Claim Severity

Jayesh John

6/21/2020

Contents

Preface

This data science project is part of 'Choose Your Own!' *HarvardX: PH125.9x* course, towards HarvardX Professional Certificate in Data Science. The overall purpose of this project is to apply R data analysis and Machine Learning skills to a real-world problems. I have chosen 'Allstate Claim Severity' for this project as it closely matches with my experience and domain expertise in the field of Property and Casualty Insurance.

Introduction

When you've been devastated by a serious car accident, your focus is on the things that matter the most: family, friends, and other loved ones. Pushing paper with your insurance agent is the last place you want your time or mental energy spent. *Allstate*, a personal insurer in the United States, continually seeks fresh ideas to improve their claims service for the over 16 million households they protect.

In 2016, *Allstate* posted an open challenge to develop automated methods of predicting the cost, and hence severity, of claims. The objective is to create an algorithm that accurately predicts claims severity or 'loss'. Allstate provided the dataset where each row in the dataset represents an insurance claim.

For this project, we will be creating Machine Learning models to predict the 'loss' of insurance claims. The target is to predict the value for the 'loss' column and measure the performance of the models using MAE (Mean Absolute Error).

The Claim Severity Dataset

Load Libraries and Dataset

The Claim Severity dataset is provided by Allstate. It is a masked dataset having variables prefaced with 'cat' denoting categorical, and prefaced with 'cont' denoting continuous.

```
############################## ALLSTATE Claim Severity ################################

#########################################################################################
# Step 1: Load the libraries and read the CSV dataset
#########################################################################################

# Note: The below scripts have been coded and executed with R 4.0 version

# The Library loading/installation process could take a couple of minutes

if(!require(tidyverse)) install.packages("tidyverse", repos = "http://cran.us.r-project.org")
if(!require(caret)) install.packages("caret", repos = "http://cran.us.r-project.org")
if(!require(forecast)) install.packages("forecast", repos = "http://cran.us.r-project.org")
if(!require(corrplot)) install.packages("corrplot", repos = "http://cran.us.r-project.org")
if(!require(PCAmixdata)) install.packages("PCAmixdata", repos = "http://cran.us.r-project.org")
if(!require(e1071)) install.packages("e1071", repos = "http://cran.us.r-project.org")
if(!require(xgboost)) install.packages("xgboost", repos = "http://cran.us.r-project.org")
if(!require(viridis)) install.packages("viridis", repos = "http://cran.us.r-project.org")


# Read the CSV file and Load the data:
raw_data <-
```

```
  read.csv("https://raw.githubusercontent.com/jayeshjohn/Claim-Severity/master/Claim%20Severity.csv")

# Check the dataset dimensions and Loss SD, Mean, Range
dim(raw_data)
```

```
## [1] 188318    132
```

```
sd(raw_data$loss)
```

```
## [1] 2904.086
```

```
mean(raw_data$loss)
```

```
## [1] 3037.338
```

```
range(raw_data$loss)
```

```
## [1]      0.67 121012.25
```

```
# remove ID column from training set as it is not needed for training
raw_data$id <- NULL
```

Tidy View of Data

Claim Severity dataset has 188318 rows with each row representing a customer claim and the loss. The dataset has 132 columns or features.

```
# see 'Claim Severity' dataset in tidy format
raw_data %>% tibble()
```

```
## # A tibble: 188,318 x 131
##    cat1  cat2  cat3  cat4  cat5  cat6  cat7  cat8  cat9  cat10 cat11 cat12 cat13
##    <chr> <chr> <chr> <chr> <chr> <chr> <chr> <chr> <chr> <chr> <chr> <chr> <chr>
##  1 A     B     A     B     A     A     A     A     B     A     B     A     A
##  2 A     B     A     A     A     A     A     A     B     B     A     A     A
##  3 A     B     A     A     B     A     A     A     B     B     B     B     B
##  4 B     B     A     B     A     A     A     A     B     A     A     A     A
##  5 A     B     A     B     A     A     A     A     B     B     A     B     A
##  6 A     B     A     A     A     A     A     A     B     A     A     A     A
##  7 A     A     A     A     B     A     A     A     A     A     A     A     A
##  8 A     B     A     B     A     A     A     A     B     A     A     A     A
##  9 A     B     B     B     B     A     A     A     B     B     B     B     B
## 10 A     B     A     A     B     B     A     A     B     A     A     A     A
## # ... with 188,308 more rows, and 118 more variables: cat14 <chr>, cat15 <chr>,
## #   cat16 <chr>, cat17 <chr>, cat18 <chr>, cat19 <chr>, cat20 <chr>,
## #   cat21 <chr>, cat22 <chr>, cat23 <chr>, cat24 <chr>, cat25 <chr>,
## #   cat26 <chr>, cat27 <chr>, cat28 <chr>, cat29 <chr>, cat30 <chr>,
## #   cat31 <chr>, cat32 <chr>, cat33 <chr>, cat34 <chr>, cat35 <chr>,
## #   cat36 <chr>, cat37 <chr>, cat38 <chr>, cat39 <chr>, cat40 <chr>,
```

