Predicting Auto Insurance Claims

Predictive Modelling (MA5790) Final Project

Kazeem Kareem and Alan Bouwman
Michigan Technological University
Instructor: Professor Quiying Sha

December 2022

Abstract

According to *Statista*, in 2019, there were over 12 million vehicles involved in crashes in the United States, with over half that volume being passenger cars. There are a number of factors that contribute to risk for a crash, and it is usually the case that auto insurance companies are keen on understanding individuals' risk of getting involved in a crash in order to help them determine an appropriate premium for each policyholder. In this work, we deploy predictive modelling techniques to analyse various driver features and built several models to predict whether–or not– an individual would get involved in an auto crash and file an insurance claim. Our best models turn out to be a *neural network* followed by a *flexible discriminant analysis* model. Our optimal model– neural network model– performs well on the testing set without any indication of overfitting, and with a reasonable computational time. It is also observed that the nonlinear models in general outperform the linear models, albeit the linear models are much less computationally complex. Moreso, it is discovered that some of the most important predictors in determining the risk of crash include whether or not drivers live in urban area, the age of the driver and the number of kids who drive, among others.

Contents

1 Introduction

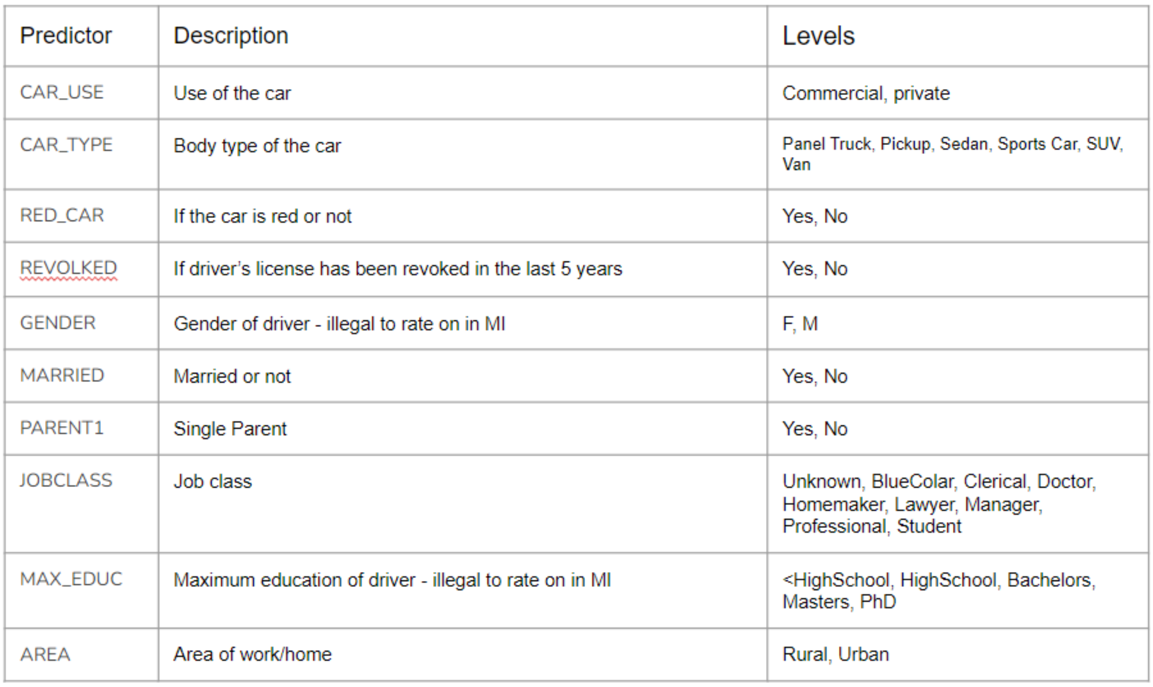
In the event of a motor vehicle accident, insurance may cover many related expenses. These expenses include injury to other drivers or passengers, property damage to other vehicles or homes, and damage to your own vehicle. Insurance plays in important roll in keeping keeping peoples lives stable on an individual level, and also keeping the economy stable. Due to the importance of insurance, companies are often required by law to keep sufficient funds for paying out customer claims. Insurance companies are also under pressure to not overcharge their customers, as the customer may choose to take out a policy through a different company that can offer a lower price. Pricing models must also be interpretable, as regulating bodies examine these models in great detail to make sure the prices are fair and non-discriminatory.

Insurance companies sell their customers (policy holders) a promise that they will be reimbursed in the event of a loss. Typically, when a product or service is sold, the seller sets the price to the sum of the profit they wish to make and the cost of the product or service to themselves. When an insurance policy is sold, the ultimate cost of the policy is not known at the time of the sale. Predicting this cost is one of the fundamental challenges of insurance and the task of the actuary. The price of an insurance policy is called the premium.

One important step in calculating the premium is determining what types of customers are likely to file claims for accidents. This will be the focus of this work.

2 Data set

Our dataset came from the compound Poisson linear models (cplm) package in R. The original dataset had 10,296 samples with 10 categorical predictors and 24 numerical predictors. The dataset includes two response variables, CLM_FLAG and CLM_AMNT. The dataset also included other columns with no predictive ability, such as the policy date, the policy number, and whether or not the sample was used in a certain paper. The table below presents a breakdown of the categorical predictors:



Predictor	Description	Levels
CAR_USE	Use of the car	Commercial, private
CAR_TYPE	Body type of the car	Panel Truck, Pickup, Sedan, Sports Car, SUV, Van
RED_CAR	If the car is red or not	Yes, No
REVOLKED	If driver's license has been revoked in the last 5 years	Yes, No
GENDER	Gender of driver - illegal to rate on in MI	F, M
MARRIED	Married or not	Yes, No
PARENT1	Single Parent	Yes, No
JOBCLASS	Job class	Unknown, BlueColar, Clerical, Doctor, Homemaker, Lawyer, Manager, Professional, Student
MAX_EDUC	Maximum education of driver - illegal to rate on in MI	<HighSchool, HighSchool, Bachelors, Masters, PhD
AREA	Area of work/home	Rural, Urban

The breakdown of the continuous variables is given in the table below:

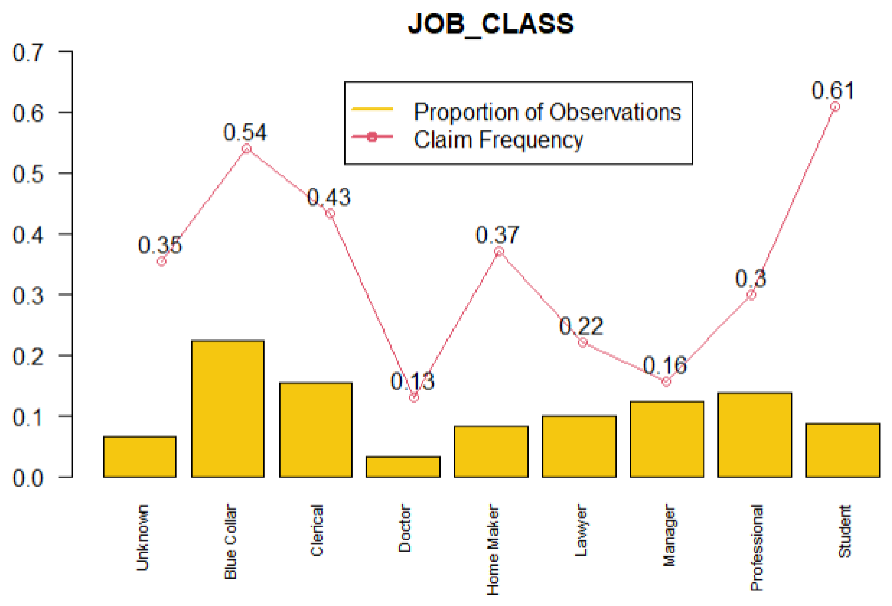
AGE	Age of the driver
HOMEKIDS	Number of children living in your home
YOJ	Number of years in your current job, includes NA's
INCOME	Annual income, includes NA's
HOME_VAL	Value of home, includes NA's
SAMEHOME	Number of years lived in the same home, includes NA's

When preprocessing, we noticed that the JOBCLASS predictor for the level DOCTOR will have near zero variance after creating dummy columns. If we were to simply throw this variable out, it would have been equivalent to putting doctor into the same jobclass as unknown (as the unknown level is dropped when we make dummy columns). However, intuition tells us that a doctor might be a particularly safe driver. For example,

Predictor	Description	Variable type
CLM_FREQ5	Number of times claim has been filed in last 5 years	integer
CLM_AMT5	Aggregate claim loss of policy (in thousands)	integer
KIDSDRIV	Number of kids who drive	integer
TRAVTIME	Commute time	integer
BLUEBOOK	(log) car value	integer
RETAINED	The number of years as a customer	integer
NPOLICY	Number of policies	integer
MVR_PTS	Number of motor vehicle record points	integer
AGE	the age of the driver	integer
HOMEKIDS	The number of children	integer
YOJ	years at current job	integer
INCOME	Annual income	integer
HOME_VAL	the value of the insured's home	integer
SAMEHOME	years in the current address	integer

Table 1: Numerical Predictors and descriptions

a doctor who has seen many patients getting into an accident after texting and driving would be more likely to never text and drive. Instead of removing this predictor completely, we should test to see if the doctor class actually does provide predictive power, and if it does, we should group the doctor level with the jobclass level that is most similar to doctor in terms of response. In the graph below, the yellow bars represent the proportion of each level of job class in the data. The red dots represent the proportion of claims for that job class. Thus, 61%, 16% of managers, and 13% of doctors in this dataset filed a claim. Since doctors are most similar to managers in terms of the response, we group those levels of the JOBCLASS predictor together.



3 Goal of Study

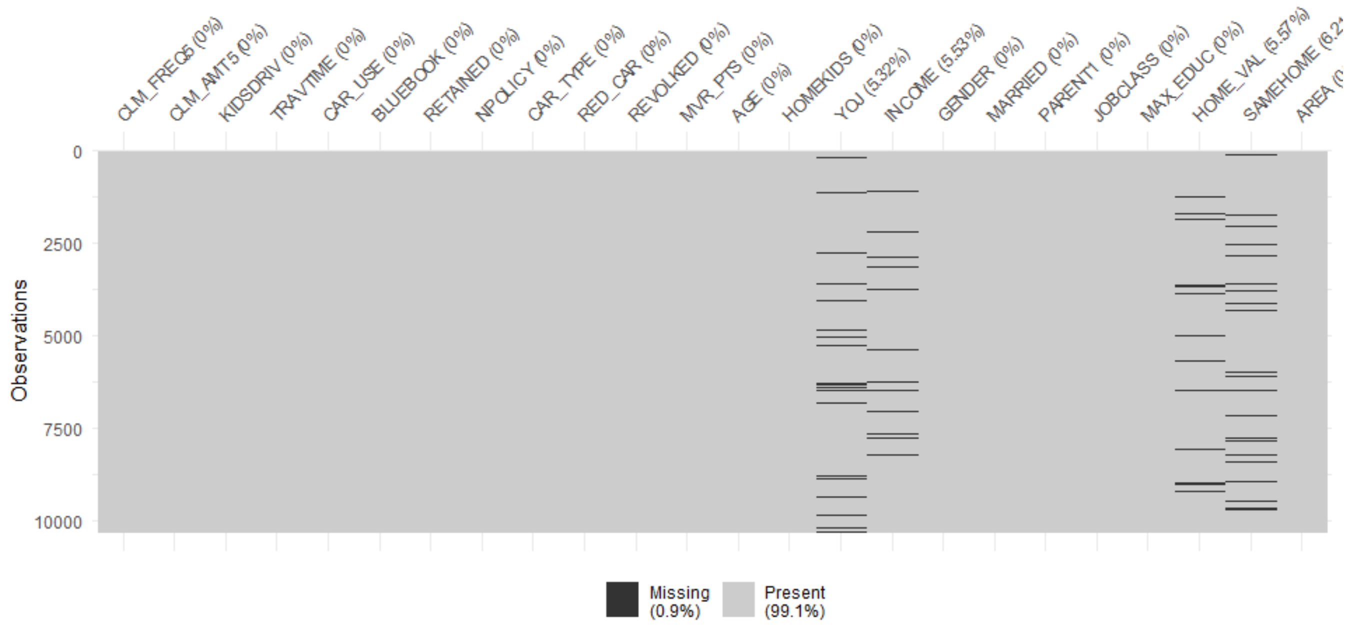
The goal of this study is to build several predictive models–linear and nonlinear– that would predict if a policyholder would file claim or not based on some of their characteristics, and determine which model perform best for our dataset. Also we want to identify what the most important variables are for this prediction.

