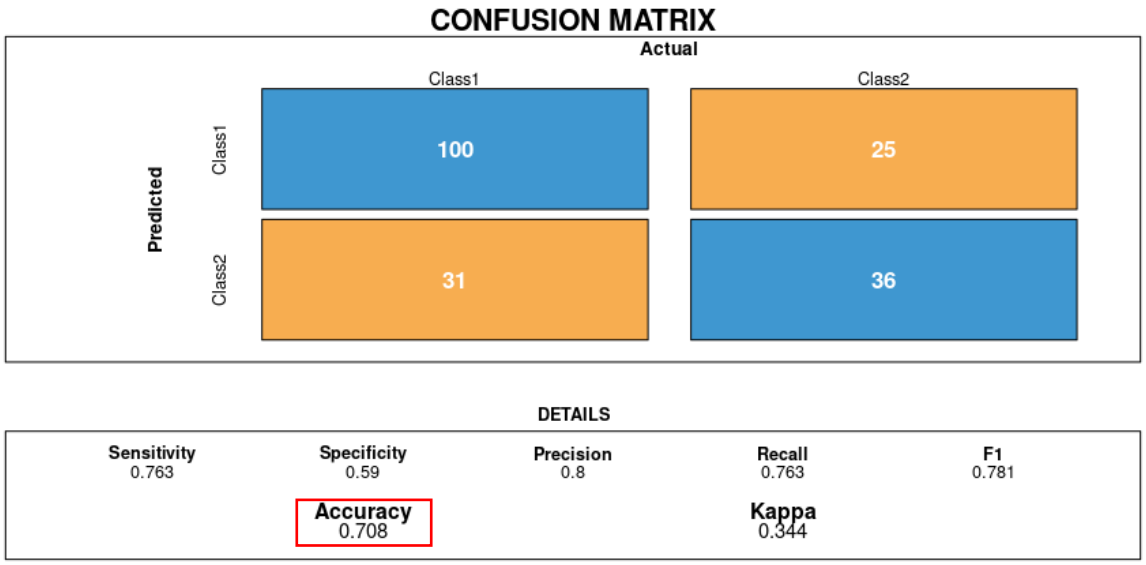


Apparently, this tells how scaling is **significant** for neural networks.

d. For scaling and 2 hidden nodes → this is the output of number **'C'** with scaling, and number **'B'** without scaling.

However, for 2 layers with 5 nodes each: **Accuracy** degraded→ **70 %**





As we use more layers in the neural network with more hidden nodes, the accuracy degrades along with other metrics as the model misclassifies more. Thus, as **simple** as better, as we simplify the model it performs better.

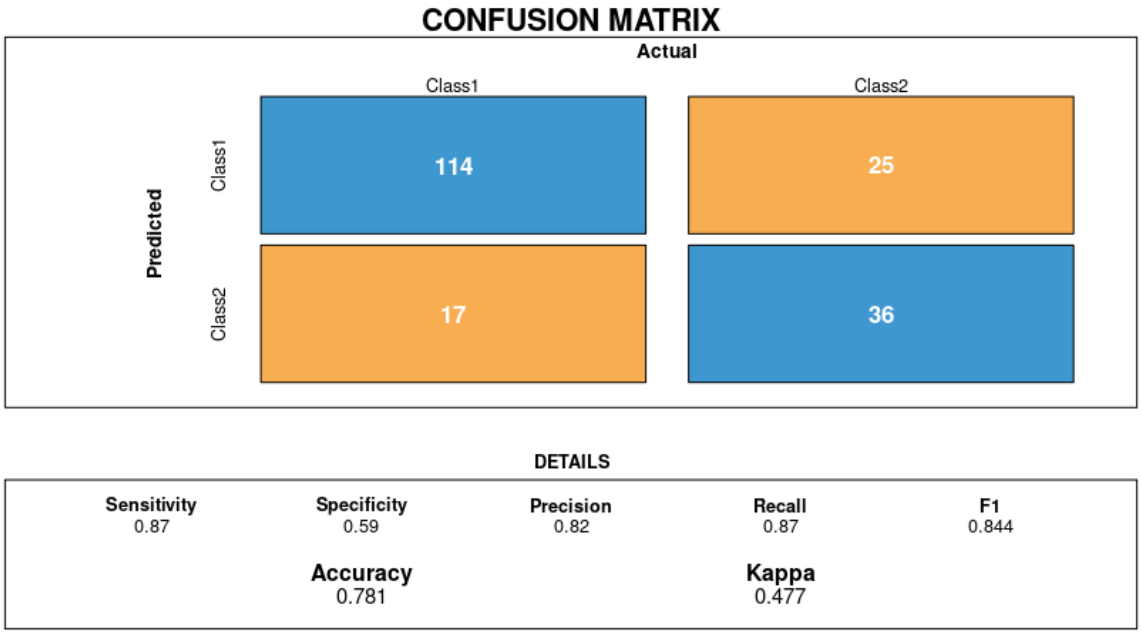
e. For changing the hyperparameters, we will work with the best data we got up until now, which is the scaled data and 2 nodes only for 1 layer.