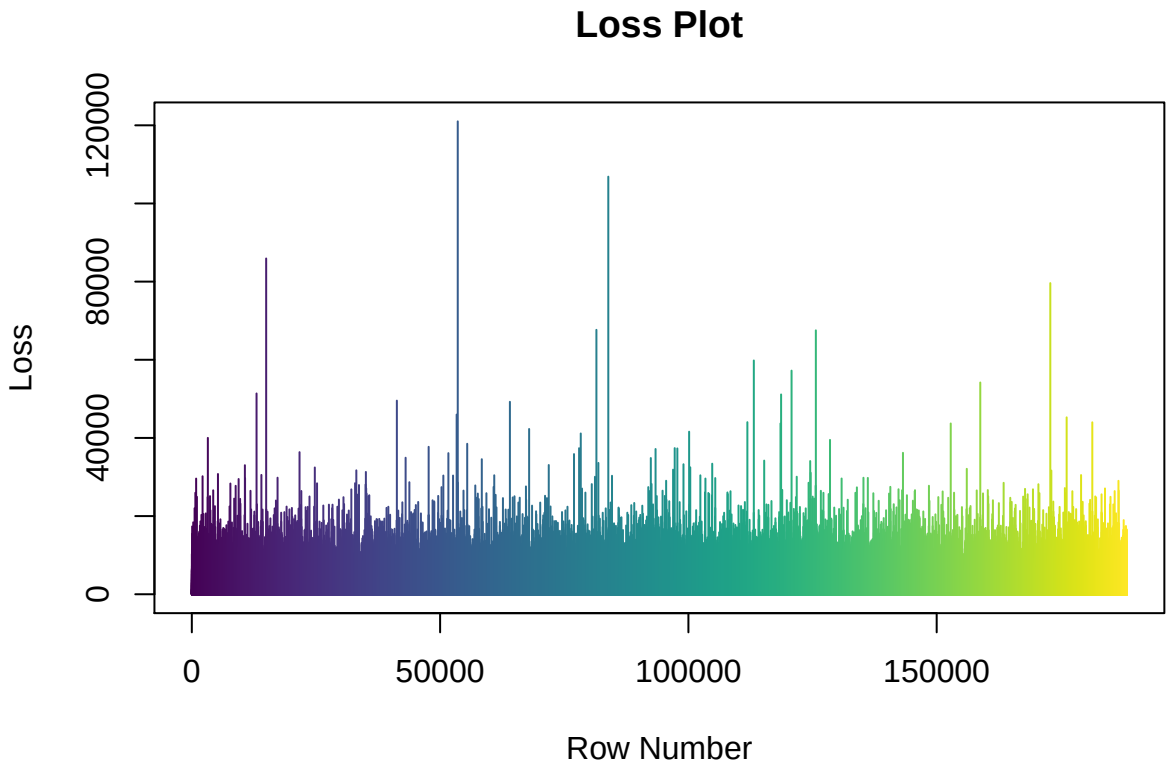
```
## #   cat41 <chr>, cat42 <chr>, cat43 <chr>, cat44 <chr>, cat45 <chr>,
## #   cat46 <chr>, cat47 <chr>, cat48 <chr>, cat49 <chr>, cat50 <chr>,
## #   cat51 <chr>, cat52 <chr>, cat53 <chr>, cat54 <chr>, cat55 <chr>,
## #   cat56 <chr>, cat57 <chr>, cat58 <chr>, cat59 <chr>, cat60 <chr>,
## #   cat61 <chr>, cat62 <chr>, cat63 <chr>, cat64 <chr>, cat65 <chr>,
## #   cat66 <chr>, cat67 <chr>, cat68 <chr>, cat69 <chr>, cat70 <chr>,
## #   cat71 <chr>, cat72 <chr>, cat73 <chr>, cat74 <chr>, cat75 <chr>,
## #   cat76 <chr>, cat77 <chr>, cat78 <chr>, cat79 <chr>, cat80 <chr>,
## #   cat81 <chr>, cat82 <chr>, cat83 <chr>, cat84 <chr>, cat85 <chr>,
## #   cat86 <chr>, cat87 <chr>, cat88 <chr>, cat89 <chr>, cat90 <chr>,
## #   cat91 <chr>, cat92 <chr>, cat93 <chr>, cat94 <chr>, cat95 <chr>,
## #   cat96 <chr>, cat97 <chr>, cat98 <chr>, cat99 <chr>, cat100 <chr>,
## #   cat101 <chr>, cat102 <chr>, cat103 <chr>, cat104 <chr>, cat105 <chr>,
## #   cat106 <chr>, cat107 <chr>, cat108 <chr>, cat109 <chr>, cat110 <chr>,
## #   cat111 <chr>, cat112 <chr>, cat113 <chr>, ...
```

Data Exploration and Visualization

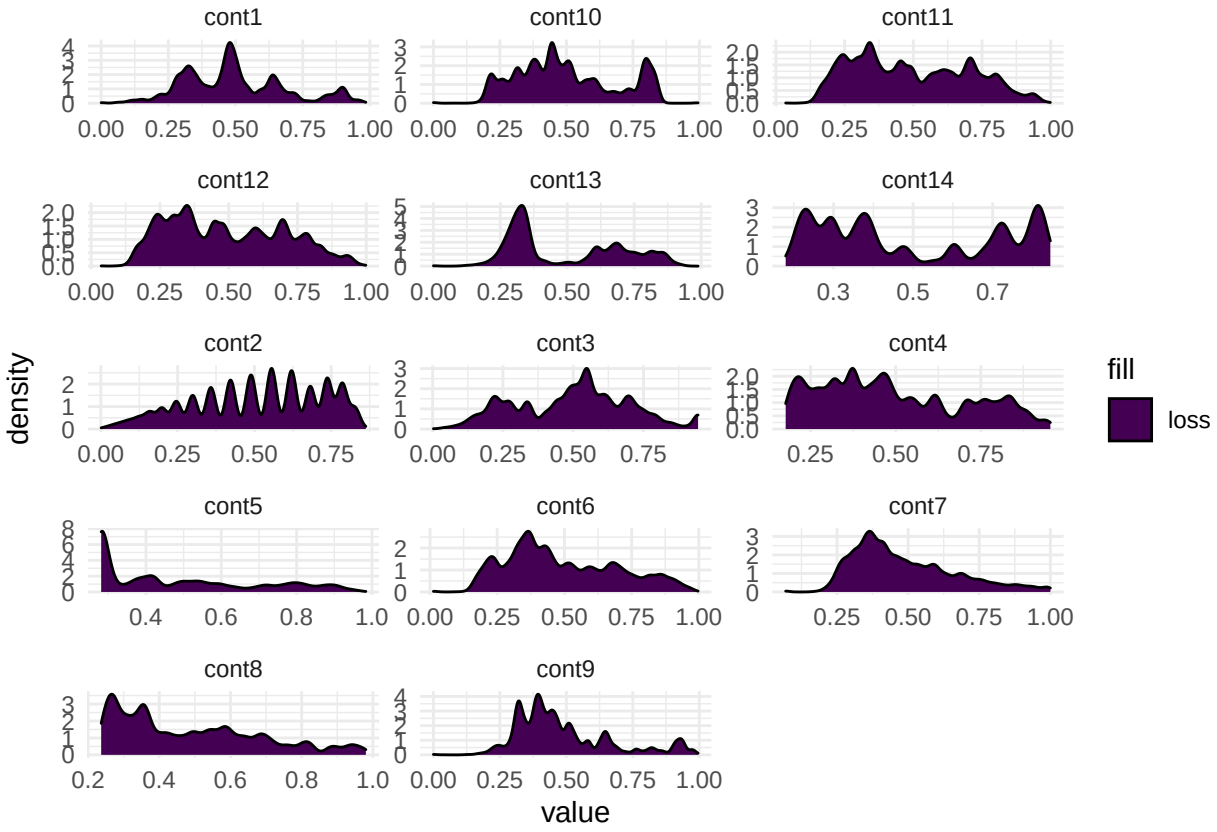
The dataset features are divided into *Categorical* and *Continuous*. There are 116 Categorical variables, 14 Continuous variables and a target loss Continuous variable.

We look at the Continuous variable density distribution and the distribution with the log(loss) using ggplot. The log of loss is taken considering the skewed distribution and to eliminate the outliers.