4 Preprocessing

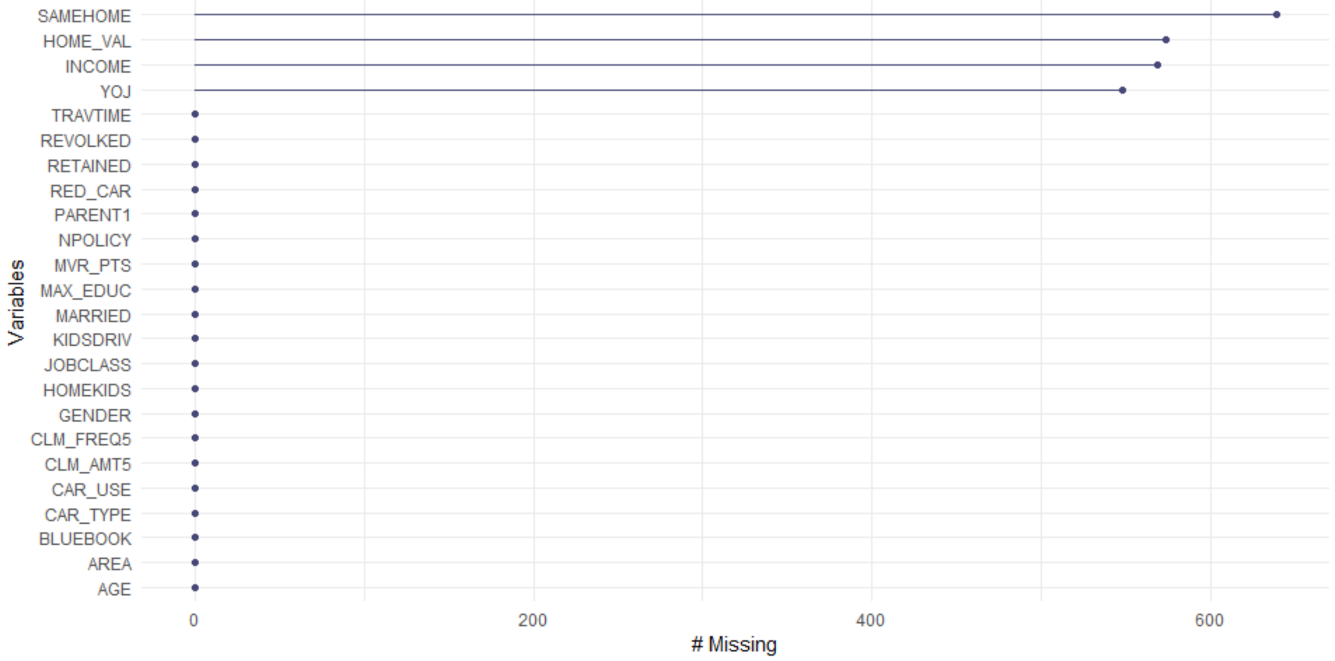
More often than not, datasets are in their "raw" forms– forms in which they cannot be used directly by models. It is therefore important to do some data cleaning in order to get the dataset in form that can be usable by the various models. While some models are robust to certain properties which need to be fixed through preprocessing, other models are adversely affected by these properties.

4.1 Missing Values

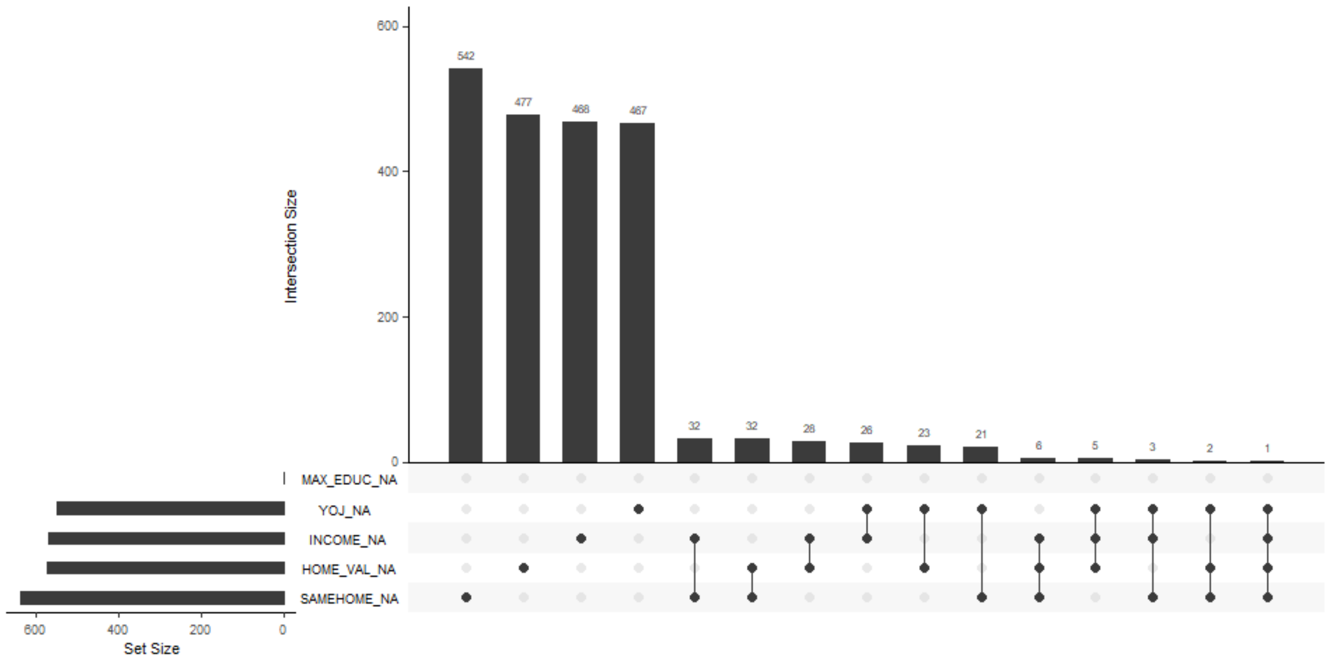
We start by analyzing the missing values in our dataset.



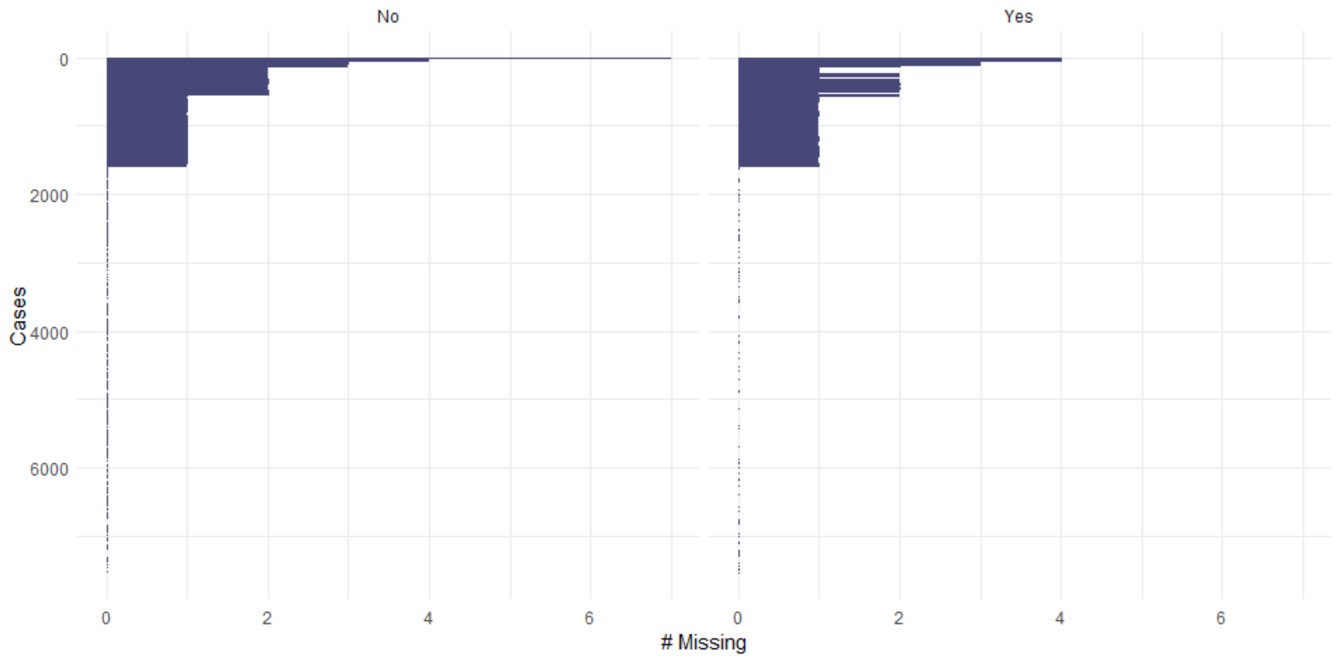
The above plot indicates that 0.9% of our dataset is missing. It also gives the positions of the missing values.



It would be noticed the missing values are restricted to just four predictors. We note that the percentage of missing values in the predictor with the highest missing values is ¡ 7%, which is not large, we do not contemplate deleting these predictors.



The above plot maps the coincidences of missingness of the missing values. It can be seen that there is minimal coincidences of missingness among the predictors, and so, we can hardly find a pattern in the missingness.



Lastly, the above plot shows the distributions of the missing values among the two classes in our response variable. It would be observed that the distribution of the missingness is not related to a particular class.

It can be deduced from the above plots that the missing values are simply random and do not bear any important information, and can be fixed by imputation.

4.1.1 Imputation

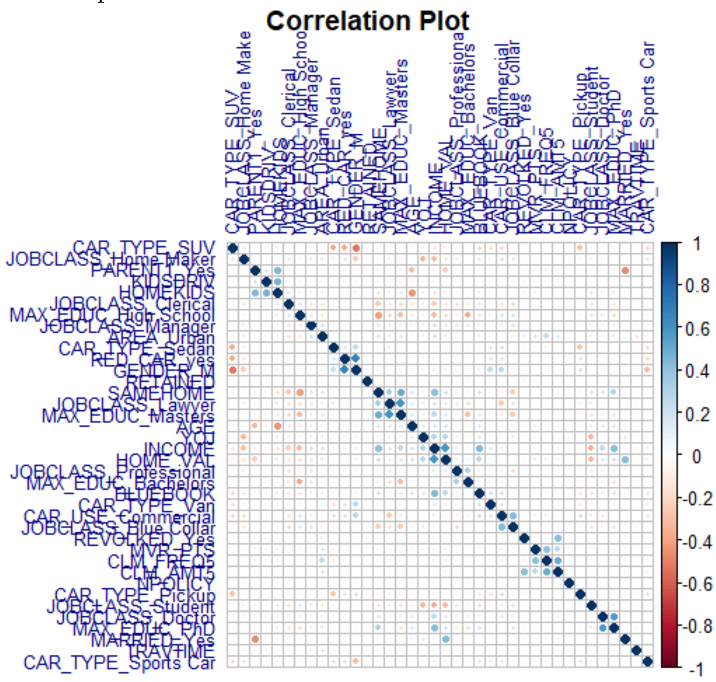
We use K- nearest neighbor imputation technique (with K=5) to impute the missing values.

4.2 Dummy Variables

As mentioned earlier on, our dataset includes 14 numerical variables and 10 categorical variables. Some models would only accept categorical variables with a threshold on the admissible number of classes present in a categorical predictor, in which case one might have to convert the categorical variables to dummy variables. Moreover, it also helps to make other preprocessing techniques, such as centering and scaling seamless, among other benefits. In this project, all our categorical variables are converted to dummy variables and the original categorical variables expunged. Furthermore, we observe that the vectors of dummy variables generated from each categorical variable always form a linearly dependent set, in the sense that each vector in this set is a linear combination of the other vectors. This can give rise to multicollinearity among the resulting dummy predictors. We therefore also delete one of the dummy variables for each categorical variable to avoid this *dummy variable trap*. The final number of predictors after implementing this process becomes 37, up from 24 predictors present initially.

4.3 Multicollinearity

We consider the following correlation plot.



It can be seen from the above plot that our predictors do not seem to have any significant high correlations among one another. Moreover, with a cutoff of 0.7, we are not able to screen any predictors possessing high correlation. Thus, the predictors look pretty much well-behaved at this point in terms of correlation.

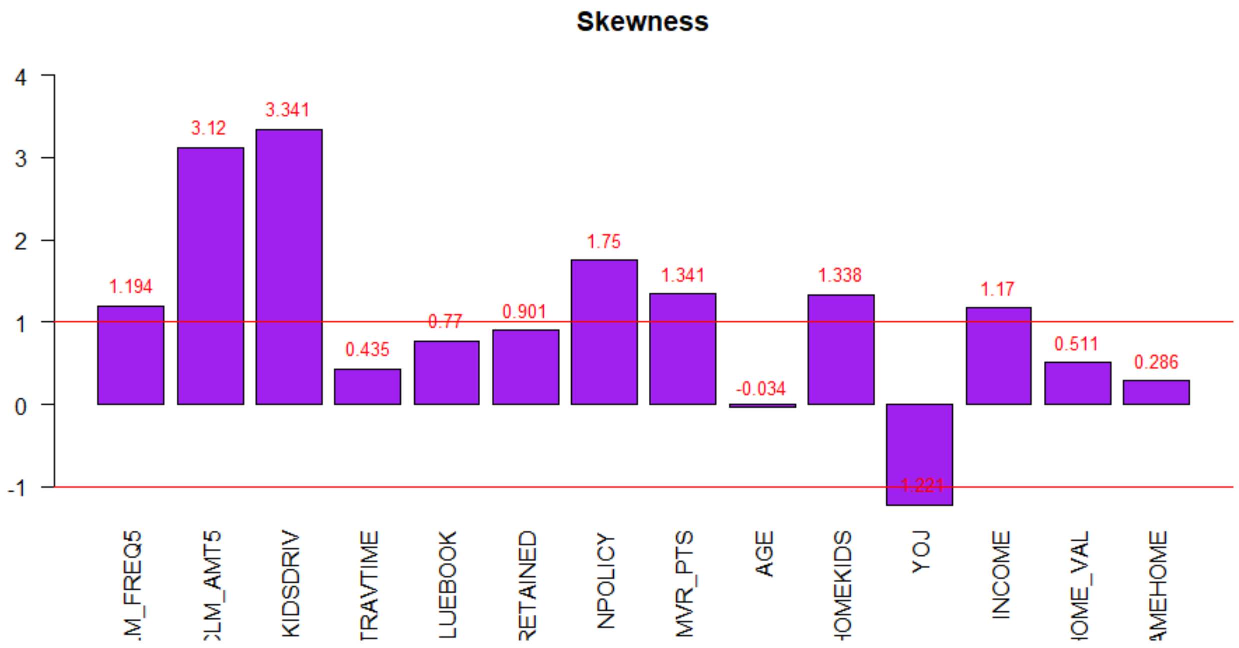
4.4 Near-Zero Predictors

We also check the dataset for near-zero predictors. There are no near-zero predictors that we need to delete.

4.5 Transformations

4.5.1 Skewness and Box-Cox Transformation

The following plot gives the skewness of various numerical variables.

