**Objective**

To predict whether the employee will leave the company or not based on the attributes like Age, Work life balance, Job satisfaction and many more (34 attributes).

**Dataset**

The dataset has 2940 obs. of 35 variables. Their frequency of data types is as following,

factor integer

9 26

**Exploratory Data Analysis**

The dataset has divided into 7:3 (development and holdout). The following are the exploratory analysis done on the development dataset.

* All categorical variables (factor) should have at least two levels, if not that are not contributing any significant information about the attrition (target).

[1] Over18 has only 1 category and its field id is 22

* Similarly, all continuous values should have at least two values.

[1] All the values of EmployeeCount are : 1 and its ID is : 9

[1] All the values of StandardHours are : 80 and its ID is : 27

* The primary key of the dataset also doesn’t help to build the model because every value is unique. In our case the field is ‘'Employeenumber'’.
* The data set contains no missing values.

From the above points, we have concluded to remove the non-significant variables and they are 'Over18', 'EmployeeCount', 'StandardHours', 'Employeenumber'.

**Neural Networks**

Neural Network doesn’t have the ability to interpret categorical variables (factor), in order to handle them we have to either code them numerically or replace the factor variables with dummy variables. We have followed the later one and below is a sample illustration for one factor variable

> levels(Integer\_dataset$BusinessTravel)

[1] "Non-Travel" "Travel\_Frequently" "Travel\_Rarely"

Here it’s having three levels, we will remove this factor variable with two binary numeric dummy variables, 1 representing the employee belong to this category and 0 as not this category. If both of these dummy variables are 0 then that represents the employee belongs to the third category which is not represented as a separate dummy variable. Below are the new variables replacing ‘BusinessTravel’.

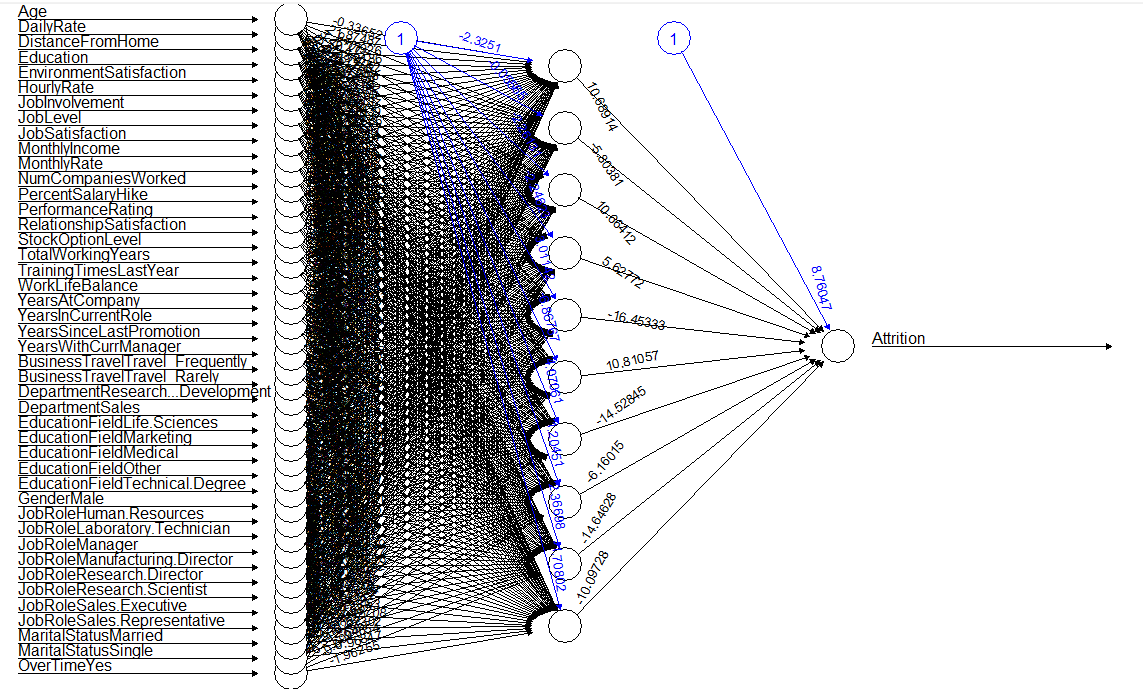
[32] "BusinessTravelTravel\_Frequently"

[33] "BusinessTravelTravel\_Rarely"

Similarly, all other factors are coded.