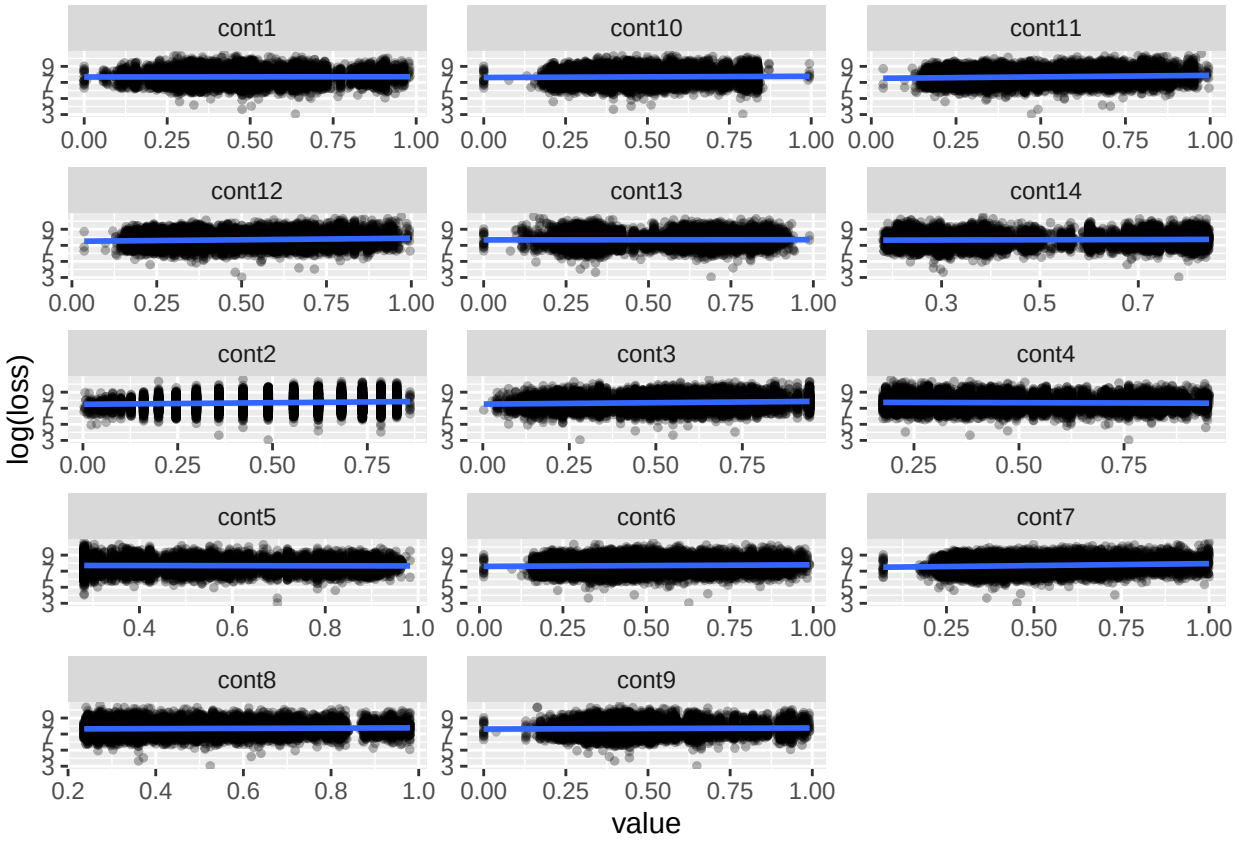
```
################################################################################################
# Step 2: Visualization on dataset features - Categorical and Continuous fields
################################################################################################

# Plot Loss
plot(x = 1:nrow(raw_data), y = raw_data$loss, type = "h", main = "Loss Plot",
     xlab = "Row Number", ylab = "Loss", col = viridis(nrow(raw_data)))
```




```
## Split the dataset having Categorical and Continuous fields
cont_fields <- raw_data[, -grep("^cat", colnames(raw_data))]
cat_fields <- raw_data[,-grep("^cont",colnames(raw_data))]

# Continuous variable Density distribution
cont_fields %>%
  gather(-loss, key = "key", value = "value") %>%
  ggplot(aes(value)) +
  facet_wrap(~ key, scales = "free", ncol = 3) +
  geom_density(aes(fill = "loss")) +
  scale_color_viridis(discrete = TRUE, option = "D")+
  scale_fill_viridis(discrete = TRUE) +
  theme_minimal()
```




```
# Continuous Variable Distribution with Loss
# To reduce the plot size we use 10000 rows
cont_fields_plot <- cont_fields[1:10000,]
cont_fields_plot %>%
  gather(-loss, key = "var", value = "value") %>%
  ggplot(aes(x = value, y = log(loss))) +
  geom_point(size=1, alpha=0.3) +
  geom_smooth(method = lm) +
  theme(legend.position = "bottom") +
  facet_wrap(~ var, scales = "free", ncol = 3)
```



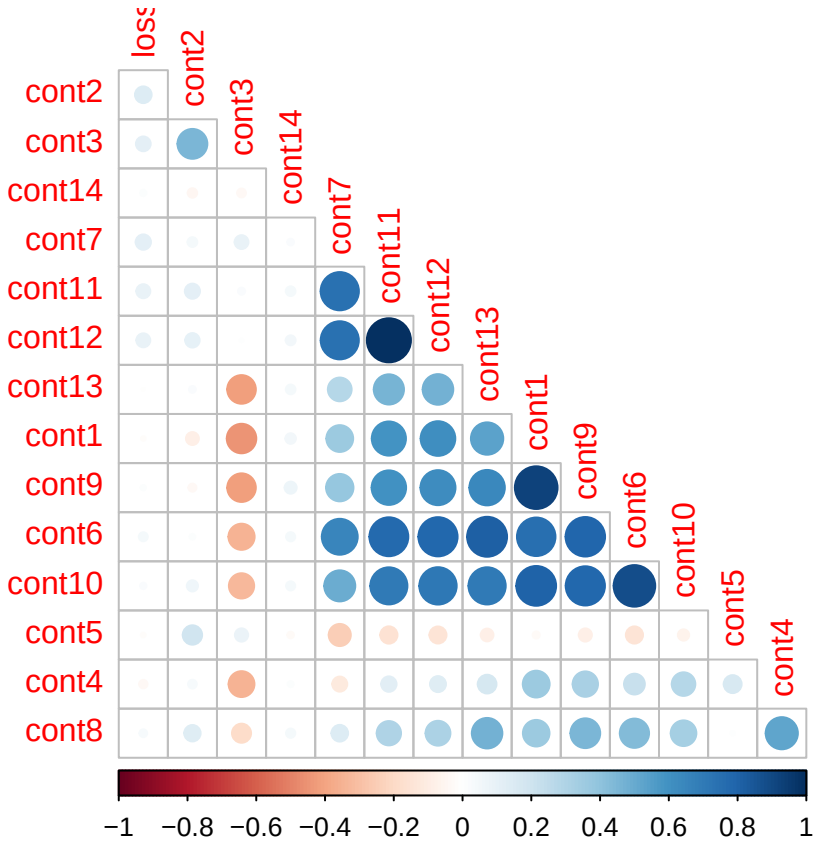
We notice that some of the Continuous variables like cat2 appears to be variables that were originally Categorical.

Data Correlation

Using one single value, Data Correlation describes the *degree of relationship* between two variables. Correlation ranges from -1 to +1. Negative values of correlation indicate that as one variable increases the other variable decreases. Positive values of correlation indicate that as one variable increase the other variable increases as well. Pearson's measures the linear relationship between two variables.

Data Correlation on Continuous variables is shown by the below Corrplot:

```
### Data Pre-Processing

## Treating Continous variables
# Cont fields, plot the correlation
corr <- cor(cont_fields, method = c('pearson'))
corrplot(corr, method = "circle", order = "hclust", type = "lower", diag = FALSE)
```




```
# Running correlation on continuous variables
corr.df <- as.data.frame(as.table(corr))
subset(corr.df[order(-abs(corr.df$Freq)),], (abs(Freq) > 0.8 & abs(Freq) < 1))
```

```
##       Var1   Var2      Freq
## 162 cont12 cont11 0.9943841
## 176 cont11 cont12 0.9943841
## 9    cont9  cont1 0.9299117
## 121  cont1  cont9 0.9299117
## 85  cont10  cont6 0.8833505
## 141  cont6 cont10 0.8833505
## 88  cont13  cont6 0.8150911
## 186  cont6 cont13 0.8150911
## 10  cont10  cont1 0.8085509
## 136  cont1 cont10 0.8085509
```

```
# Removing Variables with correlations greater than 80% - cont9, cont10, cont12, cont13
cont_vars_all <- cont_fields[,-c(9,10,12,13)]
```

After filtering for highly correlated pairs (cont9, cont10, cont12, cont13), we remove them to optimize the overall features and to help with linear regression models.