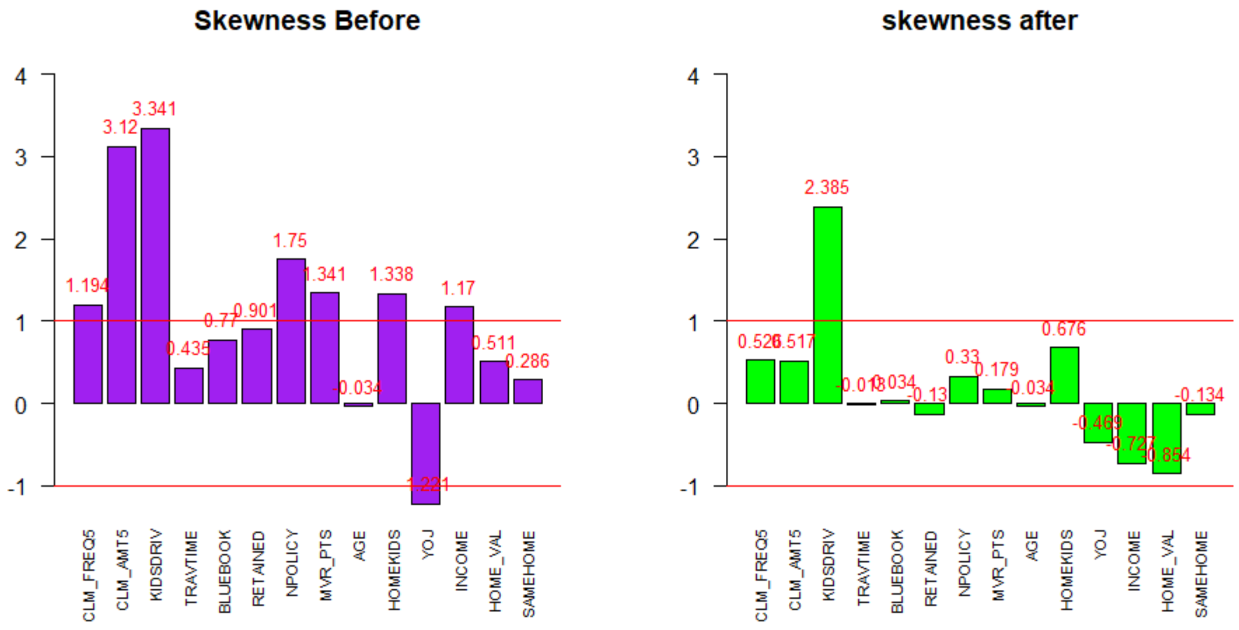


It can be observed that more than half the numerical predictors are highly skewed, with most of them skewed to the right. We attempt to correct this skewness by implementing Box-Cox transformation.

4.5.2 Box-Cox Transformation

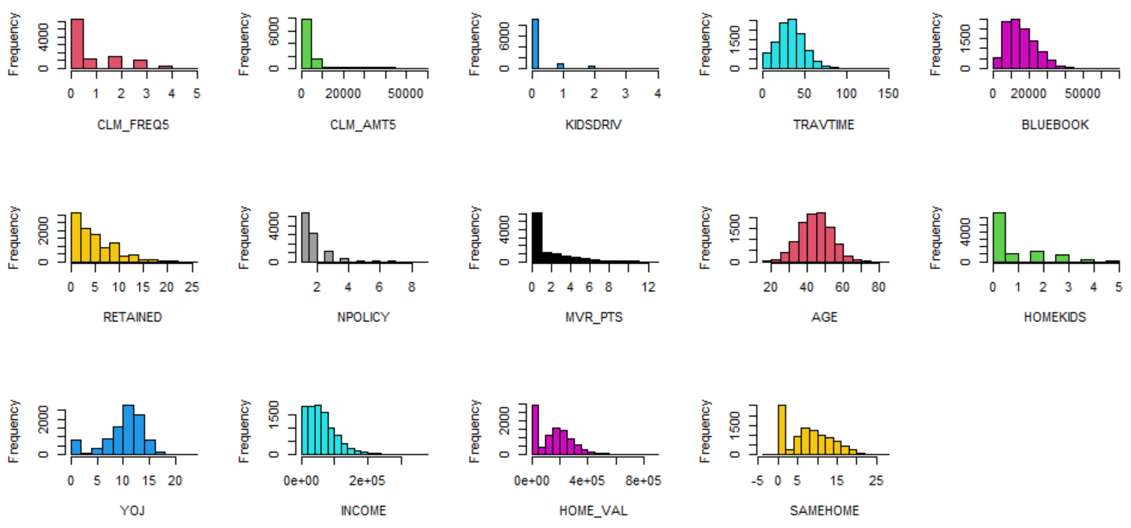
The box-Cox transformation for a variable $y = (y_1, y_2, \ldots, y_n)$ where $y_i > 0$ for each $i = 1, \ldots, n$, is given by

$$y_i^{(\lambda)} = \begin{cases} \frac{y_i^\lambda - 1}{\lambda}, & \text{if } \lambda \neq 0 \\ \log(y_i), & \text{if } \lambda = 0. \end{cases}$$

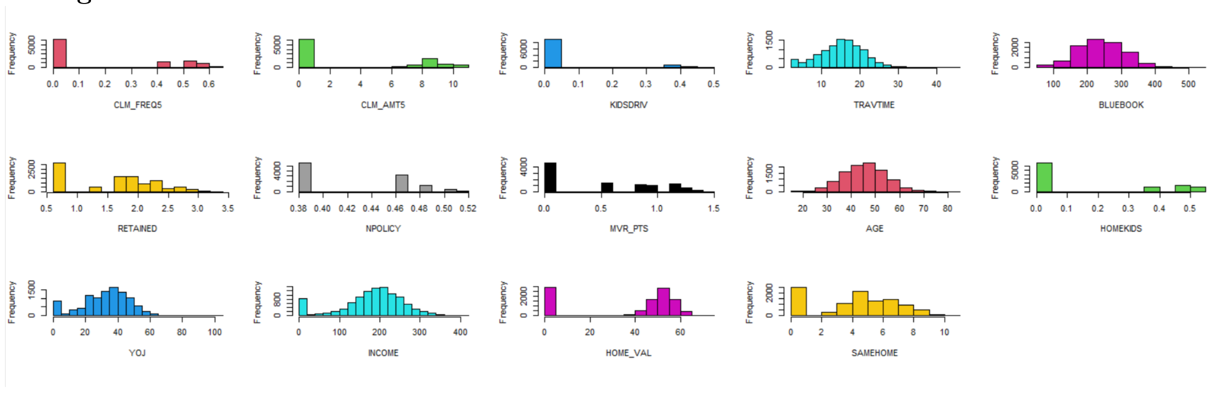


From the plot above, it is observed that the Box-Cox transformation significantly reduces the skewness of the numerical predictors with just one predictor being highly skewed.

Histogram Before Transformations



Histogram After Transformations

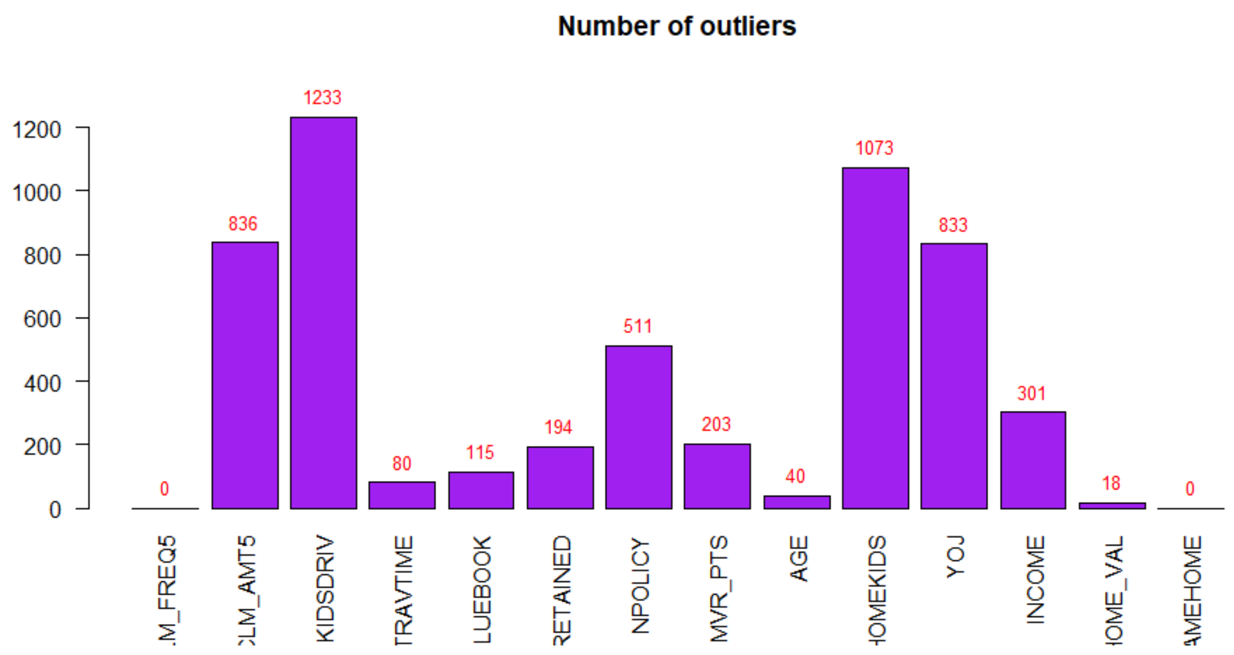


4.5.3 Centering and Scaling

Also, we center and scale all our predictors after carrying out Box-Cox transformation on the numerical counterparts.

4.5.4 Outliers and Spatial Sign Transformation

The following chart gives the number of outliers in each numerical variable.

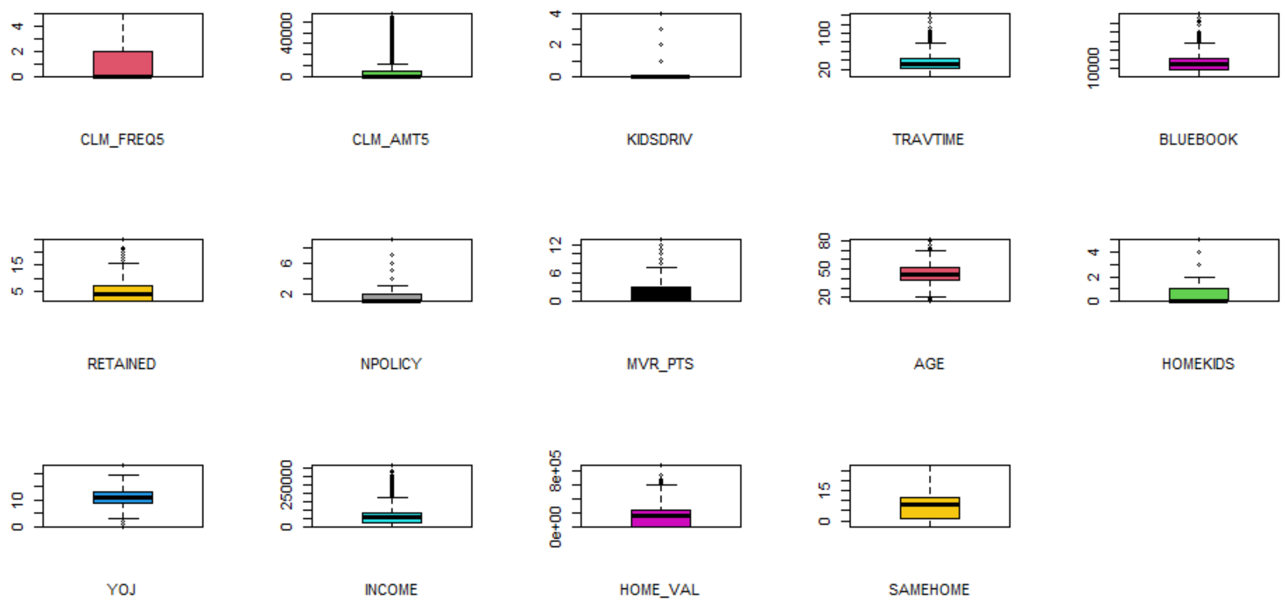


As seen above, there are a considerable number of outliers among the numerical variables.

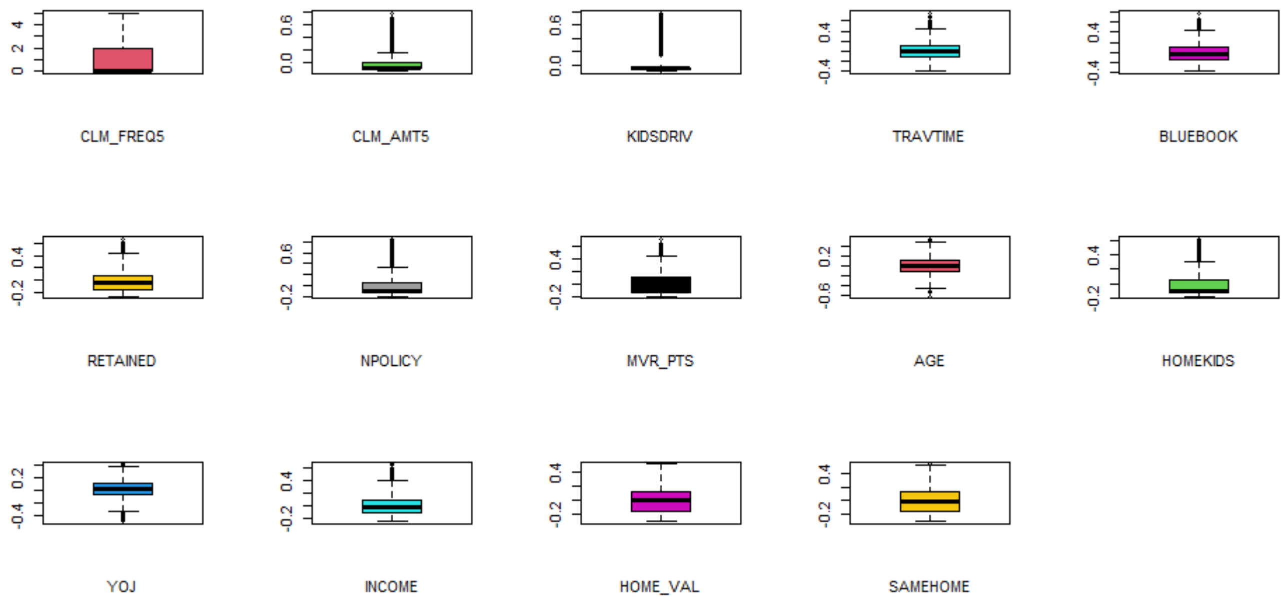
We address this problem by appliying Spatial Sign transformation on the dataset. This is done after all other transformations have been implemented. The outcome is depicted in the following barplots which shows that the number of outliers has been reduced in most of the numerical predictors. However, for three predictors, the numbers of outliers happened to be slightly increased. Overall, the totality of number of outliers is reduced.

