ISEN 613-Fall 2018 Course Project

PROJECT REPORT ON CAR INSURANCE CLAIMS

Harshavarshan Rao Chunduri (727004888)
Rohit Birhade (427009769)
Mohammed Tuqeer Muqtadar (827001924)
Srivanth Navaneethakrishnasamy (227003041)

EXECUTIVE SUMMARY

To: CEO of Insurance Company
From: Our Team
Subject: Analysis and Prediction of Insurance claim by car owners

As a complete unit of hard-working and diligent people, we thank you for giving us an opportunity to undertake the project for Analysis and prediction of insurance claim instances. We were extremely happy when your company associates approached us for helping your company in achieving this goal. It was a wonderful experience working alongside your team who were up on their heels when asked any queries regarding the project.

In our proposal we have employed Neural Network model to predict the insurance claim instances from the 50,000 data points provided for the test data. From the initial observations of the relations, we can establish that the most important or dominant variable of all the variables provided is **Var7** and the second most influencing variable from the given set is **Vehicle.** It is furthermore observed that, the beforementioned variables are directly proportional to the chances of insurance being claimed which means that with a higher value of either of the two most important variables the chances of a particular house hold claiming insurance becomes more likely. And on the other hand, **Var4** has the maximum negative effect on the insurance claim status. That is, with the increase in the magnitude of **Var4** the chances of insurance being claimed becomes highly unlikely

Considering the critical factors that determine the insurance claim status, model with best fit is obtained by removing the attributes with least relevance on insurance claim, eliminating some of the outlying values that give false representation on the claims and reducing the model error to the possible minimum. The final model given in the technical summary backed by appropriate validation plots is depicted in the technical summary of report. Conclusively, I would like to mention that there is always a more accurate model but the art of analyzing the data is knowing where to stop! I hope this report and its findings help you and your team with your claim prediction. We would be happy to assist you and your team regarding any queries and also look forward to working with you in the future.

Contribution: The members of the group contributed equally with each of us performing all the tasks assigned by ourselves.

Thank you,
Yours Sincerely
Srivanth Navaneethakrishnasamy
Harshavarshan Rao Chunduri
Rohit Birhade
Mohammed Tuqeer Muqtadar

RANDOM FOREST TECHNIQUE

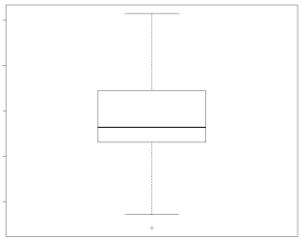
Random forest technique is initialized with the 100,000-data point given in the problem statement. The following processes were carried out to obtain a result from the above-mentioned process.

1. **Data Cleaning:** The raw data given initially had numerous missing values which were represented by question marks ('?'). Thus, the initial task was to replace the question marks with "NA" so R recognizes the missing values. Since working with the missing data will lead to a drop in the accuracy of the test result, replacing or removing them is the ideal choice. We prefer imputation since removal of data might lead to increased test error and this can either be done by imputing Mean, mode or median. Imputation using mean is performed (using impute() from HMISC package) since the raw data set given contains character variables. We can check the number of missing values in each variable using the following function

```
> colSums(is.na(data))
  Household_ID        Vehicle  Calendar_Year     Model_Year     Blind_Make    Blind_Model
             0              0              0              0             48             48
Blind_Submodel           Cat1           Cat2           Cat3           Cat4           Cat5
            48            210          39683             40          42309          42365
          Cat6           Cat7           Cat8           Cat9          Cat10          Cat11
           210          54306             37              0             30            245
         Cat12         OrdCat           Var1           Var2           Var3           Var4
             0             44              0              0              0              0
         Var5           Var6           Var7           Var8          NVCat         NVVar1
             0              0              0              0              0              0
       NVVar2         NVVar3         NVVar4   Claim_Amount
             0              0              0              0
```

It can be seen on further observation that, 'Make', 'Model' and 'Sub-model' have missing values corresponding to identical observations. Thus, these observations are removed since they increase the bias. Once the missing values have been removed, we now look for outliers using boxplots.