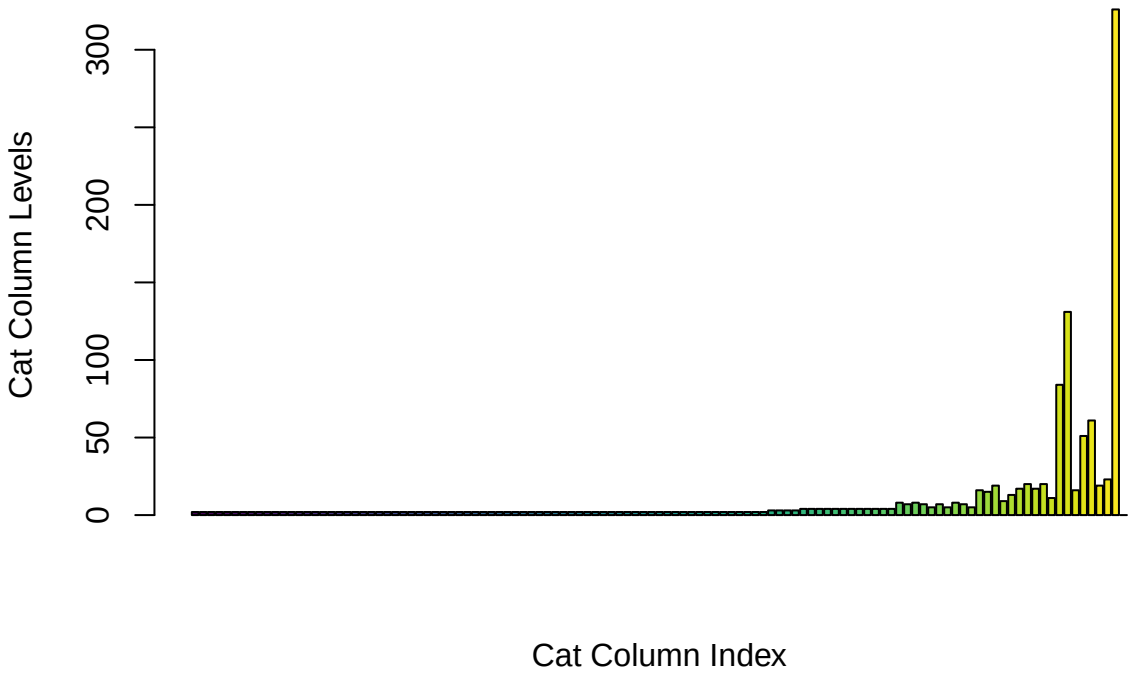
PCA (Principal Component Analysis)

Since we have large number of features, we look at ways to reduce the feature dimension list to help with the modeling and reduce the computation time.

The central idea of principal component analysis (PCA) is to reduce the dimensionality of a data set consisting of a large number of interrelated variables while retaining as much as possible of the variation present in the data set. This is achieved by transforming to a new set of variables, the principal components (PCs), which are uncorrelated, and which are ordered so that the first few retain most of the variation present in all of the original variables.

PCA transformation will be applied on Categorical variables by splitting into three based on the levels. The below bar plot will provide a view of various levels within the categorical variables:

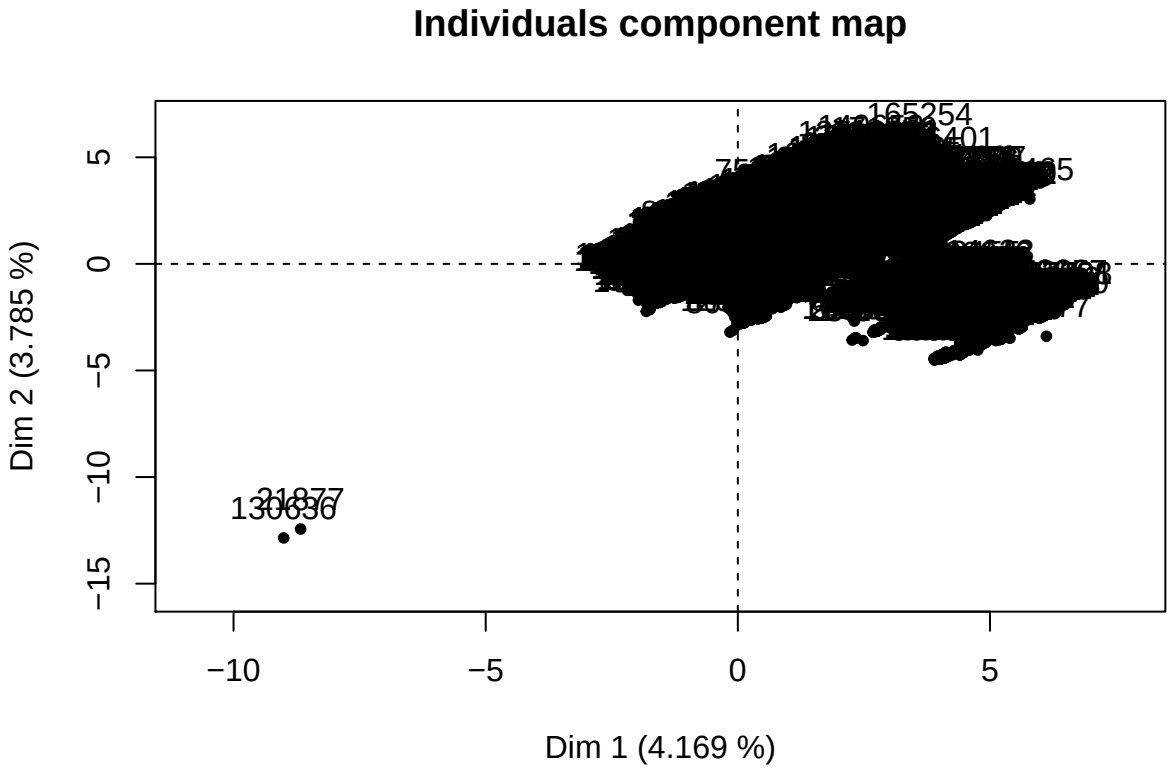
```
##############################################################################################
# Step 3: Apply Principal Component Analysis (PCA) transformation based on levels
##############################################################################################

# Convert Categorical Character fields into Factors
cat_fields <- cat_fields %>% mutate_if(is.character,as.factor)

# Get the length of cat data levels
cat_data_levels <- sapply(seq(1, ncol(cat_fields)), function(d){
  length(levels(cat_fields[,d]))
})

# Plot the cat data levels
barplot(cat_data_levels, type = "l", ylab="Cat Column Levels",
        xlab = "Cat Column Index", col = viridis(117))
```