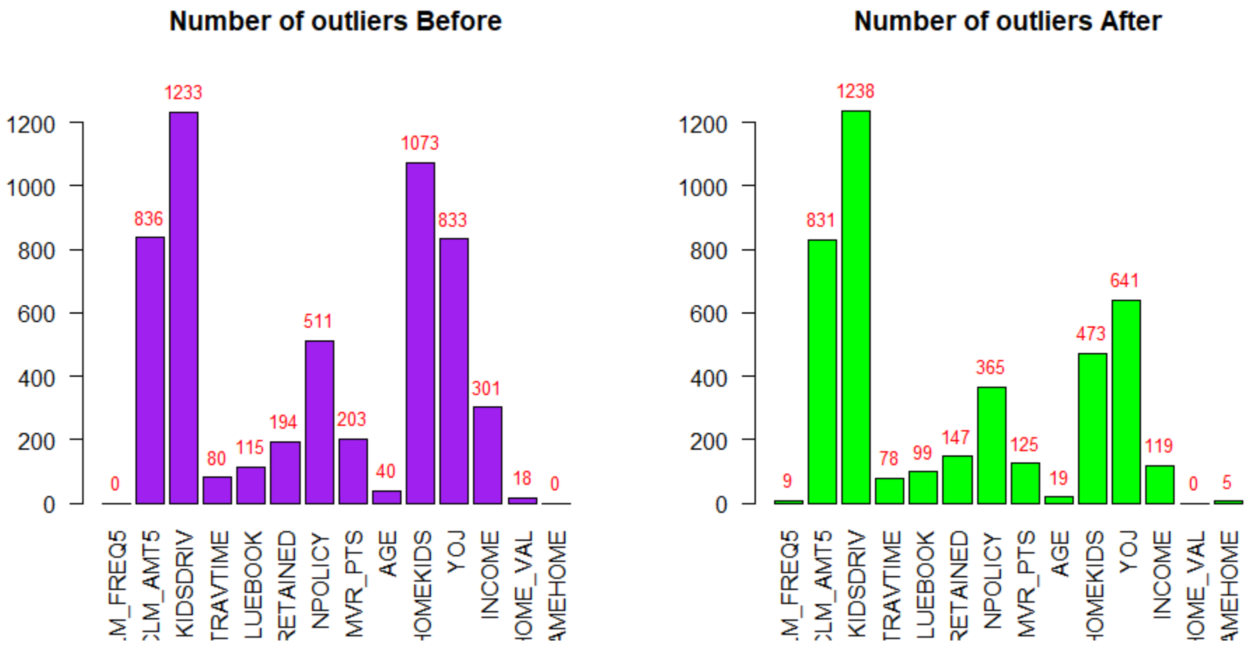
We can compare the boxplots before and and after the transformations below.

Boxplots Before Transformations



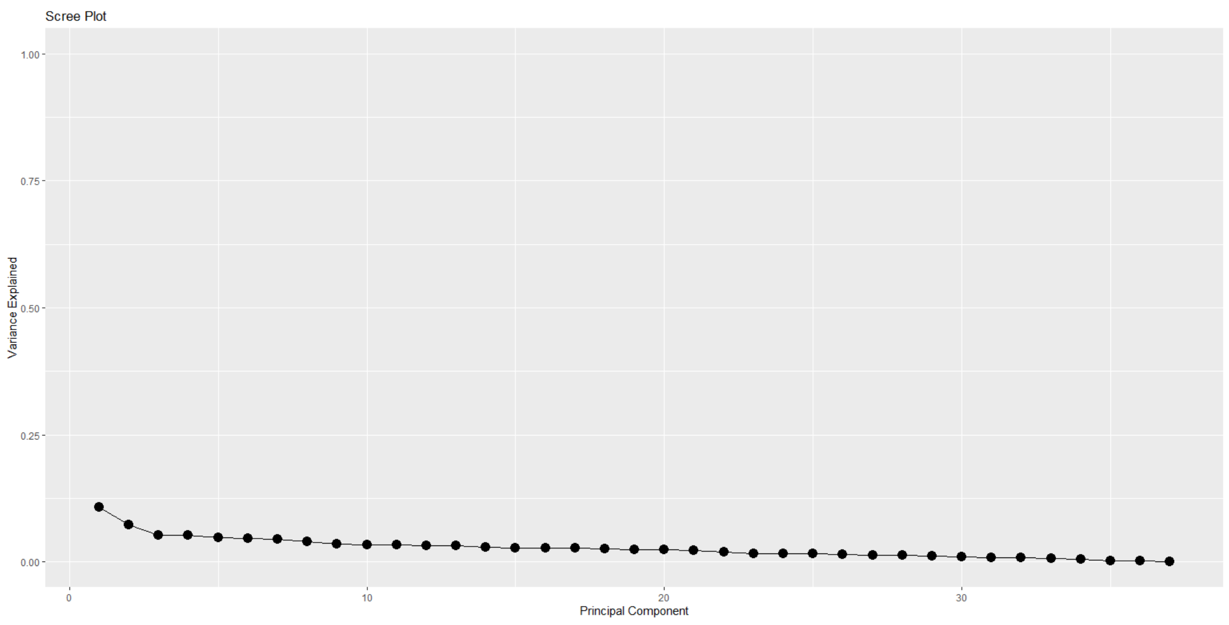
Boxplots Before Transformations

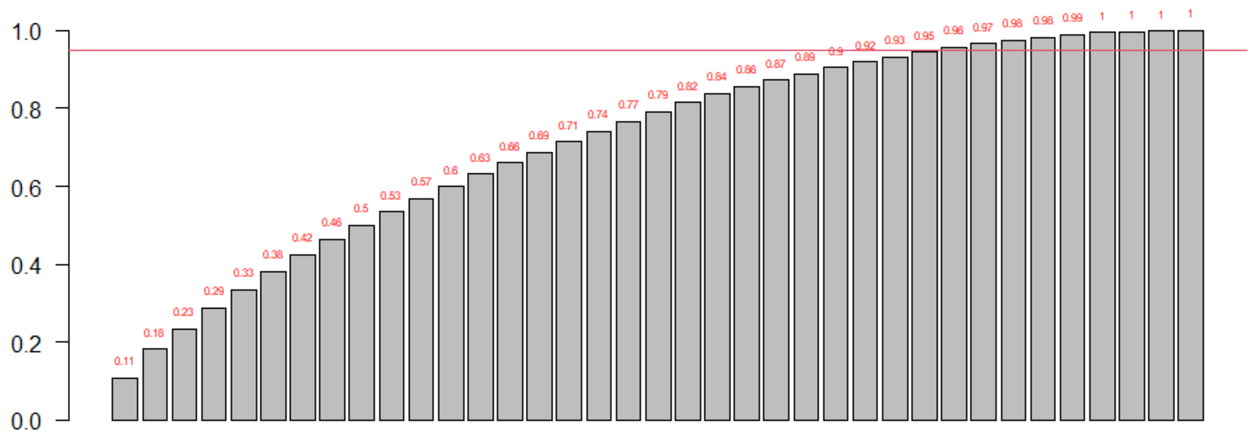




4.5.5 Principal Component Analysis

We also seek to see if it helps to reduce the dimension of the dataset. We implement principal component analysis and obtain the principal components. Below is a plot of the cumulative amount of variations that the principal components explain.

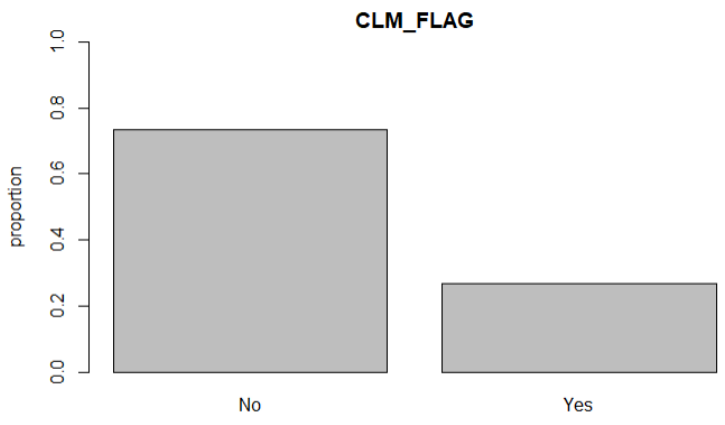




From the above scree plot and the plot that follows, it would be observed there is a very slow progression in the cumulative amount of variations explained by the PCs, which implies that the PCs do not substantially reduce the dimension of the dataset. Thus, we would not consider Principal Components Regression (PCR).

5 Data Splitting

At this point, our data is almost set for modelling. The response variable that we want to predict is the `CLM_FLAG` variable.



Our response variable `CLM_FLAG` is highly unbalanced (73% No; 27% Yes) as indicated by the barplot above. Due to this nature of the response variable, we use stratified random split to partition our dataset into training and test sets. We will use an 80/20 train/test split. For each of the models that would be considered, we do 10-fold Cross Validation on the training set.

6 Results

In this section, we present a summary of the model results on the training validation set along with the best tuning parameters. We also give will show the tuning parameter graphs for each model, and briefly discuss the training.

6.1 Summary of Results

In the table below, we show the Accuracy, Kappa score, sensitivity, specificity precision, F1, and AUC for each model. The models with the best Kappa score are the Neural Network (NNetwork) and Flexible Discriminant Analysis (FDA). Interestingly, the model with the worst performance is Naive Bayes. This is likely because the assumption of Independence among predictors does not hold well for many of our predictors. The most extreme example of this in in the case of CLM_AMNT5 and CLM_FREQ5. Also, JOBCLASS, MAX_EDUCATION, and INCOME are all likely related. Interestinly, Naive Bayes achieves an accuracy of only 0.71. Since 73% of the predictors response variables are CLM_FLAG=No, a model that always predicts No would have peformed with higher accuracy than the Naive Bayes Model.

Model	Accuracy	Kappa	Sensitivity	Specificity	Precision	F1	AUC
logistic	0.787	0.392	0.915	0.437	0.817	0.863	0.812
penalized	0.789	0.398	0.916	0.440	0.819	0.864	0.816
LDA	0.789	0.391	0.919	0.428	0.816	0.865	0.808
QDA	0.758	0.435	0.781	0.696	0.876	0.826	0.795
MDA	0.794	0.436	0.896	0.512	0.835	0.865	0.747
PLSDA	0.786	0.359	0.936	0.370	0.804	0.865	0.804
random glm	0.789	0.392	0.919	0.429	0.816	0.865	0.816
NaiveBayes	0.716	0.256	0.819	0.431	0.799	0.809	0.737
KNN	0.758	0.309	0.893	0.385	0.800	0.844	0.778
SVM	0.789	0.435	0.883	0.532	0.839	0.860	0.797
FDA	0.809	0.462	0.920	0.501	0.836	0.876	0.800
NNetwork	0.809	0.466	0.918	0.508	0.837	0.876	0.804

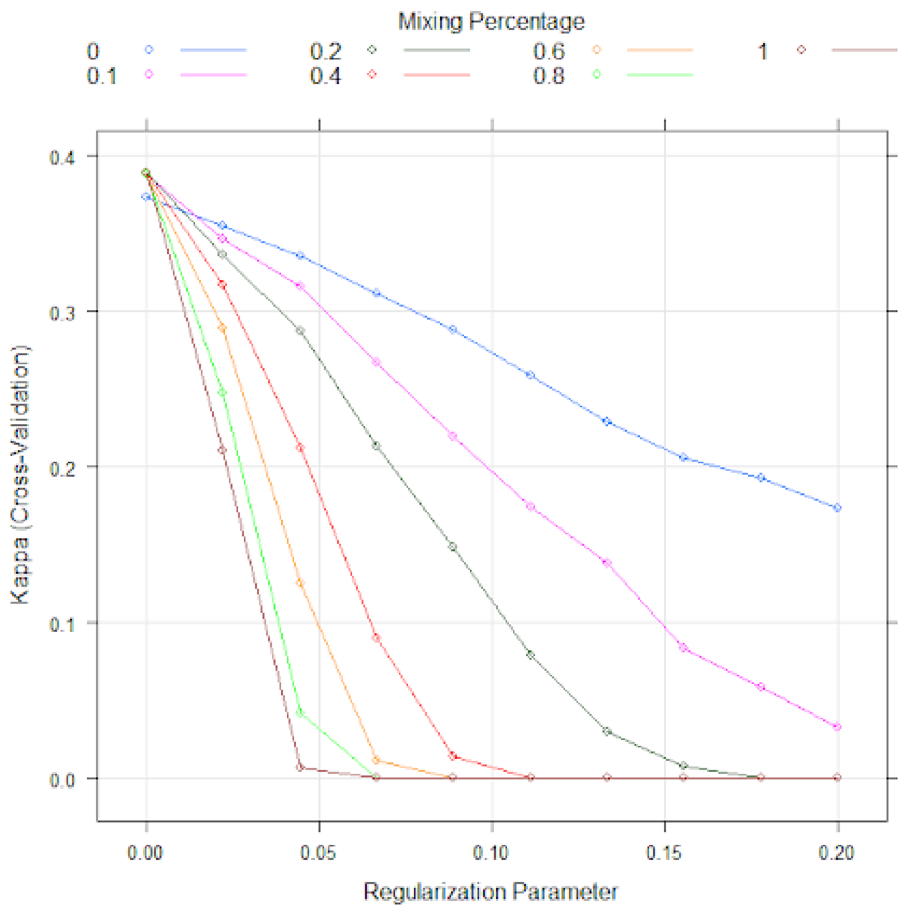
Table 2: Performance Profiles of various Classification models

6.1.1 Logistic Regression

Logitic regression had a kappa score of 0.392. This model has no tuning parameters.