To the left we can see the boxplot with outliers and to the right we observe the boxplot without any and this achieved by setting a benchmark value of 3rd quartile + 1.5 (Inter-quartile range). And replacing the values higher than benchmark to benchmark.



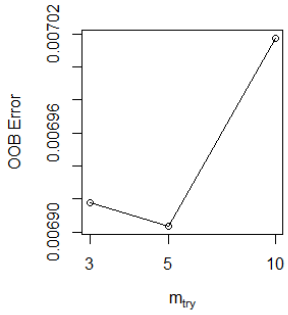
After performing the above operations, we have a data set that can be considered as clean. We now proceed with the formulation of model.

2. **Model Formulation:** The variables, 'Make', 'Model' and 'Sub-model' when treated as categorical variable, have more than 55 levels; making it impossible to use in Random Forest technique. Hence, we convert them to numeric type by using **as.numeric** function. We then sample the data into test and train of which, 70% is allocated to train data. We now use randomForest() present in RandomForest package which takes input in the form of number of trees (ntree) and number of variables (mtry). We use tuneRF() present in the same package to find out the optimum value of 'mtry'. Since the number of observations is high, we use improve=0.1 and stepFactor is taken as 2.

```
mtry=tuneRF(train[,-34],train$C_claim,
            stepFactor = 2, improve = 0.1,
            trace = T,plot=T)
```

On the left we have the tuneRF function where in at each iteration, mtry is inflated by 2 and the relative improvement in OOB error must be by 0.1.

On running this command, we obtain an optimal value of mtry as 5 for which OOB error is the least as shown in the plot below.



We then run the "RandomForest" function using mtry=5 and ntree=200 where in the values have been taken in accordance to the data size.

rf=randomForest(C_claim~.,data=train,mtry=5,ntree=200)

When we predict using the test sample, we get the results as shown below:

```
      predicted
actual     0     1
     0 29750     0
     1   236     0

               Accuracy : 0.9921297
                 95% CI : (0.9910637, 0.9930986)
    No Information Rate : 1
    P-Value [Acc > NIR] : 1

                  Kappa : 0
 Mcnemar's Test P-Value : <0.0000000000000002

            Sensitivity : 0.9921297
            Specificity :        NA
         Pos Pred Value :        NA
         Neg Pred Value :        NA
             Prevalence : 1.0000000
         Detection Rate : 0.9921297
   Detection Prevalence : 0.9921297
      Balanced Accuracy :        NA
```

We can observe from the results that the false positive rate is high, leading to a loss to the insurance firm. The primary reason for this result is that the data is **highly unbalanced**. Thus we cannot end the model here owing to further modification being made.

0	1
99233	719

In the given data set, the number of claims is very much smaller than the non-claims which makes the data highly imbalanced.

We can overcome the error due to unbalanced data by sampling. We use "**both sampling"** method in order to incorporate both under and over sampling. This gives the best result compared to performing either of under or over sampling individually. We use "**ovun.sample()"** present in **ROSE** package and use **method=Both** as shown below.

```
both=ovun.sample(C_claim~.,data=train,method ="both",
                 p=0.5,seed=111,N=69966)$data
```

After performing the sampling function, we now use **tuneRF()** to find the optimal value of **"mtry"**. In this second attempt, we get the least **OOB** error at **mtry=10.** So we now run RandomForest using mtry=10 and ntree=200 as shown below.

```
      predicted
actual     0     1
     0 35050     1
     1     0 34915

               Accuracy : 1
                 95% CI : (0.9999, 1)
    No Information Rate : 0.501
    P-Value [Acc > NIR] : <2e-16

                  Kappa : 1
 Mcnemar's Test P-Value : 1

            Sensitivity : 1.000
            Specificity : 1.000
         Pos Pred Value : 1.000
         Neg Pred Value : 1.000
             Prevalence : 0.501
         Detection Rate : 0.501
   Detection Prevalence : 0.501
      Balanced Accuracy : 1.000

       'Positive' Class : 0
```

The output is attached on the left and it can be seen that after the sampling is performed, all the parameters like sensitivity, specificity, accuracy increase. Thus this model obtained is a better model than the one obtained previously.