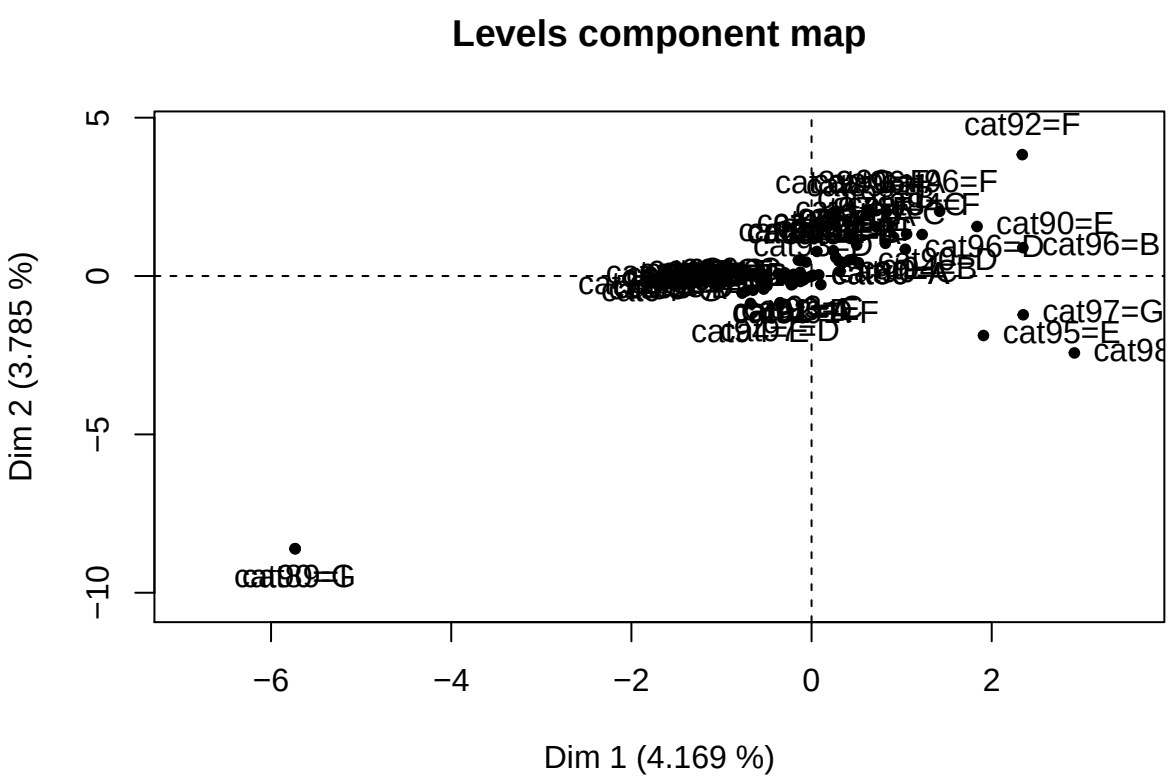


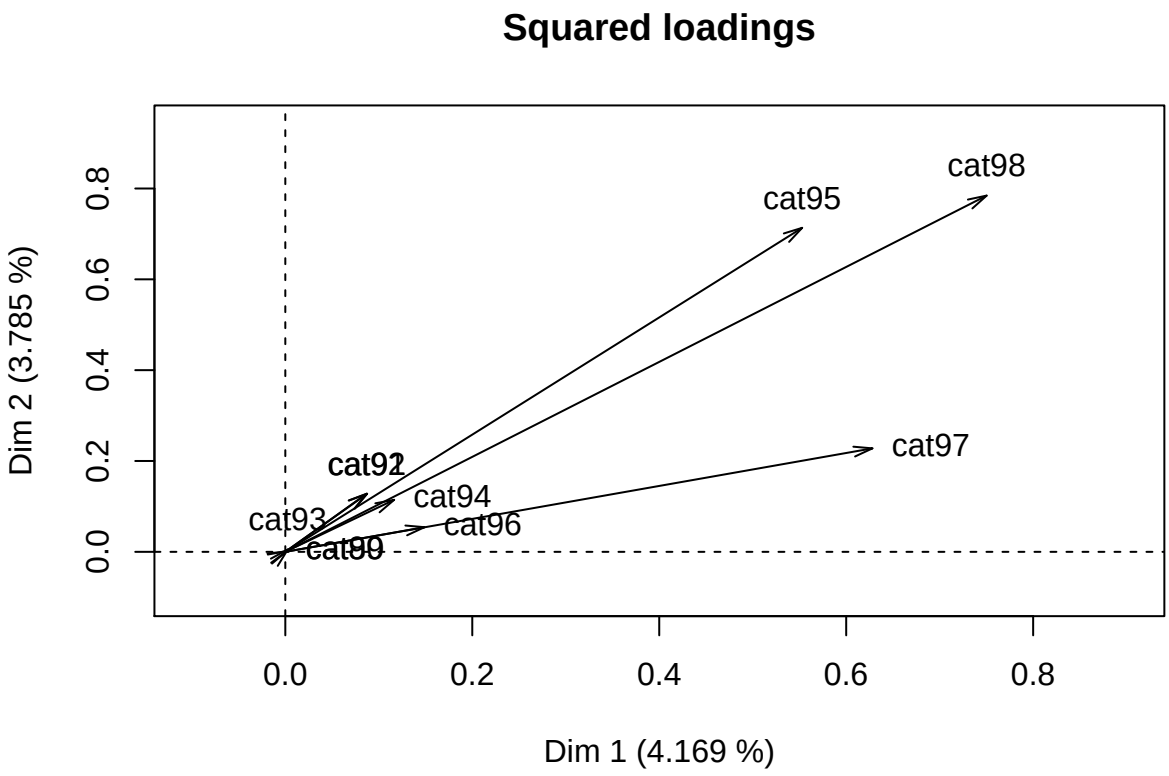
As we see, we have binary levels on 72 Columns and multiple > 2 levels on rest of the columns. We split the Categorical Columns into three based on the levels for applying PCA:

```
#############################################################################
## We split the Categorical columns into three:
#  1. Binary Columns with 2 Levels - Columns 1:72
#  2. Columns with 3-9 Levels - Columns 73:98
#  3. Columns with over 9 Levels - Columns 99:116
#############################################################################

## 1. PCA on Categorical Variables Binary: 2 Levels
# Extracting the binary variables from all_data
binary_all <- cat_fields[,1:72]

# Converting the variables to Numeric
binaryNum_all <- binary_all %>% mutate_if(is.factor,as.numeric)

# Performing PCA on the numeric matrix of binary variables
pca_all <- princomp(binaryNum_all, cor = TRUE, scores = TRUE)

# Choose 25 PCA Scores
Binary_vars_all <- as.data.frame(pca_all$scores[,1:25])


## 2. PCA on Categorical Variables: 3-9 Levels
# Extracting the 3-9 level categorical variables from the whole dataset
```

```r
# Creating two categories
cat1_all <- cat_fields[ ,73:88]
cat2_all <- cat_fields[ ,89:98]

# Running pca on both categories separately
pca_cat1_all <- PCAmix(X.quali = cat1_all, ndim = 15, rename.level = TRUE, graph = FALSE)
pca_cat2_all <- PCAmix(X.quali = cat2_all, ndim = 10, rename.level = TRUE)
```




Levels component map
cat92=F
cat90=E
cat96=D
cat96=B
cat97=G
cat95=E
5
0
−5
−10
−6
−4
−2
0
2
Dim 2 (3.785 %)
Dim 1 (4.169 %)




```
# Extracting the scores to create a dataframe of the new variables from the step above
cat1_vars_all <- data.frame(pca_cat1_all$scores)
cat2_vars_all <- data.frame(pca_cat2_all$scores)

# Changing the column names so as to not mix with cat1 variables
colnames(cat2_vars_all) <- paste("cat", colnames(cat2_vars_all), sep = "_")

## 3. PCA on Categorical Variables: >9 Levels
#extracting all more than 9 level variables and converting them into numeric values
high_level_vars_all <- cat_fields[,99:116]

# Converting variables to Numeric
high_level_vars_all <- high_level_vars_all %>% mutate_if(is.character,as.factor)
high_level_vars_numeric_all <- high_level_vars_all %>% mutate_if(is.factor,as.numeric)

# final dataset
final.dataset.all <- cbind(Binary_vars_all, cat1_vars_all, cat2_vars_all,
                           high_level_vars_numeric_all, cont_vars_all)
ncol(final.dataset.all)
```

```
## [1] 79
```

```
# We see that the data features are transformed into 79 Rows including Loss
```

We transformed the 116 dimensions into 79 dimensions using PCA. This can be further reduced based on the *Cumulative Proportion* derived from the PCA and choosing the variables.