6.1.2 Penalized Logistic Regression

Penalized Logistic Regression (PLR) perfromed slightly better than Logsitic Regression, wiht a kappa score of The best model for Penalized Logistic Regression, with a kappa score of 0.398. The best tuning parameter for PLR was with $\lambda = 0, \alpha = 0.2$. Since a $\lambda = 0$, no penalty was applied to the model. We noticed that as we increased λ and α the model would tend to predict "No" more often. With sufficiently high tuning parameters, the model would learn to always predict "No". When training, α ranged from 0 to 1 with a step size of 0.1, and λ ranged from 0 to 2 with 10 steps.



6.1.3 Linear Discriminant Analysis (LDA)

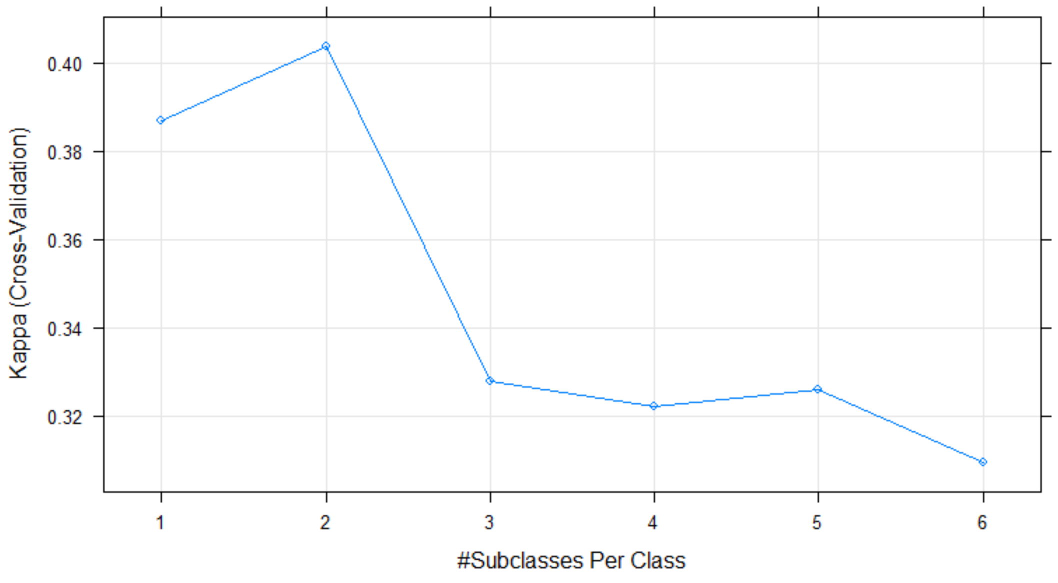
LDA has no tuning parametes. LDA achiaved a Kappa score of 0.391.

6.1.4 Quadratic Discriminant Analysis (QDA)

QDA has no tuning parameters. QDA model achieved a kappa score of 0.435.

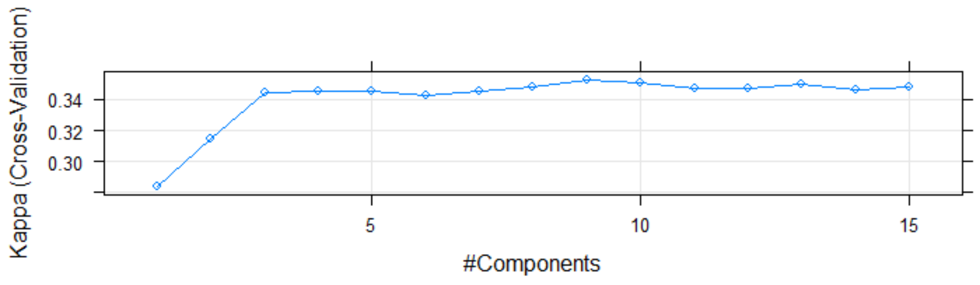
6.1.5 Mixture Discriminant Analysis (MDA)

The best MDA model achieved a kappa score of 0.436. The best tuning parameter was number of sublasses=2. The number of subclasses was allowed to range from 0 to 6 during tuning.



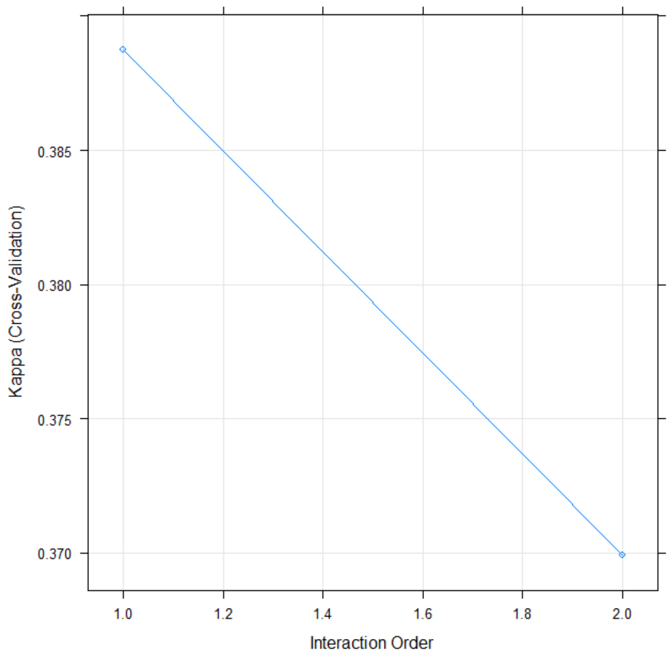
6.1.6 Partial Least Squares Discriminant Analysis (PLSDA)

The best PLSDA model achieved a kappa score of 0.359. The optimal model used $n = 9$ partial least square compoenets. In tuning, the number of components varied from 1 to 15.



6.1.7 Random GLM

The best Random GLM model achieved a kappa of 0.392, which was the same as logistic regression. The tuning parameter for Random GLM is degree of interaction, and was allowed to range from 1 to 2 during tuning. An optimal value of 1 was selected. The random GLM took an incredible amount of time to train. With little to no gain in performance, this model is not well suited for this problem but could perhaps perform better with more fine tuning.

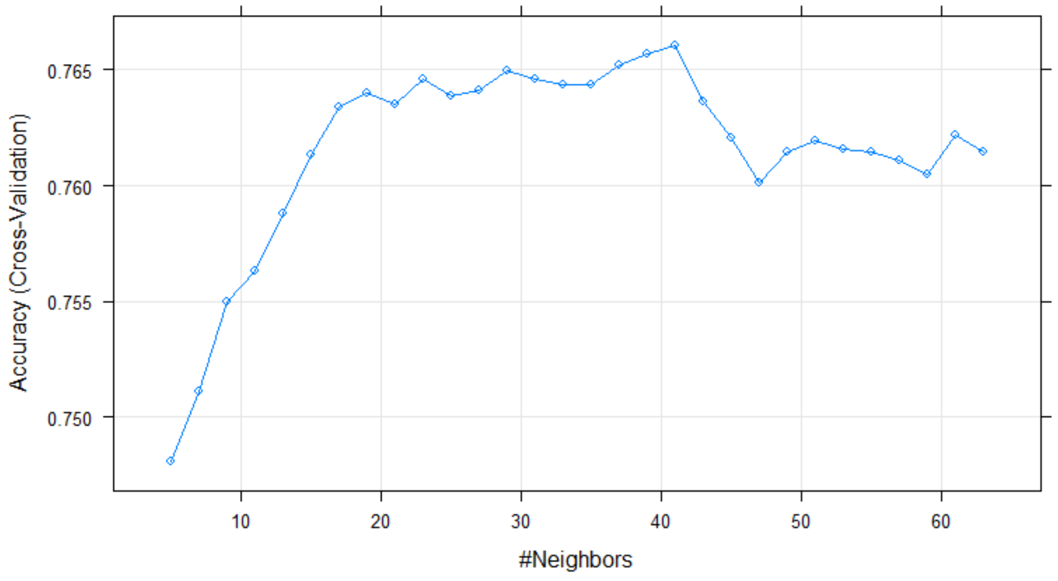


6.1.8 Naive Bayes

Naive Bayes has no tuning parameters, however, was ran with a Laplcian smoothing coefficient of 1. Naive Bayes had the worst overall Kappa score of any of the tuned models, with a Kappa of 0.256.

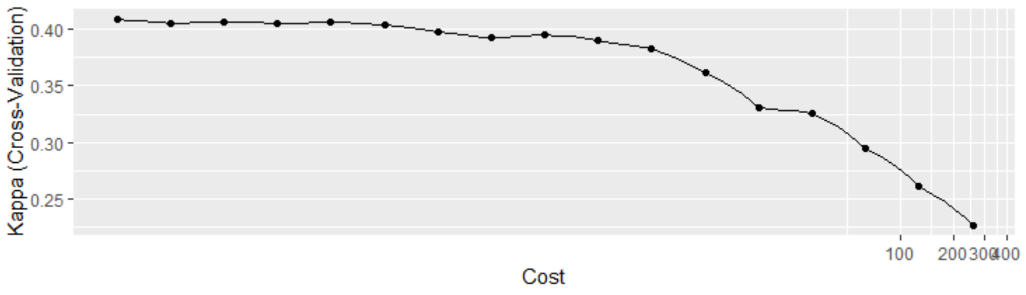
6.1.9 k-Nearest Neighbors (KNN)

The best KNN model achieved a kappa score of 0.309. The optimal number of neighbors was $k = 41$. During tuning, k was allowed to vary between 5 and 63 with a step size of 2.



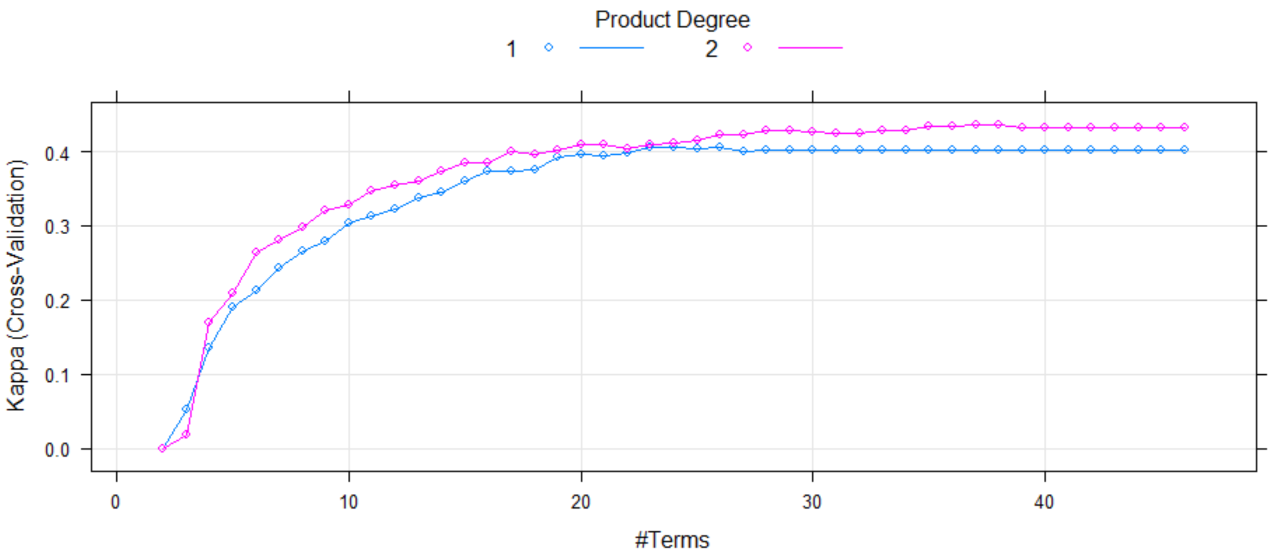
6.1.10 Support Vector Machine (SVM)

The best SVM model achieved a kappa score of 0.435. The optimal tuning parameter was Cost = 0.0039. During tuning, the parameter σ was held at a constant value of $\sigma = 0.00975$.



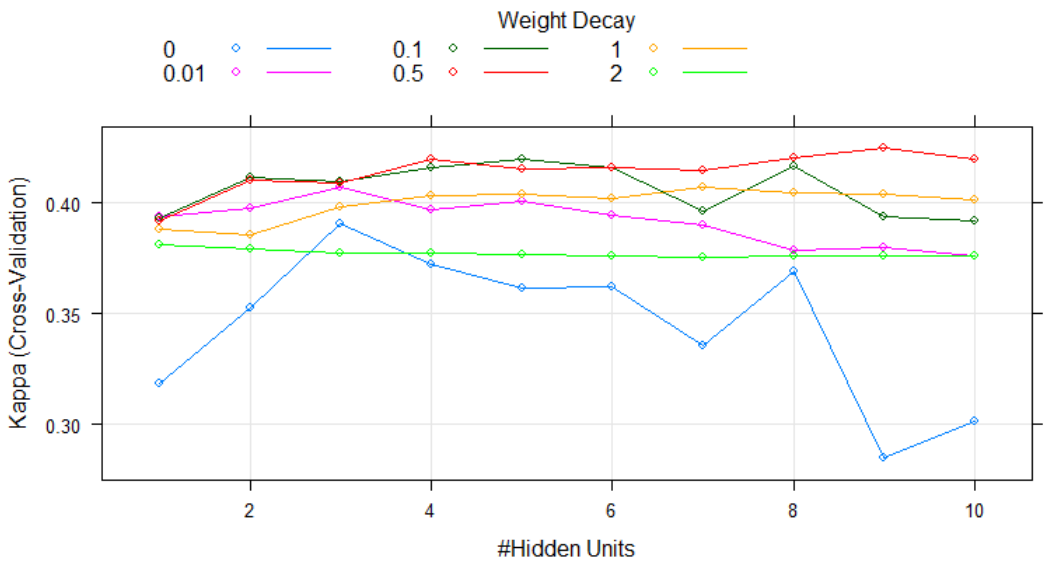
6.1.11 Flexible Discriminant Analysis (FDA)

The best FDA model achieved a kappa score of 0.462, making it the second best tuned model overall. This is only very slightly worse than the Neural Network. The interpretability of FDA makes it optimal for this problem, where premium pricing models are heavily examined by regulating bodies to ensure the prices are fair and non-discriminatory. The optimal tuning parameters were found to be degree of interaction = 2 and number of prunes = 37. During tuning, the number of interactions varied between 1 and 2 the the number of prunes varied from 2 to 38.