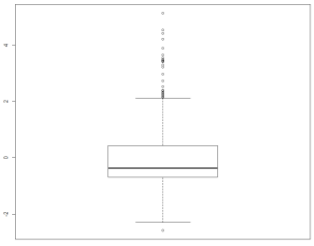
XG BOOSTING TECHNIQUE

Extreme Gradient(XGBoost) is similar to gradient boosting framework but more efficient. It has both linear model solver and tree learning algorithms. So what makes it fast is its capacity to do parallel computation in a single machine.

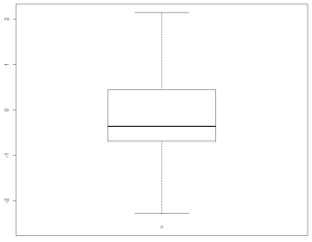
1. **Data Cleaning:** The raw data given initially had numerous missing values which were represented by question marks ('?'). Thus, the initial task was to replace the question marks with "NA" so R recognizes the missing values. Since working with the missing data will lead to a drop in the accuracy of the test result, replacing or removing them is the ideal choice. We prefer imputation since removal of data might lead to increased test error and this can either be done by imputing Mean, mode or median. Imputation using mean is performed (using impute() from HMISC package) since the raw data set given contains character variables. We can check the number of missing values in each variable using the following function

```
> colSums(is.na(data))
  Household_ID        Vehicle  Calendar_Year     Model_Year     Blind_Make    Blind_Model
             0              0              0              0             48             48
Blind_Submodel           Cat1           Cat2           Cat3           Cat4           Cat5
            48            210          39683             40          42309          42365
          Cat6           Cat7           Cat8           Cat9          Cat10          Cat11
           210          54306             37              0             30            245
         Cat12         OrdCat           Var1           Var2           Var3           Var4
             0             44              0              0              0              0
          Var5           Var6           Var7           Var8          NVCat         NVVar1
             0              0              0              0              0              0
       NVVar2         NVVar3         NVVar4   Claim_Amount
             0              0              0              0
```

It can be seen on further observation that, 'Make', 'Model' and 'Sub-model' have missing values corresponding to identical observations. Thus, these observations are removed since they increase the bias. Once the missing values have been removed, we now look for outliers using boxplots.



To the left we can see the boxplot with outliers and to the right we observe the boxplot without any and this achieved by setting a benchmark value of 3rd quartile + 1.5 (Inter-quartile range). And replacing the values higher than benchmark to benchmark.



After performing the above operations, we have a data set that can be considered as clean. We now proceed with the formulation of model.

2. **Model Formulation:** We sample the given data set into Test and Train data sets. Subsequently we create a matrix and one-hot encoding dummy variable. One-hot encoding creates a dummy variable for factor variables to convert data into numeric format.
   The matrix which we created hold all the variable data in the numeric format since **XG Boosting** only works with numeric vectors.
   We use **sparse.model.matrix()** to create a matrix as shown below:

```
trainm=sparse.model.matrix(C_claim~.,data=train)
head(trainm)
train_label=train[,"C_claim"]
train_label
train_matrix=xgb.DMatrix(data=as.matrix(trainm),
                         label=train_label)
train_matrix

testm=sparse.model.matrix(C_claim~.,data=test)
head(testm)
head(test)
test_label=test[,"C_claim"]
test_label
test_matrix=xgb.DMatrix(data=as.matrix(testm),
                        label=test_label)
test_matrix
```

We take only C_Claim from both test and train and put them in appropriate labels as shown in the code alongside. And then we determine the number of classes based upon the train_label as shown below:

```
nc=length(unique(train_label))
```

We can select the number of parameters in the tree using the list function as shown below:

```
nc=length(unique(train_label))
nc
xgb_parameters=list("objective"="multi:softprob",
                    "eval_metric"="mlogloss",
                    "num_class"=nc)
xgb_parameters

watchlist=list(train=train_matrix,test=test_matrix)
watchlist
```

Watch list is used to split the data set into train and test data into respective matrix form. The command for the same can be seen in the code alongside.