For example, notice that we choose 25 PCA dimensions from the 72 binary variables. The 25 dimensions corresponds to a Cumulative Proportion of 0.57.

We will test our models using PCA transformed data having the reduced dimension as well as with the raw data.

Machine Learning Data Modeling

Train and Test Datasets

The following code is used to generate *Train* and *Test* sets in the ratio 7:3. The Models are trained using the *Train* set. The prediction accuracy of the models are further tested using the *Test* set and MAE determined.

Note that we create two sets of datasets, one having PCA transformed dimensional data and the raw data. We will finally evaluate how these fit in our models. We also convert the raw data with categorical variables into numeric in order to apply xgBoost models.

```
##############################################################################################
# Step 4: Create Train and Test partitions on PCA transformed and Raw dataset
##############################################################################################


# Convert Categorical variables into Numeric data type
raw_data_fac <- raw_data %>% mutate_if(is.character,as.factor)
raw_data_num <- raw_data_fac %>% mutate_if(is.factor,as.numeric)

set.seed(500)

# Raw Data - splitting datasets into 2 - train and test
raw_train_index <- createDataPartition(raw_data_num$loss, p = .7, list = FALSE, times =1)
raw_tr <- raw_data_num[raw_train_index,]
raw_ts <- raw_data_num[-raw_train_index,]

# PCA Data - splitting training into 2 - train and test
train_index <- createDataPartition(final.dataset.all$loss, p = .7, list = FALSE, times =1)
tr <- final.dataset.all[train_index,]
ts <- final.dataset.all[-train_index,]
```

Model 1: The Average Model

Let's start by building the simplest possible model: we predict the same loss (the Mean loss value) for all claims regardless of the features. We will set this as our baseline to evaulate further models.

```
##############################################################################################
# Step 5: Prediction Modeling and compare MAE for each of the Models
##############################################################################################

#########################################################
### MODEL 1: The Average Model
#########################################################

# Naive Model that takes Mean of Train dataset as the loss outcome
```

```
tr <- data.frame(tr)
simple_mean <- mean(raw_data$loss)
pred_var <- ts$loss
pred_var[] <- simple_mean

# Checking the accuracy of the the model using MAE
avgmodel_accuracy <- accuracy(pred_var, ts$loss)
df1 <- as.data.frame(avgmodel_accuracy)
avgmodel_mae <- df1$MAE

# Add MAE results in the table
mae_results <- data.frame(Method = "The Average Model", MAE = avgmodel_mae)
mae_results %>% knitr::kable()
```

Method	MAE
The Average Model	1968.891

Model 2: Linear Regression with PCA

We now apply Linear Regression on PCA transformed dataset with loss as the predictor variable. Mathematically a linear relationship represents a straight line when plotted as a graph.

```
#########################################################
### MODEL 2: Linear Regression with PCA data
#########################################################

fit <- lm(loss ~ ., data = tr)
valid_pred <- predict(fit, ts)

# Checking the accuracy of the the model using MAE
linear_reg_pca_accuracy = accuracy(valid_pred, ts$loss)
df2 <- as.data.frame(linear_reg_pca_accuracy)
linear_reg_pca_mae <- df2$MAE

# Add MAE results in the table
mae_results <- bind_rows(mae_results,
                         data_frame(Method="Linear Regression Model - PCA",
                                    MAE = linear_reg_pca_mae))
mae_results %>% knitr::kable()
```

Method	MAE
The Average Model	1968.891
Linear Regression Model - PCA	1351.492

Model 3: Linear Regression with Raw Data

The third model applies the Linear regression on the raw numeric dataset.

```
#######################################################
### MODEL 3: Linear Regression with Raw data
#######################################################

fit <- lm(loss ~ ., data = raw_tr)
valid_pred <- predict(fit, raw_ts)

# Checking the accuracy of the the model using MAE
linear_reg_raw_accuracy = accuracy(valid_pred, raw_ts$loss)
df3 <- as.data.frame(linear_reg_raw_accuracy)
linear_reg_raw_mae <- df3$MAE

# Add MAE results in the table
mae_results <- bind_rows(mae_results,
                         data_frame(Method="Linear Regression Model - Raw",
                                    MAE = linear_reg_raw_mae))
mae_results %>% knitr::kable()
```

Method	MAE
The Average Model	1968.891
Linear Regression Model - PCA	1351.492
Linear Regression Model - Raw	1335.037

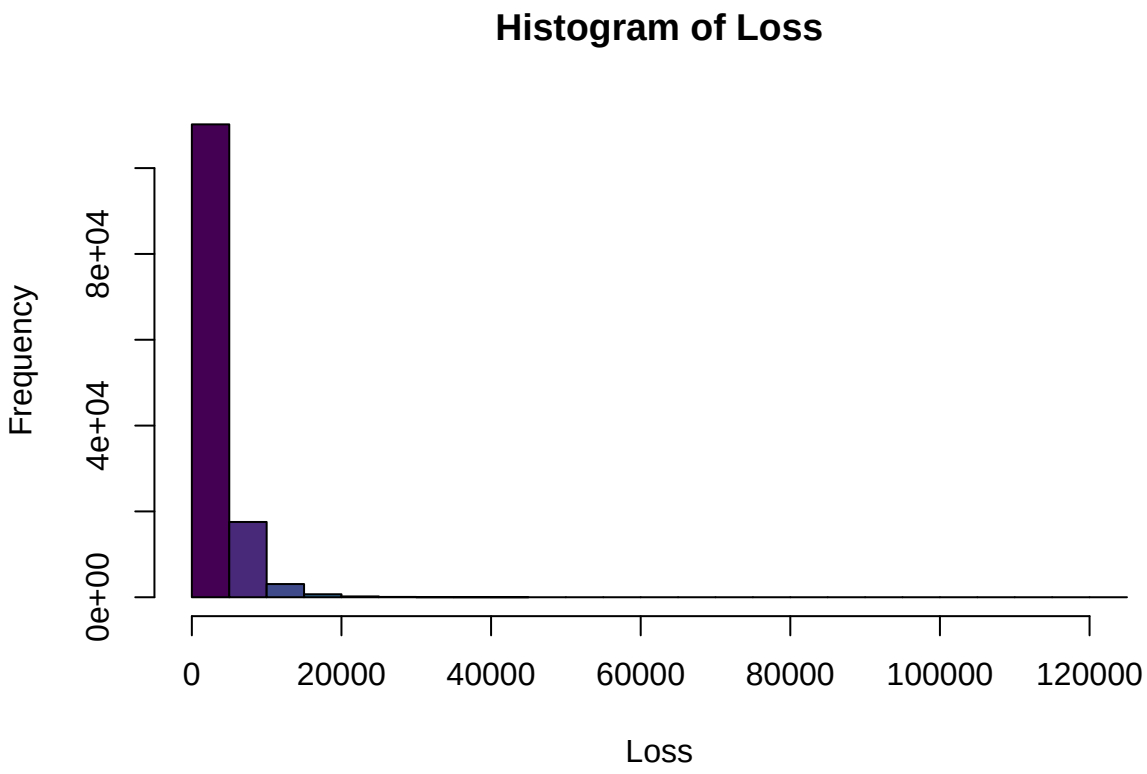
Model 4: xgBoost Model - Using PCA Data

For the next model we choose xgBoost to further improve from our Linear Regression models. Considering the data dimensions and the complexity of the dataset and the need for computational efficiency, this model will be best suited. We apply xgBoost algorithm on numeric variables.