6.2 Neural Network (NNetwork)

The best Neural Network model achieved a kappa score of 0.466. The optimal tuning parameters were number of hidden units = 9 and decay = 0.5. During tuning, the number of units varied from 1 to 10 and the decay took values in $\{0, 0.1, 0.5, 1.0, 2.0\}$.



7 Best Models

In determining the best models, we primarily consider Kappa values due to the imbalanced nature of the response variable. Our two best models using this criterion are for this dataset are The Neural Network model and the flexible discriminant model. The Neural network and the FDA model run time were 19.3 minutes and 26.53 minutes respectively. For proper context, we used the parallel computing package, `doParallel`, in R on a Windows 10, Core i7, OctaCore machine to run these models and took the time measures.

We then predict with these two models on the test set.

Model	Accuracy	Kappa
FDA	0.793	0.414
NNetwork	0.793	0.416

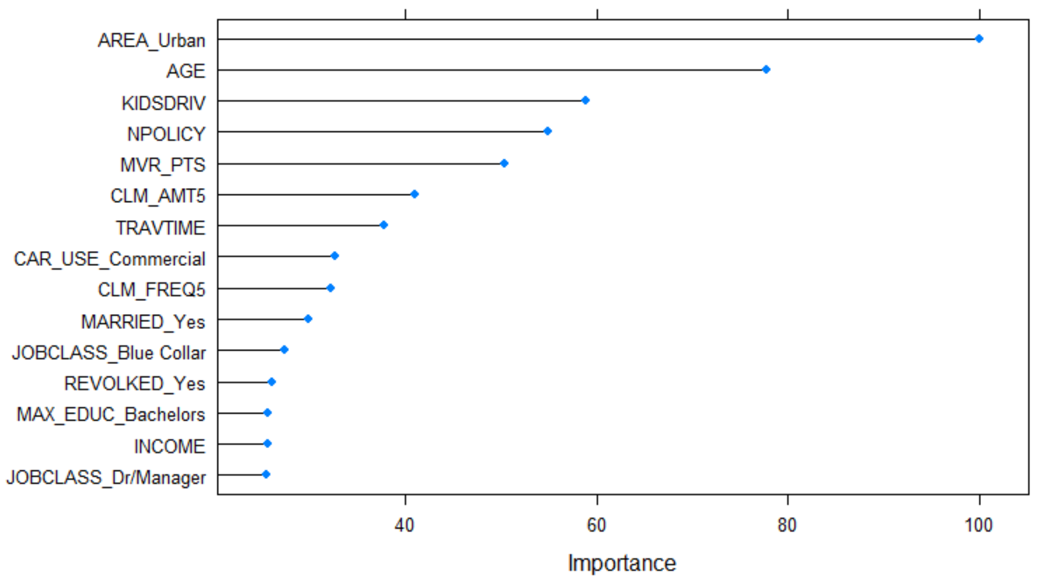
Table 3: Performances of the best Classification models on Test set

The confusion matrix of the Neural Network model is given below.

Prediction	Reference	
	No	Yes
No	1375	291
Yes	136	257

Table 4: Performances of the best Classification models on Test set

8 Important Variables



It can be inferred from the plot above that the most important variables are: Area: whether Urban area or not; the age of the driver; number of child passengers and so on, it the order presented above. It is also noticed that being a doctor or a manager is also among the top 15 important variables in predicting auto crash. This corroborates our insight that being a doctor or not, though sparse when standing alone, or not potentially bears some useful signal when combined with some other job.

9 Conclusion

We conclude by noting the following

1. We recommend that the best model is the neural network model. It had a Kappa value of 0.466 and an accuracy of 0.809 on the training set. On the test set, it had Kappa and accuracy values of 0.416 and 0.793 respectively.
2. The most important variables were Area (whether Urban or Not), Age and Number of kids who Drive in a household.
3. The Discriminant Analysis Models–except the PLSDA– performed very well.
4. Logistic model was also competitive and extremely fast to implement compared to the other models; it runs within a minute under our machine set up, with a Kappa value of 0.392. If speed were our topmost priority, this model would have been chosen.
5. The worst performing models are Naive Bayes and KNN model.
6. We recommend that more data be collected with more features and sample point so this model can be improved.

10 References

1. `https://www.bankrate.com/insurance/car/auto-insurance-statistics/#what-types-of-auto-insurance-claims-are-filed. Accessed on 12/14/2022 at 20:48.`
2. Max Kuhn , Kjell Johnson, Applied Predictive Modelling, Springer 2013.