Initially we run the XG Boosting for 50 iterations; we then select the iteration with the lowest test error rate and run another model for the same number of iterations. We then predict the result using test_label as shown below:

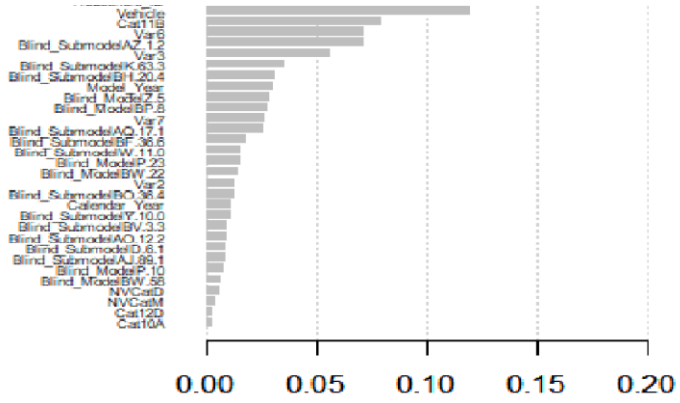
```
pred=matrix(predict,nrow=nc,
            ncol=length(predict)/nc) %>%
  t() %>% data.frame() %>%
  mutate(label=test_label,
         max_prob=max.col(.,"last")-1)
```

Using the actual values and predicted values, we can formulate the confusion matrix which is as shown below:

```
         predicted
actual     0     1
0        19850   137
1          0     3

Accuracy : 0.9931
95% CI : (0.9919, 0.9942)
No Information Rate : 0.993
P-Value [Acc > NIR] : 0.4213

Kappa : 0.0417
Mcnemar's Test P-Value : <2e-16

Sensitivity    : 1.00000
Specificity    : 0.02143
Pos Pred Value : 0.99315
Neg Pred Value : 1.00000
Prevalence     : 0.99300
Detection Rate : 0.99300
Detection Prevalence : 0.99985
Balanced Accuracy :    0.51071

'Positive' Class : 0
```

The output of the model is pasted alongside and it can be seen that we obtain a really high value of sensitivity. Even though the sensitivity is high, the specificity of the model has a very low value which makes this model not our priority.

In order to get an idea of the most important variables i.e the various variables that affect the output the most are identified. This is done using the **importance()** from the RandomForest package. As a result, we get the following output and thus the importance of variables:



As an obvious statement the car turns out to be the most important variable in our case where our output identifies if a vehicle claims insurance or not. Variable "Vehicle" is followed by variable "Cat11B" in the importance order.

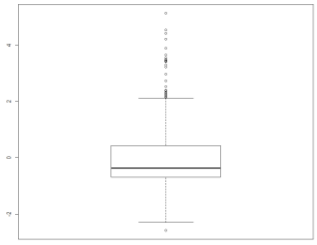
NEURAL NETWORK TECHNIQUE

Artificial Neural networks are forecasting methods that are based on simple mathematical models of the brain. They allow complex non-linear relationships between the response variables and its predictors.