Neural Networks works under converging techniques, if the various variable values are of highly different range there are changes that it might take many iterations before converging to the solutions. So scaling is applied on all the independent variables.

Finally, a neural network is built for 45 variable dataset with sigmoidal activation function and one hidden layer with ten neurons. The output was as follows

  
The results on development data is a follows

> table

dev\_NN\_pred

0 1 Sum

0 1726 0 1726

1 29 303 332

Sum 1755 303 2058

> dev\_NN\_acc

[1] 0.9859086492

On holding data it is a follows

> table

holdout\_NN\_pred

0 1 Sum

0 722 18 740

1 33 109 142

Sum 755 127 882

> hold\_NN\_acc

[1] 0.9421768707

As there is not much drop in accuracies and being very high, we are not further tuning the model.

**Random Forest**

Initial random forest is built to have a large number of trees with mtry close to square root of variables and node size as 2% of the development dataset population. The results are as following

Call:

randomForest(x = Dev\_data[-2], y = Dev\_data$Attrition, ntree = 5000, mtry = 6, nodesize = 40, data\_set = Dev\_data)

Type of random forest: classification

Number of trees: 5000

No. of variables tried at each split: 6

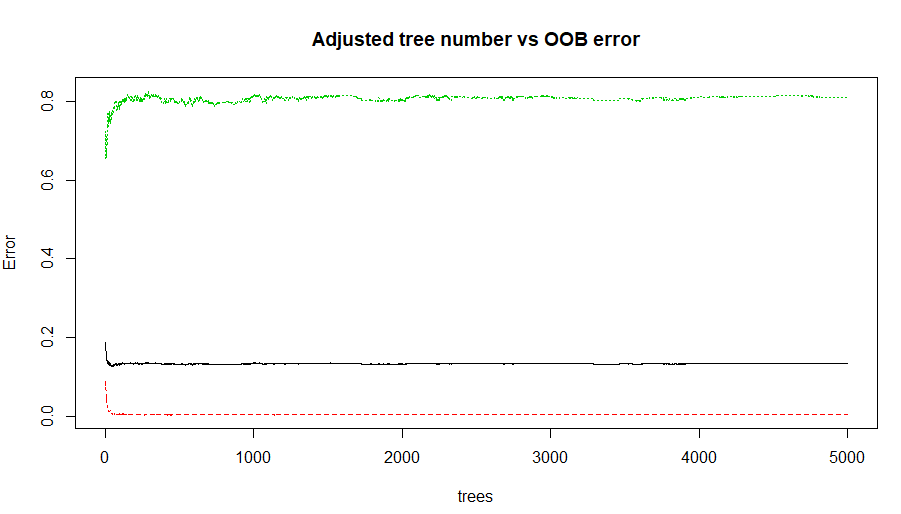
OOB estimate of error rate: 13.31%

Confusion matrix:

No Yes class.error

No 1721 5 0.002896871379

Yes 269 63 0.810240963855



From the figure we are clear that post 500 there is no significant change in OOB error, from this we can conclude that the optimal range of ntrees would be around 500.

Next step is to optimize the mtry value, for this we have tried tuneRF technique and gridSearch technique following were their respective results

tuneRF

Call:

randomForest(x = x, y = y, mtry = res[which.min(res[, 2]), 1], nodesize = 40, importance = ..3, data\_set = ..1)

Type of random forest: classification

Number of trees: 500

No. of variables tried at each split: 19

OOB estimate of error rate: 11.86%

Confusion matrix:

No Yes class.error

No 1700 26 0.01506373117

Yes 218 114 0.65662650602

girdSearch

Random Forest

2058 samples

30 predictor

2 classes: 'No', 'Yes'

No pre-processing

Resampling: Bootstrapped (25 reps)

Summary of sample sizes: 2058, 2058, 2058, 2058, 2058, 2058, ...