11 Appendix: Code

```
############# Predictive Modelling Project #############
library(naniar)
library(dplyr)
library(moments)
library(corrplot)
library(AppliedPredictiveModeling)
library(caret)
library(e1071)
library(cplm)
library(tweedie)
library(moments)
library(fastDummies)
library(statmod)
library(HDtweedie)
library(Hmisc)
library(pROC)
#####  AutoCaim data
data(AutoClaim)
# View(AutoClaim)

z <- AutoClaim
str(z)

z1 <- z[, -c(1, 2, 5, 16, dim(z)[2] )] #remove the columns 1,2,5, 16 and last.
# We need to mention why we removed these columns

response_amt <- z[,5] # response variable
response_flag <- z[,16]
x <-z1
barplot(table(response_flag), main = "response Variable: CLM_FLAG" )
########################### Group JobClass Doctor With Manager ########################################

n_observations = aggregate((z$CLM_FLAG), list(z$JOBCLASS), FUN=length)
n_claims = aggregate((z$CLM_FLAG=="Yes"), list(z$JOBCLASS), FUN=sum)
claim_freq = n_claims[,2]/n_observations[,2]
dif_from_dr = abs(claim_freq[4] - claim_freq)
min(dif_from_dr[-4]) # the 7th agrument (Manager) matches doctor the cloest, Lawyer (6th argument)
is the second closest
level_names = c("Unknown", "Blue Collar", "Clerical", "Doctor", "Home Maker",
"Lawyer", "Manager", "Professional", "Student")

df = data.frame(level_names)
#df$n_obervations = n_observations
df$n_observations = n_observations[,2]
df$n_claims = n_claims[,2]
df$claim_freq = claim_freq
df$dif_from_dr = dif_from_dr
df

#This barplot suggests:
#  a) Job class==Doctor could have a lot of predictive power
# ‘b) Doctors are very similar to Managers in terms of the reponse variable
barplot(prop.table(table(AutoClaim$JOBCLASS)), ylim=c(0,0.7), main=’JOB_CLASS’, las=2,
cex.names = 0.7, col=7)
ticks=seq(1, 10.2, length=9)
points(ticks, t(claim_freq), type=’l’, col=2)
points(ticks, t(claim_freq), col=2)
text(ticks, t(claim_freq)+0.03, round(t(claim_freq), 2))
legend(x = 3.5, y = 0.65, legend = c("Proportion of Observations", "Claim Frequency"),
       col = c(7, 2), lwd = 2, lty=c(1,1), pch=c(NA,1))

# re group
x <-z1
```

```
levels(x$JOBCLASS) <- c("Unknown", "Blue Collar", "Clerical", "Dr/Manager",
"Home Maker", "Lawyer", "Dr/Manager", "Professional", "Student")

# barplot again after the grouping is done
x$CLM_FLAG = response_flag
n_observations = aggregate((x$CLM_FLAG), list(x$JOBCLASS), FUN=length) # length-- or count
n_claims = aggregate((x$CLM_FLAG=="Yes"), list(x$JOBCLASS), FUN=sum)
claim_freq = n_claims[,2]/n_observations[,2]
claim_freq
barplot(prop.table(table(x$JOBCLASS)), ylim=c(0,0.7), main='JOB_CLASS', las=2, cex.names = 0.7, col=7)
ticks=seq(.8, 9, length=8)
points(ticks, claim_freq, type='l', col=2)
points(ticks, t(claim_freq), col=2)
text(ticks, t(claim_freq)+0.03, round(t(claim_freq), 2))
legend(x = 3.5, y = 0.65, legend = c("Proportion of Observations", "Claim Frequency"),
       col = c(7, 2), lwd = 2, lty=c(1,1), pch=c(NA,1))

x <- x[,-dim(x)[2] ]
######################################### Group JobClass Doctor With Manager End ######################


num_cols <- unlist(lapply(x, is.numeric)) #determine thenumeric predictors
fact_cols <- unlist(lapply(x, is.factor))  # or lapply(x,  is.character):  #determine the
non-numeric predictors
numCols = x[,num_cols] #sepearate out the numerical columns
factCols = x[,fact_cols] # same for factor  columns




#histograms of numerical variables
par(mfrow=c(3,5))
for (i in 1:dim(numCols)[2]) {
  hist(numCols[,i], main = " ", xlab = colnames( numCols)[i], col =i+1 )
}



#bar plots of categorical variables
par(mfrow=c(3,4))
for (i in 1:dim(factCols)[2]) {
  barplot(table( factCols[,i]), main = " ", xlab = colnames( factCols)[i], col =i+1 )
}

#boxplots plots of numerical variables
par(mfrow=c(3,5))
for (i in 1:dim(numCols)[2]) {
  boxplot(numCols[,i], main = " ", xlab = colnames( numCols)[i], col =i+1 )
}



# The following function counts the  number of outliers in a data.
outliercount <- function(x)  {
  length(boxplot.stats(x)$out)
}

# We then compute the number of outliers in each predictor, in a table.
outliertable <- apply(numCols, 2, outliercount ) #gives a table of the frequency
                                                 # of outliers in a predictor
par(mfrow= c(1,1))
tt <- barplot(outliertable, las =2, ylim = c(0,1350),
              main = "Number of outliers", col = "purple")  # prints a barplot of
                                                 # number of outliers in each column.
```

```
text(x = tt, y = outliertable, label = outliertable,
                     pos = 3, cex = 0.8, col = "red")


##  Missing values analysis

library(naniar)
#which predictors have higher missing values? See plot below


gg_miss_var(x, show_pct = FALSE) # naniar package required


vis_miss(x) # naniar package required


gg_miss_upset(x) # to see the combination of interconnectedness of
                  # missing variables among predictors

x.temp <- cbind(x, response_flag)
gg_miss_case(x.temp,  x.temp$response_flag ) # to visualize how missing values relate to class.

sum(is.na(x$SAMEHOME))/length(x$SAMEHOME) # proportion of missing values in a column.

                                          # The column with highest missing values
                                              # less than 7% missing values.
                                              # So no need to remove the predictors


# We use knn imputation with k=5.

# During model building, we vary k to find which gives us the optimal model.
# KNN Imputation with k=5.
imputation <- preProcess(x,method="knnImpute") ##need {caret} package
# Apply the transformations:
imputed <- predict(imputation, x)

# undo center scale occasioned by imputation
#means = rowMeans(x, )
for(i in 1:dim(imputed)[2]){
  if (is.numeric(imputed[,i])){
    std = sd(x[,i], na.rm=TRUE)
    mean = mean(x[,i], na.rm=TRUE)
    imputed[,i] <- imputed[,i]*std+mean
  }
}

# fix the one value that is negative
imputed$SAMEHOME = abs(imputed$SAMEHOME)

dim(x)
dim(imputed)
 # 'imputed' is our updated data

xxx <- imputed


#Skewness  Before
par(mfrow=c(1,1))
skew <- round(apply(xxx[, num_cols], 2, skewness), 3)  #apply after imputation
ss <- barplot(skew, las =2, ylim = c(-1.2,4.0), main = "Skewness", col = "purple")
```

```
# Create bar plot with labels
abline(h=1, col = "red")  # cutoff line
abline(h=-1, col ="red")  # cutoff line
text(x = ss, y = skew, label = skew, pos = 3, cex = 0.8, col = "red")

# Creating dummy columns

xxx.dummies <- dummy_cols(xxx, colnames(factCols),
                          remove_first_dummy = TRUE,  # leaves us with a
                                                      # perfect correlation plot
                         remove_selected_columns = TRUE)

dim(xxx.dummies)
dim(xxx)

###########################
#    We use xxx.dummies  ########

# Personal note: manual way create dummy variables for CAR_USE
#private  <- ifelse(factCols$CAR_USE == "Private", 1, 0)
#commercial <- ifelse(factCols$CAR_USE == "Commercial", 1, 0)


new.numeric <- xxx.dummies[,1:14]  # Numeric col's after imputation.


min(new.numeric+0.01)

# Boxcox
boxcox.num <- preProcess( new.numeric+0.01, method = "BoxCox")  ## need {caret} package
# the output
boxcoxed <-  predict(boxcox.num,new.numeric+1)



#before and after  plots
#############################

#### before boxcox

#histograms of numerical variables
#par(mfrow=c(3,5))
#for (i in 1:dim(numCols)[2]) {
#  hist(numCols[,i], main = " ", xlab = colnames( numCols)[i], col =i+1 )
#}
#### After boxcox

#par(mfrow=c(3,5))
#for (i in 1:dim(boxcoxed)[2]) {
#  hist(boxcoxed[,i], main = " ", xlab = colnames( boxcoxed)[i], col =i+1 )
#}
#####################

boxcoxed.xx <- cbind(boxcoxed, xxx.dummies[,-(1:14)]) # Merge the boxcoxed numeric
                                                  # vars with the dummied vars


# Correlation plot
par(mfrow=c(1,1))
corrplot(cor(boxcoxed.xx),
      #  type="lower",
         order="hclust",
         tl.col="navy",
      #  addCoef.col="black",
         number.cex=1.0,
         tl.cex=0.8,
         title="Correlation Plot",
         # hide correlation on principal diagnal
      #  diag=FALSE,
         mar=c(0,0,1,0))

# Next we want to remove high correlations
highCorr <- findCorrelation(cor(boxcoxed.xx), cutoff = .80)
length(highCorr)
highCorr
#filteredxx <- xx[, -c(highCorr)]  # tosses out CLM_FREQ5
#str(filteredxx)
#str(boxcoxed.xx)

# near zero variances
#nzv<- nearZeroVar(filteredxx, freqCut = 95/5, uniqueCut = 10, saveMetrics = T); nzv
# The result affects only JOBCLASS_Doctor
#nzv.remove <- xx[-xxx.dummies$JOBCLASS_Doctor]    #this drops Doctors. But we retain doctors
                                            # What is the significance of income
nzv<- nearZeroVar(boxcoxed.xx, freqCut = 95/5, uniqueCut = 10, saveMetrics = T); nzv

#Transformations

#pca <- prcomp(filteredxx, scale = TRUE)
pca <- prcomp(boxcoxed.xx, scale = TRUE)

# compute total variance
variance = pca$sdev^2 / sum(pca$sdev^2)

variance

pcs = 1:length(boxcoxed.xx)
df = data.frame(pcs)
df$pecent_variance = variance
df$cumulative_percent = cumsum(variance)
df

qplot(c(1:length(boxcoxed.xx)), variance) +
  geom_line() +
  geom_point(size=4)+
  xlab("Principal Component") +
  ylab("Variance Explained") +
  ggtitle("Scree Plot") +
  ylim(0, 1)


pcbar <- barplot(df$cumulative_percent, las =2,  ylim = c(0,1.1),
               main = "")
text(x = pcbar, y = df$cumulative_percent, label = round(df$cumulative_percent,2),
     pos = 3, cex = 0.5, col = "red", las =2)
abline(h=.95, col =2)
##############
#############
#trans <- preProcess(filteredxx, method = c( "center", "scale", "pca", "spatialSign"))
trans <- preProcess(boxcoxed.xx, method = c( "center", "scale", "pca", "spatialSign"))
trans
```

```
# the output
transformed <-  predict(trans, boxcoxed.xx) # our new variable after all
                                            # necessary transformations.

# Next we check the various plots and compare the before's  with the after's.
transnumeric <- transformed[, 1:14]
#transfactor <- transformed[, fact_cols]

#Next, we do spatial sign transformation on the whole dataset.

spatial.trans <- preProcess(transformed[,1:dim(boxcoxed.xx)[2]], method = "spatialSign")
spatial.out <- predict(spatial.trans, transformed[, 1:dim(boxcoxed.xx)[2] ])
#We also do  spatial sign transformation of the pca's

SpatialPca <- preProcess(transformed[,(dim(boxcoxed.xx)[2]+1) : (dim( transformed))[2]   ],
                         method = "spatialSign")
spatialpca.out <- predict(SpatialPca,
                          transformed[,(dim(boxcoxed.xx)[2]+1) : (dim( transformed))[2]   ] )
##############################################################
# Before and after  plots
##############################################################

#### Before boxcox, centering, scaling, spatial for numeric
par(mfrow=c(3,5))
for (i in 1:dim(numCols)[2]) {
  hist(numCols[,i], main = " ", xlab = colnames( numCols)[i], col =i+1 )
}


#### After boxcox
par(mfrow=c(3,5))
for (i in 1:dim(numCols)[2]) {
  hist(transnumeric[,i], main = " ", xlab = colnames(transnumeric)[i], col =i+1 )
}

##############################
### Boxplots before transformations
par(mfrow=c(3,5))
for (i in 1:dim(numCols)[2]) {
  boxplot(numCols[,i], main = " ", xlab = colnames( numCols)[i], col =i+1 )
}

#### After boxcox
par(mfrow=c(3,5))
for (i in 1:dim(transnumeric)[2]) {
  boxplot(transnumeric[,i], main = " ", xlab = colnames(transnumeric)[i], col =i+1 )
}



##############################
#Skewness  Before
par(mfrow=c(1,2))
skew <- round(apply(xxx[, num_cols], 2, skewness), 3)  #apply after imputation
ss <- barplot(skew, las =2, ylim = c(-1.2,4.0),  cex.names = 0.7, main = "Skewness Before",
col = "purple")
# Create bar plot with labels
abline(h=1, col = "red")  # cutoff line
abline(h=-1, col ="red")  # cutoff line
text(x = ss, y = skew, label = skew, pos = 3, cex = 0.8, col = "red")
```

```
# After
skew2 <- round(apply(boxcoxed, 2, skewness), 3)  #apply after imputation
ss2 = barplot(skew2, las =2, cex.names = 0.7, main='skewness after', ylim=c(-1.2,4), col ="green")
# Create bar plot with labels
abline(h=1, col = "red")  # cutoff line
abline(h=-1, col ="red")  # cutoff line
text(x = ss, y = skew2, label = skew2, pos = 3, cex = 0.8, col = "red")

#############################

# Outlier counts before and after
par(mfrow= c(1,2))
# We then compute the number of outliers in each predictor, in a table.
outliertable <- apply(numCols, 2, outliercount ) #gives a table of the frequency
                     # of outliers in a predictor
tt <- barplot(outliertable, las =2, ylim = c(0,1350),
main = "Number of outliers Before", col = "purple")  # prints a table of number
of outliers in each column.
text(x = tt, y = outliertable,

label = outliertable, pos = 3, cex = 0.8, col = "red")

outliertable2<- apply(transnumeric, 2, outliercount)
#gives a table of the frequency of outliers in a predictor
tt2 <- barplot(outliertable2, las =2,  ylim = c(0,1350),
               main = "Number of outliers After", col = "green")  # prints a table of number
               of outliers in each column.
text(x = tt2, y = outliertable2,
label = outliertable2, pos = 3, cex = 0.8, col = "red")
##############

#####################
############################# Data Spending ########################################

# Our final cleaned data is ready, below

predictors <- spatial.out
predictors.pca <- spatialpca.out

# We now want to split out data into train-test. Stratified random
#  sampling applies here.


# response_flag is our response variable.
set.seed(75)
trainingRows <- createDataPartition(response_flag, p = .80, list= FALSE)  # requires caret
head(trainingRows)
nrow(trainingRows)
# Subset the data into objects for training using
# integer sub-setting

data <- xxx.dummies + 0.1
trainPredictors <- data[trainingRows, ]
trainresponse <- response_flag[trainingRows]

# Do the same for the test set using negative integers.
testPredictors <- data[-trainingRows, ]
testresponse <- response_flag[-trainingRows]

########################################################
###########################     M  O  D  E  L  S   ##################################################
```

```
#########################################################
ctrl <- trainControl(method = "cv", number = 10,
                     #summaryFunction = twoClassSummary,
                     classProbs = TRUE,
                     savePredictions = TRUE)

#### MODELS

############        1.   Support Vector Machine        #####################################
library(kernlab)


sigmaRangeReduced <- sigest(as.matrix(trainPredictors))
## sigest estimates the range of values for the sigma parameter
## which would return good results when used with a Support Vector Machine
## ksvm). The estimation is based upon the 0.1 and 0.9 quantile
## of ||x -x'||^2. Basically any value in between those two bounds
## will produce good results.
svmRGrid <- expand.grid(.sigma = sigmaRangeReduced[1],
                               .C = 2^(seq(-8, 8)))
library(doParallel)
cl <- makePSOCKcluster(7)
registerDoParallel(cl)
start <- Sys.time()
svmRTune  <- train(x = trainPredictors,
                   y = trainresponse,
                   method = "svmRadial",
                   metric = "Kappa",
                   preProc = c("center", "scale"),    # , "BoxCox", "spatialSign"),
                   tuneGrid = svmRGrid,
                   fit = FALSE,
                   trControl = ctrl)

svmRTune
plot(svmRTune)
library(ggplot2)
ggplot(svmRTune)+coord_trans(x='log2')
end <-  Sys.time()
duration.svm <- (as.numeric(end) - as.numeric(start)) / 60 # duration in minutes
duration.svm
predictedsvm <- predict(svmRTune, trainPredictors)
stopCluster(cl)

# Confusion matrix
c.svm <-  confusionMatrix(data =predictedsvm,
                reference = trainresponse)
roc.svm <- roc(response = svmRTune$pred$obs,
               predictor = svmRTune$pred$Yes,
                 #levels = rev(levels(svmRTune$pred$obs))
                            )
# plot(roc.svm, legacy.axes = TRUE)
auc.svm <- auc(roc.svm)
auc.svm
SVM <- c(c.svm$overall[1:2], c.svm$byClass[c(1:2,5:7) ], AUC= auc.svm) #  extracting some
SVM
##########      2.     Random forest model          #####################################
set.seed(278)
library(randomForest)
library(doParallel)
cl <- makePSOCKcluster(7)
registerDoParallel(cl)
```

```
start <- Sys.time()
rf_random <- train(x = trainPredictors,
                   y = trainresponse,
                   method = 'rf',
                   metric = 'Kappa',
                   preProc = c("center", "scale" , "BoxCox", "spatialSign"),
                   tuneLength  = 15,
                   trControl = ctrl)
print(rf_random)
plot(rf_random)
end <-  Sys.time()
duration.rf <- (as.numeric(end) - as.numeric(start)) / 60 # duration in minutes
stopCluster(cl)
duration.rf
predictedrf <- predict(rf_random, trainPredictors)

c.rf <-  confusionMatrix(data =predictedrf,
                          reference = trainresponse)