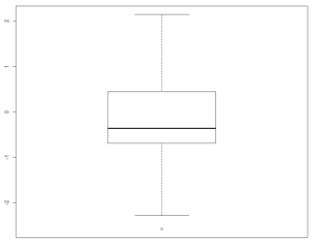
1. **Data Cleaning:** The raw data given initially had numerous missing values which were represented by question marks ('?'). Thus, the initial task was to replace the question marks with "NA" so R recognizes the missing values. Since working with the missing data will lead to a drop in the accuracy of the test result, replacing or removing them is the ideal choice. We prefer imputation since removal of data might lead to increased test error and this can either be done by imputing Mean, mode or median. Imputation using mean is performed (using impute() from HMISC package) since the raw data set given contains character variables. We can check the number of missing values in each variable using the following function

```
> colSums(is.na(data))
  Household_ID        Vehicle  Calendar_Year     Model_Year     Blind_Make    Blind_Model
             0              0              0              0             48             48
Blind_Submodel           Cat1           Cat2           Cat3           Cat4           Cat5
            48            210          39683             40          42309          42365
          Cat6           Cat7           Cat8           Cat9          Cat10          Cat11
           210          54306             37              0             30            245
         Cat12         OrdCat           Var1           Var2           Var3           Var4
             0             44              0              0              0              0
          Var5           Var6           Var7           Var8          NVCat        NVVar1
             0              0              0              0              0              0
       NVVar2         NVVar3         NVVar4   Claim_Amount
             0              0              0              0
```

It can be seen on further observation that, 'Make', 'Model' and 'Sub-model' have missing values corresponding to identical observations. Thus, these observations are removed since they increase the bias. Once the missing values have been removed, we now look for outliers using boxplots.



To the left we can see the boxplot with outliers and to the right we observe the boxplot without any and this achieved by setting a benchmark value of 3rd quartile + 1.5 (Inter-quartile range). And replacing the values higher than benchmark to benchmark.



After performing the above operations, we have a data set that can be considered as clean. We now proceed with the formulation of model.

2. **Model Formulation:** Neural network works on the numeric data. Hence, we need to convert the variables into numeric form before using it in the Neural Network model. We can use **model.matrix()** to create numeric matrix as shown below.

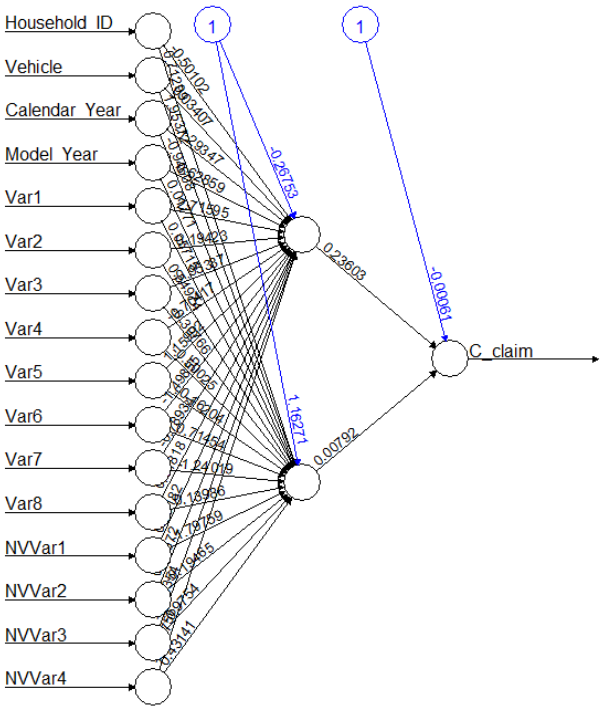
```
m = model.matrix( ~C_claim+Household_ID+Vehicle+Calendar_Year+Model_Year+
    Blind_Make+Blind_Model+Blind_Submodel+Var1+Var2+Var3+Var4+
    Var5+Var6+Var7+Var8+NVVar1+NVVar2+NVVar3+NVVar4, data= train)
```

The numeric matrix is stored in 'm' which now serves as the data for the Neural Network model. We then formulate the model using **neuralnet()** which is present in **neuralnet** package.

The function takes variables as input along with the value for **'hidden'** which represents the number of hidden layers between the input and output. One major syntax modification that we make in the command is that, we use **'+'** instead of **','.** We use **hidden=2** for large data sets like the one given to us and a threshold value of 0.01 as shown below

```
model1=neuralnet(C_claim~Household_ID+Vehicle+Calendar_Year+Model_Year+
           Var1+Var2+Var3+Var4+Var5+Var6+Var7+Var8+
             NVVar1+NVVar2+NVVar3+NVVar4, data= m,
            hidden=2,threshold=0.01)
```

The model that has been formulated is then plotted using the **plot()** to visually view the relationships between the input variables and the output.