- **Activation Function** → Tanh: Accuracy increases more → 78%, This means that adding some sort of non-linearity to the NN might yield better performance.

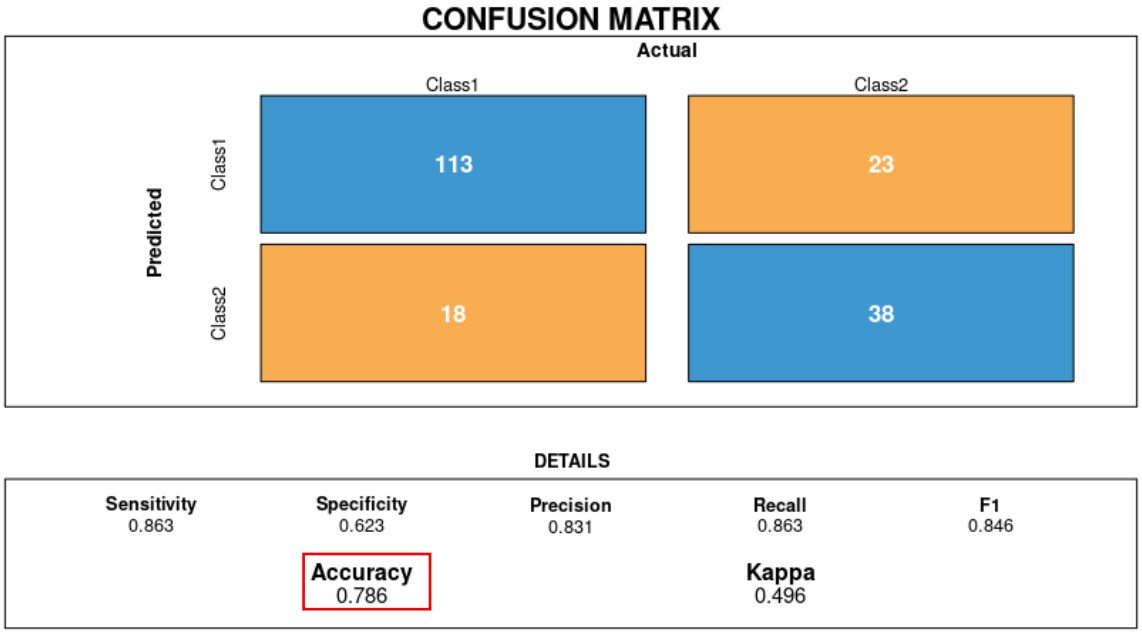
f. **Learning rate** → 0.01

The learning rate didn't change much as LR is effective more in the backward network not the feed-forward one.

Thus, the accuracy didn't change, nevertheless slight changes in other metrics have occurred.



g. Changing number of **epochs** → when I tried changing it to 5 or 10 epochs it didn't perform well, this could mean that the model was about to overfit so we have to stop training earlier, thus for **3** epochs the performance was slightly better.



References:

[1] Handling missing data with MICE package; a simple approach | DataScience+ (datascienceplus.com)

[2] Feature Selection with the Caret R Package (machinelearningmastery.com)

[3] The Practical Importance of Feature Selection - KDnuggets