roc.rf <- roc(response = rf_random$pred$obs,
               predictor = rf_random$pred$Yes)
# plot(roc.svm, legacy.axes = TRUE)
auc.rf <- auc(roc.rf)
auc.rf
random.forest <- c(c.rf$overall[1:2], c.rf$byClass[c(1:2,5:7)], AUC = auc.rf)
random.forest

############      3.      Neural network model        #####################################
cl <- makePSOCKcluster(7)
registerDoParallel(cl)
nnetGrid <- expand.grid(.size = 1:10, .decay = c(0, 0.01,.1,0.5, 1, 2))
maxSize <- max(nnetGrid$.size)
numWts <- (maxSize * (  dim(trainPredictors)[2]   + 1)
+ (maxSize+1)*2) ## dim(trainPredictors)[2] is the number of predictors
set.seed(75)
start <-  Sys.time()
nnetFit <- train(x = trainPredictors,
                 y = trainresponse,
                 method = "nnet",
                 metric = "Kappa",
                 preProc = c("center", "scale",  "BoxCox", "spatialSign"),
                 tuneGrid = nnetGrid,
                 trace = FALSE,
                 maxit = 2000,
                 MaxNWts = numWts,
                 trControl = ctrl)
nnetFit
plot(nnetFit)
end <-  Sys.time()
duration.nn <- (as.numeric(end) - as.numeric(start)) / 60 # duration in minutes
duration.nn
stopCluster(cl)
#predict on training
predictednn <- predict(nnetFit, trainPredictors)
c.nn <-  confusionMatrix(data =predictednn,
                         reference = trainresponse)
roc.nn <- roc(response = nnetFit$pred$obs,
               predictor = nnetFit$pred$Yes)
# plot(roc.svm, legacy.axes = TRUE)
auc.nn <- auc(roc.nn)
auc.nn
```

```
NNetwork <- c(c.nn$overall[1:2], c.nn$byClass[c(1:2,5:7)], AUC = auc.nn)
NNetwork

############ 4a. Averaged Neural network model        #####################################
library(doParallel)
cl <- makePSOCKcluster(7)
registerDoParallel(cl)
#nnetGrid <- expand.grid(size = 1:10, decay = c(0, .1, 1, 2) , bag = TRUE)
nnetGrid <- expand.grid(.decay = c(0.001, .01, .1),
              .size = seq(1, 27, by = 2),
              .bag = FALSE)
#nnetGrid <- expand.grid(.size = 1:10, .decay = c(0, .1, 1, 2))
maxSize <- max(nnetGrid$.size)
numWts <- (maxSize * (  dim(trainPredictors)[2]   + 1) + (maxSize+1)*2) ## dim(trainPredictors)[2] is t
set.seed(75)
start <-  Sys.time()
avnnet <- train(x = trainPredictors,
                 y = trainresponse,
                 method = "avNNet",
                 metric = "Kappa",
                 preProc = c("center", "scale",  "BoxCox", "spatialSign"),
                 linout = TRUE,
                 trace = FALSE,
                 maxit = 1000,
                 trControl = ctrl)
avnnet


plot(avnnet)
end <-  Sys.time()
duration.avnn <- (as.numeric(end) - as.numeric(start)) / 60 # duration in minutes
duration.avnn
stopCluster(cl)
predictedavnn <- predict(avnnet, trainPredictors)

c.avnn <-  confusionMatrix(data =predictedavnn,
                         reference = trainresponse)
averagedNN<- c(c.avnn$overall[1:2], c.avnn$byClass[c(1:2,5:7)])
averagedNN
#######


############ 5.   K-Nearest Neighbor model       #####################################
library(doParallel)
cl <- makePSOCKcluster(7)
registerDoParallel(cl)
set.seed(75)
start <-  Sys.time()
knntune <- train(x = trainPredictors,
                 y = trainresponse,
                 method = "knn",
                 metric = "Kappa",
                 preProc = c("center", "scale"  ,  "BoxCox", "spatialSign"),
                 tuneLength = 30,
                 trControl = ctrl)
knntune
plot(knntune)
end <-  Sys.time()
duration.knn <- (as.numeric(end) - as.numeric(start)) / 60 # duration in minutes
duration.knn
stopCluster(cl)
```

```
# predict on train
predictedknn <- predict(knntune, trainPredictors)
c.knn <- confusionMatrix(data =predictedknn,
                         reference = trainresponse)
roc.knn <- roc(response = knntune$pred$obs,
              predictor = knntune$pred$Yes)
auc.knn <- auc(roc.knn)
auc.knn
KNN <- c(c.knn$overall[1:2], c.knn$byClass[c(1:2,5:7)], AUC = auc.knn)
KNN



############################### 6.  L D A  Model      ######################################
library(doParallel)
cl <- makePSOCKcluster(7)
registerDoParallel(cl)
set.seed(75)
start <-  Sys.time()
lda  <- train(x = trainPredictors,
               y = trainresponse,
               method =  "lda",
               metric = "Kappa",
              preProc = c("center", "scale" ),
                 trControl = ctrl)
lda
end <-  Sys.time()
duration.lda <- (as.numeric(end) - as.numeric(start)) / 60 # duration in minutes
duration.lda
stopCluster(cl)
# predict on train
predictedlda <- predict(lda, trainPredictors)
c.lda <- confusionMatrix(data =predictedlda,
                         reference = trainresponse)
roc.lda <- roc(response = lda$pred$obs,
               predictor = lda$pred$Yes)
auc.lda <- auc(roc.lda)
auc.lda
LDA <- c(c.lda$overall[1:2], c.lda$byClass[c(1:2,5:7)], AUC = auc.lda)
LDA



###############################   7.  Q D A  Model      ######################################
library(doParallel)
cl <- makePSOCKcluster(7)
registerDoParallel(cl)
set.seed(75)
start <-  Sys.time()
qda  <- train(x = trainPredictors,
              y = trainresponse,
              method =  "qda",
              metric = "Kappa",
              preProc = c("center", "scale" ),
              trControl = ctrl)
qda
end <-  Sys.time()
duration.qda <- (as.numeric(end) - as.numeric(start)) / 60 # duration in minutes
duration.qda
stopCluster(cl)
```

```
# predict on train
predictedqda <- predict(qda, trainPredictors)
c.qda <- confusionMatrix(data =predictedqda,
                         reference = trainresponse)
roc.qda <- roc(response = qda$pred$obs,
               predictor = qda$pred$Yes)
auc.qda <- auc(roc.qda)
auc.qda
QDA <- c(c.qda$overall[1:2], c.qda$byClass[c(1:2,5:7)], AUC = auc.qda)
QDA


############################## 8.  M D A  Model      ######################################
library(doParallel)
cl <- makePSOCKcluster(7)
registerDoParallel(cl)
set.seed(75)
start <-  Sys.time()
mda  <- train(x = trainPredictors,
              y = trainresponse,
              method =  "mda",
              metric = "Kappa",
              preProc = c("center", "scale" ), # ,  "BoxCox", "spatialSign"),
              tuneGrid = expand.grid(.subclasses = 1:6),
              trControl = ctrl)
mda
plot(mda)
end <-  Sys.time()
duration.mda <- (as.numeric(end) - as.numeric(start)) / 60 # duration in minutes
duration.mda
stopCluster(cl)
# predict on train
predictedmda <- predict(mda, trainPredictors)
c.mda <- confusionMatrix(data =predictedmda,
                         reference = trainresponse)
roc.mda <- roc(response = mda$pred$obs,
               predictor = mda$pred$Yes)
auc.mda <- auc(roc.mda)
auc.mda
MDA <- c(c.mda$overall[1:2], c.mda$byClass[c(1:2,5:7)], AUC = auc.mda)
MDA

############################## 9.  F D A  Model      ######################################
library(pamr)
library(doParallel)
cl <- makePSOCKcluster(7)
registerDoParallel(cl)
set.seed(75)
marsGrid <- expand.grid(.degree = 1:2, .nprune = 2:46)
start <-  Sys.time()
fda  <- train(x = trainPredictors,
              y = trainresponse,
              method =  "fda",
              metric = "Kappa",
              preProc = c("center", "scale" ), # ,  "BoxCox", "spatialSign"),
              tuneGrid = marsGrid,
              trControl = ctrl)
fda
plot(fda)
end <-  Sys.time()
duration.fda <- (as.numeric(end) - as.numeric(start)) / 60 # duration in minutes
duration.fda
stopCluster(cl)
# predict on train
predictedfda <- predict(fda, trainPredictors)
c.fda <- confusionMatrix(data =predictedfda,
                         reference = trainresponse)
roc.fda <- roc(response = fda$pred$obs,
               predictor = fda$pred$Yes)
auc.fda <- auc(roc.fda)
auc.fda
FDA <- c(c.fda$overall[1:2], c.fda$byClass[c(1:2,5:7)], AUC = auc.fda)
FDA

############################### 10.  P L S D A  Model      ######################################
library(klaR)
library(doParallel)
cl <- makePSOCKcluster(7)
registerDoParallel(cl)
set.seed(75)
start <-  Sys.time()
### PLSDA Model

set.seed(647)
plsda <- train(x = trainPredictors,
               y = trainresponse,
               method = "pls",
               tuneGrid = expand.grid(.ncomp = 1:15),
               preProc = c( "center","scale" ,"BoxCox", "spatialSign"),
               metric = "Kappa",
               trControl = ctrl)
plsda
plot(plsda)
# predict on train
predictedplsda <- predict(plsda, trainPredictors)
c.plsda <- confusionMatrix(data =predictedplsda,
                         reference = trainresponse)
roc.plsda <- roc(response = plsda$pred$obs,
               predictor = plsda$pred$Yes)
auc.plsda <- auc(roc.plsda)
auc.plsda
PLSDA <- c(c.plsda$overall[1:2], c.plsda$byClass[c(1:2,5:7)], AUC= auc.plsda)
PLSDA

############### 11.    Logistic Model       #####################################
library(doParallel)
cl <- makePSOCKcluster(7)
registerDoParallel(cl)
set.seed(75)
start <-  Sys.time()
logistic.glm <- train(trainPredictors,
                  trainresponse,
                  method = "glm",
                  metric = "Kappa",
                  preProcess = c( "BoxCox", "center", "scale", "spatialSign" ),
                  trControl = ctrl)
logistic.glm
summary(logistic.glm)
end <-  Sys.time()
duration.nb <- (as.numeric(end) - as.numeric(start)) / 60 # duration in minutes
duration.nb
stopCluster(cl)
```

```
predictedglm <- predict(logistic.glm, trainPredictors)
c.glm <- confusionMatrix(data =predictedglm,
                        reference = trainresponse)
roc.glm <- roc(response = logistic.glm$pred$obs,
               predictor = logistic.glm$pred$Yes)
auc.glm <- auc(roc.glm)
auc.glm
logistic <- c(c.glm$overall[1:2], c.glm$byClass[c(1:2,5:7)], AUC = auc.glm)
logistic

############### 12. Penalized Model      ############################################
metric = "Kappa"
disp_metric = c("Kappa", "Accuracy")
preProc = c("center", "scale" , "BoxCox", "spatialSign")
preProc2 = c("center", "scale")

# Penalized
glmnGrid <- expand.grid(.alpha = c(0, .1, .2, .4, .6, .8, 1),
                        .lambda = seq(.0, .2, length = 10))

glmnTuned <- train(x=trainX,
                   y = trainy,
                   method = "glmnet",
                   tuneGrid = glmnGrid,
                   preProc = preProc,
                   metric = metric,
                   trControl = ctrl)
res = glmnTuned$results
glmnTuned$bestTune
res[c("alpha", "lambda", disp_metric)]
plot(glmnTuned)
library(pROC)
pred <- predict(glmnTuned, trainX)
pred.prob <- predict(glmnTuned, trainX, type = "prob")
c.glmnet <- confusionMatrix(data =pred,
                            reference = trainy)
ROC = roc(response = trainy ,predictor = pred.prob[,2])
Penalized <- c(c.glmnet$overall[1:2], c.glmnet$byClass[c(1:2,5:7)], "ROC"=auc(ROC))
Penalized

################13. Random GLM  ###############################

# Random GLM
ctrl_rglm <- trainControl(method = "cv", number=10,
                          summaryFunction = multiClassSummary,
                          classProbs = TRUE,
                          savePredictions = TRUE)
tunegrid <- expand.grid(maxInteractionOrder=c(1,2))
random_glm <- train(x=trainX,
                    y = trainy,
                    method="randomGLM",
                    metric=metric,
                    preProc=preProc2,
                    tuneGrid=tunegrid,
                    trControl=ctrl_rglm)
random_glm
res <- random_glm$results
res[,c("maxInteractionOrder", disp_metric)]
plot(random_glm)
pred = predict(random_glm, testX)
vals <- data.frame(obs = testy, pred = pred)
```

```
dfSummaray = defaultSummary(vals)
dfSummaray
pred = predict(random_glm, trainX)
pred.prob <- predict(random_glm, trainX, type = "prob")
pred <- ifelse(pred.prob[,1]<0.5, "Yes", "No")
c <- confusionMatrix(data =as.factor(pred), reference = trainy)
ROC = roc(response = trainy ,predictor = pred.prob[,2])
random_glm.stats <- c(c$overall[1:2], c$byClass[c(1:2,5:7)], "ROC"=auc(ROC))
random_glm.stats



################   14. Naive Bayes ############################

nb <- train(x=trainX,
            y = trainy,
            method="naive_bayes",
            metric=metric,
            preProc=preProc,
            tuneGrid = data.frame(laplace=1, usekernel=TRUE, adjust=1),
            trControl=ctrl)
nb
pred <- predict(nb, testX)
vals <- data.frame(obs = testy, pred = pred)
dfSummaray = defaultSummary(vals)
dfSummaray
res <- nb$results
res[, disp_metric]
pred = predict(nb, trainX)
pred.prob <- predict(nb, trainX, type = "prob")
c <- confusionMatrix(data =pred, reference = trainy)
ROC = roc(response = trainy ,predictor = pred.prob[,2])
NaiveBayes <- c(c$overall[1:2], c$byClass[c(1:2,5:7)], "ROC"=auc(ROC))
NaiveBayes



#########################################
###########           Comparison Table
#########################################
# Comparison table
final.metric <- round( rbind(logistic, LDA, QDA, MDA,PLSDA, SVM, KNN, FDA,NNetwork),3)
penals <- round( c( 0.7893033,   0.3977279,   0.9158285,   0.4403893,
0.8186120,   0.9158285,   0.8644957,   0.8160756),3)
penalized <- setNames(penals, colnames(final.metric))
rand.glm <- round( c( 0.7887853,   0.3916121,   0.9193577,
0.4287105,   0.8161028,   0.9193577,   0.8646585,   0.8157560),3)
random_glm <- setNames(rand.glm, colnames(final.metric))

nb <- round( c(0.7160062,   0.2559735,   0.8194812,   0.4306569,   0.7987616,   0.8194812,   0.8089888,
NaiveBayes <- setNames(nb, colnames(final.metric))


Final.metric <- round( rbind( logistic,penalized, LDA,

QDA, MDA,PLSDA, random_glm, NaiveBayes, KNN, SVM, FDA,NNetwork),3)
Final.metric2 <- Final.metric[,-6]
m = dim(Final.metric)[1]
barplot(Final.metric2, xlab = "Performance Profiles of Models",
        main = "Performance Metrics",
        col = rainbow(m),
        beside = TRUE  , xlim = c(0,100),
        legend.text = rownames(final.metric),
        args.legend = list(title = "Model", x = "topright",
                           inset = c(-0.05, 0)), ylim = c(0,1.1)
        )



#Final.metric <- as.data.frame(final.metric)
#ggplot(data = Final.metric) +
#  geom_bar(
#    mapping = aes(x= col.names(Final.metric)), #, fill= colna  ),
#    position = "dodge"
#    )

####

# Best Two Models: Neural network
nnetFit
summary(nnetFit)
# on test test
testpredictednn <- predict(nnetFit, testPredictors)
# Confusion matrix
test_c.nn <-  confusionMatrix(data =testpredictednn,
                               reference = testresponse)
test_c.nn

testpredictedfda <- predict(fda, testPredictors)
# Confusion matrix

test_c.fda <-  confusionMatrix(data =testpredictedfda,
                              reference = testresponse)
test_c.fda


# Important Variables
plot(varImp(nnetFit), 15)


## Apendix: TEst Case With Alan
alan = data.frame("CLM_FREQ5"=0,    # How many accidents in the last 5 years
                  "CLM_AMT5"=0,     # Total cost of accidents in the last 5 years
                  "KIDSDRIV"=0,     # Number of kids who drive
                  "TRAVTIME"=10,    # Commute time to work
                  "BLUEBOOK"=4171,  # Current car price (can google)
                  "RETAINED"=8 ,    # How many years you have been with the insurance company on this p
                  "NPOLICY"= 2,     # Number of policies
                  "MVR_PTS"=0,      # You can guess or buy this info online for $12 (plus some fees)
                  "AGE"=24,         # self explanatory
                  "HOMEKIDS"=0,     # number of kids that live with you
                  "YOJ"=4,          # years at your current job
                  "INCOME"=,        # annual income
                  "HOME_VAL"=0.0,   # value of your home (I rent so I put 0)
                  "SAMEHOME"=3,     # How many years you have lived in your house
                  ## The rest of the columns are for dummy vars and are either 0 or 1. Most are self ex
                  "CAR_USE_Commercial"=0, # the other option is private (so put zero unless you use you
                  "CAR_TYPE_Pickup"=0,
                  "CAR_TYPE_Sedan"=1,
                  "CAR_TYPE_Sports Car"=0,
                  "CAR_TYPE_SUV"=0,
                  "CAR_TYPE_Van"=0,
                  "RED_CAR_yes"=1,
                  "REVOLKED_Yes"=0, # 1 if your driver's license has ever been revoked
                  "GENDER_M"=1,
                  "MARRIED_Yes"=0,
                  "PARENT1_Yes"=0, # 1 if you are a single parent
                  "JOBCLASS_Blue Collar"=0,
                  "JOBCLASS_Clerical"=0,
                  "JOBCLASS_Dr/Manager"=0,
                  "JOBCLASS_Home Maker"=0,
                  "JOBCLASS_Lawyer"=0,
                  "JOBCLASS_Professional"=0,
                  "JOBCLASS_Student"=1,
                  "MAX_EDUC_Bachelors"=1,
                  "MAX_EDUC_High School"=0,
                  "MAX_EDUC_Masters"=0,
                  "MAX_EDUC_PhD"=0,
                  "AREA_Urban"=0 )  # other option is rural. I think that's what houghton counts as

# I must've messed up some of the names
names(alan) <- names(testX)

dim(alan)



#### T H E   E N D!
```