The entities on the left are various Inputs that we have considered in the model. The circles are the nodes that serve as the interaction centers. The Values denoted on the lines are the weightage assigned to the entities that are moving in from the left towards C_Claim which is the output variable. **'1'** represented in the plot are the constant coefficients that are present in the regression equation. Thus, the variables with respective weightages are multiplied by the weightages given to the constant and this is summed for all the variables taken into consideration. The net value is then given as output in the from of C_Claim.

The output we get from the above model is the probability of the observation falling into either Claim or No Claim classes. We can set a threshold value and predict and furthermore assign classes based on that.

We then calculate the Misclassification error using mean function as shown below. In the same attachment, we have the final error value as well and as it can be seen, the value of the error obtained has a very low value obtained through Neural Network modelling.

```
> misclassificationerror=mean(data$C_claim!=predict)
> misclassificationerror
[1] 0.007193452857
```

COMPARISON BETWEEN MODELS

The best model that we have selected from the ones mentioned earlier (namely, Random Foresting Technique, XG Boosting technique and Neural Network Technique) is **Neural Network Technique.**

The following are the reasons that back up our decision to select Neural Network Technique:

- The size of the Raw data that was given initially is considerably large. Techniques like Random Forest and XG Boosting are not efficient and become computationally expensive when using such large data sets. On the other hand, Neural Network is a technique that is extremely ideal and is best suited for large data sets.
- As the number of observations increase, more number of trees will be required to fit the data which can lead to over-fitting and thus leading to a in-accurate result on test data. Whereas, this is not a possible scenario in case of Neural Network Technique since we don't consider any such parameters that are to be trained.
- The given data is highly unbalanced i.e, the given data contains much more 0's than 1's. This forces the model to get used to the 0's more than 1's which might lead to a scenario where in the model misses out on predicted '1'. Such a scenario leads to in accurate prediction of result on the test data. We need to sample the data in order to fit a random forest, but this does not guarantee the best result on test data. Hence, we prefer Neural network which does not require data sampling over unbalanced data.
- Much of Neural Network's efficiency is due to the presence of back propagation in the algorithm. Back propagation is a weight updating strategy which makes Neural network more robust which is not in both Random forest and XG Boosting technique.
- In case of Neural Network each variable is assigned a weightage depending upon its importance. This provides an accurate quantitative output when opposed to the working algorithm in random forest and XG Boosting.

Thus, owing to the above mentioned reasons we select Neural Network as our most efficient model from the ones that are mentioned in previous pages.