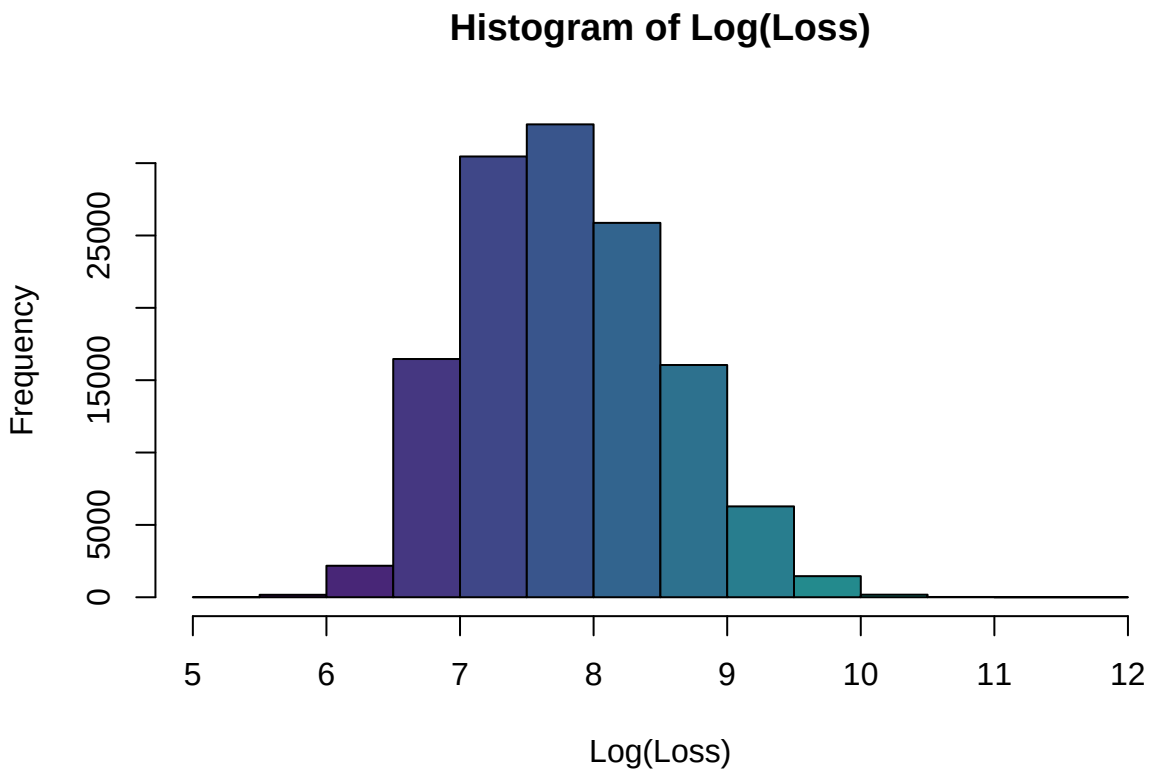
Xgboost is short for *eXtreme Gradient Boosting* package. It has both linear model solver and tree learning algorithms. The capacity to do parallel computation on a single machine makes it fast. Since it works only with numeric vectors, we converted our raw dataset into numeric. We first apply the model on PCA transformed dataset.

The model is best tuned to reduce the MAE. The value of nrounds is chosen based on the value that minimized the MAE during the execution. We use xgb.cv function to determine the best nrounds. For the PCA data the best nround is determined to be *1101* and we use that value during xgb.train.

```
#######################################################
### MODEL 4: xgBoost Model - Using PCA Data
#######################################################

#### Model preparation ####

# Extracting the loss variable from the training set
tr_label <- tr$loss
ts_label <- ts$loss

# Plotting the histogram to check the distribution
hist(tr_label, main="Histogram of Loss", xlab="Loss",col = viridis(10))
```




```
#hist(ts_label, main="Histogram of Loss", xlab="Loss",col = viridis(10))

# Feature engineering - transforming the loss variable to obtain normal-like distribution
tr_label_log <- log(tr$loss + 200)
ts_label_log <- log(ts$loss + 200)

# Plotting the histogram to check the new distribution
hist(tr_label_log, main="Histogram of Log(Loss)", xlab="Log(Loss)",col = viridis(20))
```




```
#hist(ts_label_log, main="Histogram of Log(Loss)", xlab="Log(Loss)",col = viridis(20))

# Converting the train and test dataframes to a matrix
tr_matrix <- as.matrix(tr, rownames.force = NA)
ts_matrix <- as.matrix(ts, rownames.force = NA)

# Converting the train and test dataframes to a sparse matrix
tr_sparse <- as(tr_matrix, "sparseMatrix")
ts_sparse <- as(ts_matrix, "sparseMatrix")

# For xgboost, using xgb.DMatrix to convert data table into a matrix
dtrain <- xgb.DMatrix(data = tr_sparse[,1:78], label = tr_label_log )
dtest <- xgb.DMatrix(data = ts_sparse[,1:78], label = ts_label_log )


## preparation for xgboost model

# defining default parameters
params <- list(booster = "gbtree", objective = "reg:linear", eta=0.1, gamma=0, nthread = 8,
               max_depth=6, min_child_weight=1, subsample=1, colsample_bytree=1,
               early_stopping_rounds = 25, eval_metric = "mae", lambda=0,
               prediction = TRUE, alpha=1)

# Using the xgb.cv function to calculate the best nround for this model.
# Note that we intentionally used nrounds = 3 inorder to save some computational time,
#   but for xgb.train we use the optimal nround value of 1101.
```

```
xgbcv <- xgb.cv( params = params, data = dtrain, nrounds = 3, nfold = 5,
                 showsd = T, stratified = T, print_every_n = 100, maximize = F,
                 verbose = TRUE)
```

```
## [1]  train-mae:6.569767+0.001054 test-mae:6.569754+0.004734
## [3]  train-mae:5.321575+0.000840 test-mae:5.321624+0.004664
```

```
# Finding the best nrounds value
# xgbcv$best_iteration

# Training the model on the best tuning parameters with nrounds = 1101.
# This value is derived based on prior tuning and using it directly on train
#   to save some computation time.

xgb1 <- xgb.train (params = params,
                   data = dtrain,
                   nrounds = 1101,
                   watchlist = list(val=dtest,train=dtrain),
                   print_every_n = 100,
                   verbose = TRUE,
                   maximize = F)
```

```
## [1]  val-mae:6.570143    train-mae:6.569744
## [101]    val-mae:0.382109    train-mae:0.363306
## [201]    val-mae:0.378443    train-mae:0.348768
## [301]    val-mae:0.377694    train-mae:0.338759
## [401]    val-mae:0.377608    train-mae:0.329386
## [501]    val-mae:0.377648    train-mae:0.320274
## [601]    val-mae:0.377587    train-mae:0.312087
## [701]    val-mae:0.377876    train-mae:0.304356
## [801]    val-mae:0.378141    train-mae:0.296980
## [901]    val-mae:0.378470    train-mae:0.290233
## [1001]   val-mae:0.378790    train-mae:0.283696
## [1101]   val-mae:0.379073    train-mae:0.277378
```

```
# Model prediction
xgbpred <- predict(xgb1,dtest)

# Take the antilog of the predictions and subtracting 200 to get the error value
#    in terms of the original loss variable
preds <- exp(xgbpred)-200

# Checking the accuracy of the the model using MAE
xboost_accuracy <- accuracy(preds, ts_label)
df4 <- as.data.frame(xboost_accuracy)
xboost_mae <- df4$MAE

# Add MAE results in the table
mae_results <- bind_rows(mae_results,
                         data_frame(Method="xgBoost Model - PCA",
                                    MAE = xboost_mae))
mae_results %>% knitr::kable()
```