Resampling results across tuning parameters:

mtry Accuracy Kappa

2 0.9183011333 0.6216322713

16 0.9189547080 0.6403694114

30 0.9183666896 0.6423959082

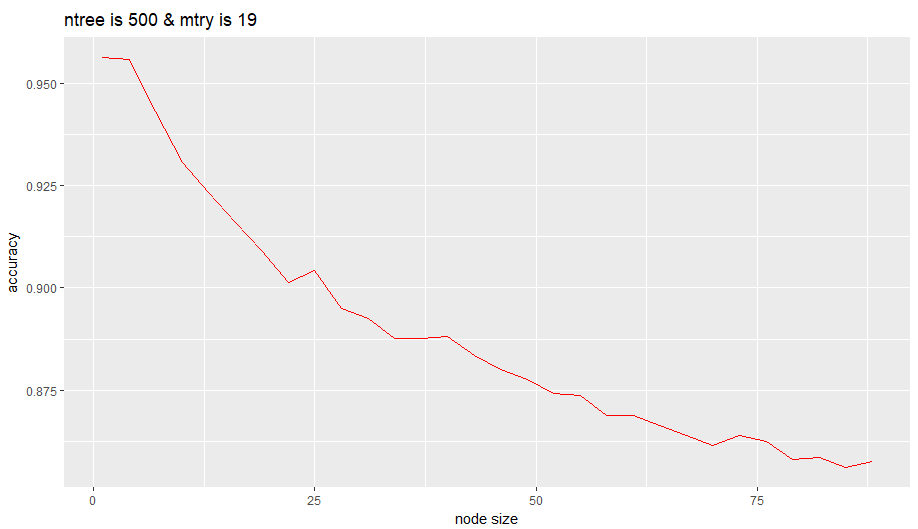
Accuracy was used to select the optimal model using the

largest value.

The final value used for the model was mtry = 16.

This concludes that mtry optimal value would be around 16-19, we had chosen to go with tuneRF result i.e. 19.

Next variable to optimize is the node size, intuitively we could say the less is the best for classification in random forest because here overfitting will not happen due to long trees and also the default value in random forest for classification is 1. To strengthen our intuition, we ran the random forest for all node sizes from 1 to 100 and the following is the plot against OOB error



It’s very clear, the increase in node size results in compromising OOB rate. So we have chosen node size as 2.

Finally, the optimised random forest result was as follows

Call:

randomForest(x = Dev\_data[-2], y = Dev\_data$Attrition, ntree = 500, mtry = 19, nodesize = 2, data\_set = Dev\_data)

Type of random forest: classification

Number of trees: 500

No. of variables tried at each split: 19

OOB estimate of error rate: 4.28%

Confusion matrix:

No Yes class.error

No 1719 7 0.00405561993

Yes 81 251 0.24397590361

> Confusion\_Matrix\_RF

Prediction

actual No Yes Sum

No 734 6 740

Yes 22 120 142

Sum 756 126 882

> Accuracy\_RF

[1] 96.82539683

**Ensemble Model**

In classification ensemble is done on voting, but in our case we are having only two models and most of the cases would fail under tie. So we considered to take the probabilities and average them and class based on the averaged probabilities. This will be done on the development sample and same cut-off probability will be taken to classify the holding data.

Following are the results for development data at cut-off probability as 0.3

> table

0 1 Sum

0 1726 0 1726

1 4 328 332

Sum 1730 328 2058

> dev\_Ensemble\_acc

[1] 0.9980563654

On holding data set

> table

0 1 Sum

0 718 22 740

1 16 126 142

Sum 734 148 882

> hold\_Ensemble\_acc = (table[1]+table[5])/table[9]

> hold\_Ensemble\_acc

[1] 0.9569160998

**Conclusion**

|  |  |  |
| --- | --- | --- |
| **Model** | **Accuracy on dev** | **Accuracy on holding** |
| Random Forest | 95.7 | 96.8 |
| Neural Network | 98.6 | 94.2 |
| Ensemble model | 99.8 | 95.7 |

In general ensemble modelling is more stable than individual model and in our case we expect the ensemble model to handle both bias (which can be handled by NN) and variance (which can be handled by RF) because it is a combination of RF and NN.