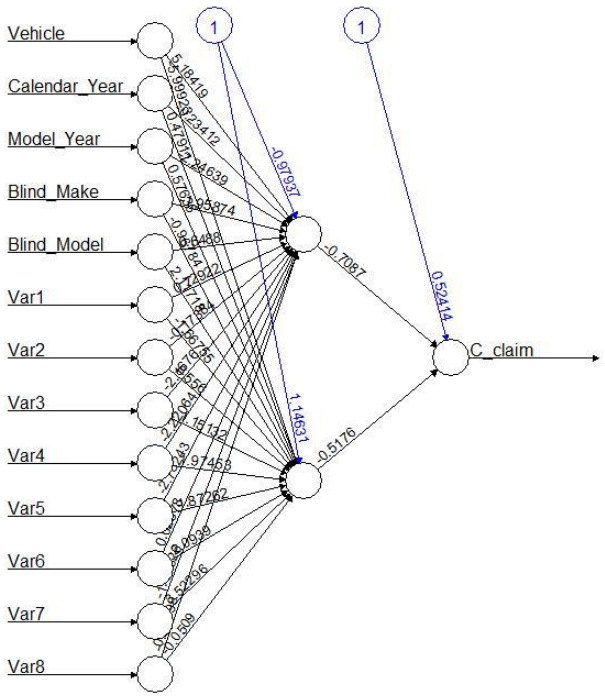
EVALUATING TEST RESULTS

The normalization of data for Neural Network model was done through **min-max** scaling and the following confusion matrix was obtained as the result for the given Test data.

```
> confusionMatrix(table(actual=testing$C_claim,pred=tpred1))
Confusion Matrix and Statistics

      pred
actual     0     1
     0 42783  6850
     1   257   110

               Accuracy : 0.85786
                 95% CI : (0.8547685, 0.8609099)
    No Information Rate : 0.8608
    P-Value [Acc > NIR] : 0.9713867

                  Kappa : 0.0163087
 Mcnemar's Test P-Value : < 0.00000000000000022

            Sensitivity : 0.9940288
            Specificity : 0.0158046
         Pos Pred Value : 0.8619870
         Neg Pred Value : 0.2997275
             Prevalence : 0.8608000
         Detection Rate : 0.8556600
   Detection Prevalence : 0.9926600
      Balanced Accuracy : 0.5049167

       'Positive' Class : 0
```

We can see that we get an accuracy of 85.8% and a sensitivity value of 99.4%.
Of the given 50,000 data points as test data the number of claims and no-claims that were predicted by our model is shown in the confusion matrix shown alongside.

The confidence interval value signifies that, 95% of the time our model results or prediction will have an accuracy value ranging between 85.4% to 86%.



The entities on the left are various Inputs that we have considered in the model. The circles are the nodes that serve as the interaction centers. The Values denoted on the lines are the weightage assigned to the entities that are moving in from the left towards C_Claim which is the output variable. **'1'** represented in the plot are the constant coefficients that are present in the regression equation. Thus, the variables with respective weightages are multiplied by the weightages given to the constant and this is summed for all the variables taken into consideration. The net value is then given as output in the form of C_Claim.

For example: If we consider the variable "**vehicle"** alone, and assume that it has a value of 1; then:

The value 1, gets multiplied by the weightage assigned to the variable that is, 5.18419 and the coefficient -0.97937 is added to it which gives the value of intermediate node. This net value is then multiplied by the weightage assigned to the intermediate node and furthermore the respective co-efficient is added from which we obtain the output. And based upon the threshold we divide them into **claim** or **no-claim**.

CHANGES MADE FROM THE INITIAL MODEL

We made few changes in the previous model in order to improve the prediction accuracy and sensitivity for the given test data. The changes that were made are listed below:

1. The process of normalizing the data in order to use them as arguments was done through min-max scaling instead of matrix scaling. In case of matrix scaling, each label will have its column created and this leads to difficulty in obtaining the confusion matrix in the end. We avoid this by introducing min-max scaling technique.

   Previous model:

```
m = model.matrix( ~C_claim+Household_ID+Vehicle+Calendar_Year+Model_Year+
    Blind_Make+Blind_Model+Blind_Submodel+Var1+Var2+Var3+Var4+
    Var5+Var6+Var7+Var8+NVVar1+NVVar2+NVVar3+NVVar4, data=data)
```

   Revised model:

```
ddata=data[,c(2,3,4,5,6,21,22,23,24,25,26,27,28,34)]
max = apply(ddata, 2 , max)
min = apply(ddata, 2 , min)
scaled = as.data.frame(scale(ddata, center = min, scale = max - min))
```

2. The threshold value to divide the output into claim and no-claim was changed from 0.4 in the previous model to 0.2 in the revised model; which leads to a better sensitivity and better accuracy of prediction on test data compared to the previous model.

   Previous model:

```
predict=ifelse(model1$net.result[[1]]>0.4,1,0) # treshold o.4
```

   Revised model:

```
tpred1=ifelse(tp>0.2 ,1,0)
```

3. The variable "**Household_Id**" was not included in the revised model which in turn leads to a better prediction accuracy.

CITATIONS

1. www.AnalyticsVidya.com
2. www.google.com
3. www.wikepedia.com