Method	MAE
The Average Model	1968.891
Linear Regression Model - PCA	1351.492
Linear Regression Model - Raw	1335.037
xgBoost Model - PCA	1167.819

We see a significant improvement from our prior models.

Model 5: xgBoost Model - Using Raw Data

As a final step we apply xgBoost on the raw numeric datset. We use xgb.cv function to determine the best nrounds. For the raw numeric data the best nround is determined to be *1181* and we use this value during xgb.train.

```
#########################################################
### MODEL 5: xgBoost Model - Using Numeric Raw Data
#########################################################

#### Model preparation ####

# Extracting the loss variable from the training set
tr_label <- raw_tr$loss
ts_label <- raw_ts$loss

# Feature engineering- transforming the loss variable to obtain normal-like distribution
tr_label_log <- log(raw_tr$loss + 200)
ts_label_log <- log(raw_ts$loss + 200)

# Converting the train and test dataframes to a matrix
tr_matrix <- as.matrix(raw_tr, rownames.force = NA)
ts_matrix <- as.matrix(raw_ts, rownames.force = NA)

# Converting the train and test dataframes to a sparse matrix
tr_sparse <- as(tr_matrix, "sparseMatrix")
ts_sparse <- as(ts_matrix, "sparseMatrix")

# For xgboost, using xgb.DMatrix to convert data table into a matrix
dtrain <- xgb.DMatrix(data = tr_sparse[,1:130], label = tr_label_log )
dtest <- xgb.DMatrix(data = ts_sparse[,1:130], label = ts_label_log )

## preparation for xgboost model

#defining default parameters
params <- list(booster = "gbtree", objective = "reg:linear", eta=0.1, gamma=0, nthread = 8,
               max_depth=6, min_child_weight=1, subsample=1, colsample_bytree=1,
               early_stopping_rounds = 25, eval_metric = "mae", lambda=0, alpha=1)

# Using the xgb.cv function to calculate the best nround for this model.
# Note that we intentionally use nrounds = 3 inorder to save some computational time,
#  but for xgb.train we use the optimal nround value of 1181 determined during prior execution.
```

```
xgbcv <- xgb.cv( params = params, data = dtrain, nrounds = 3, nfold = 5,
                 showsd = T, stratified = T, print_every_n = 100, maximize = F)
```

```
## [1]  train-mae:6.569377+0.000775 test-mae:6.569358+0.003211
## [3]  train-mae:5.321304+0.000625 test-mae:5.321300+0.002871
```

```
# finding the best nrounds value
#xgbcv$best_iteration

# Training the model on the best tuning parameters with nrounds = 1181.
# This value is derived based on prior tuning and using it directly on train
#   to save some computation time.

xgb1 <- xgb.train (params = params,
                   data = dtrain,
                   nrounds = 1181,
                   watchlist = list(val=dtest,train=dtrain),
                   print_every_n = 100,
                   verbose = TRUE,
                   maximize = F)
```

```
## [1]  val-mae:6.571148    train-mae:6.569365
## [101]    val-mae:0.378364    train-mae:0.366120
## [201]    val-mae:0.374798    train-mae:0.355372
## [301]    val-mae:0.373549    train-mae:0.347330
## [401]    val-mae:0.373036    train-mae:0.340158
## [501]    val-mae:0.372936    train-mae:0.333639
## [601]    val-mae:0.373037    train-mae:0.327380
## [701]    val-mae:0.373209    train-mae:0.321743
## [801]    val-mae:0.373379    train-mae:0.316164
## [901]    val-mae:0.373693    train-mae:0.311074
## [1001]   val-mae:0.374000    train-mae:0.306039
## [1101]   val-mae:0.374290    train-mae:0.301677
## [1181]   val-mae:0.374517    train-mae:0.297824
```

```
# Model prediction
xgbpred <- predict(xgb1,dtest)

# Taking the antilog of the predictions and subtracting 200 to get the error value
#   in terms of the original loss variable
preds <- exp(xgbpred)-200

# Checking the accuracy of the the model using MAE
xboost_raw_accuracy <- accuracy(preds, ts_label)
df5 <- as.data.frame(xboost_raw_accuracy)
xboost_raw_mae <- df5$MAE

# Add MAE results in the table
mae_results <- bind_rows(mae_results,
                         data_frame(Method="xgBoost Model - Raw",
                                    MAE = xboost_raw_mae))
```

Results

We can inspect the MAEs for the various model trained from naive approach using the Average, to the xgBoost approach.

The MAE result table for various models are as follows:

Method	MAE
The Average Model	1968.891
Linear Regression Model - PCA	1351.492
Linear Regression Model - Raw	1335.037
xgBoost Model - PCA	1167.819
xgBoost Model - Raw	1153.630

As we see from the table, the MAEs of each model is an improvement from the previous models, with the last model having an MAE ~ 1153.6. This is obviously a significant improvement from the first naive model.

Conclusion

Accurate prediction of loss or severity is a key important factor for Insurance companies, as it helps drive business decisions and help reduce potential claim processing delays caused during the claims process. It can also help automate some of the manual review and claim adjudication steps reducing the overall claim processing lead time. This will ultimately deliver real business value leading to better overall customer satisfaction.

The application of xgBoost model on raw numeric data achieved the best MAE results. Although PCA data based models yielded slightly less performance, the concept can be applied on multiple scenarios to reduce the data dimension and improve computational and model efficiencies.

Future Impovement

Further improvements to the models can be achieved if we have the knowledge of the complete data features. For example Demographic information of the Insured, Driving History, Loss by State, date and time etc. will help apply domain knowledge for the modeling. Alternate models like Neural Networks can also be considered for future improvement, but with the cost of high computational expense.

References

https://rafalab.github.io/dsbook

https://www.kaggle.com/c/allstate-claims-severity/overview

https://github.com/jayeshjohn/Claim-Severity

https://raw.githubusercontent.com/jayeshjohn/Claim-Severity/master/Claim%20Severity.csv