Problem 2. In this question, we are going to do the case study \Improving Customer Engagement at AMWARE Through Analytics" from HBP. This is an open-ended case study. You can take any approaches that you think are suitable for tackling this case. However, to have a direction to follow, please answer the following questions in addition to those ones that are already included at the end of the case.
(a) How does B2B personalized marketing differ from B2C marketing?

In a B2B scenario, the customer is not an individual who makes the decision unlike in B2C. The company might be highly diverse in its needs and the customer might be on different stages of purchase like awareness, interest, desire to purchase or action. Based on this the marketing campaigns will differ for B2B and B2C.

(b) How is the propensity to respond model different from the traditional propensity to buy model?

The traditional propensity to buy model is based on predicting whether the customer will make a purchase or not but the propensity to respond model will predict the customer's next move based on the past decisions made by the customer. This will also set an order of the decisions made by various customer and will form a rule in our model.

(c) What kind of modeling problems can Kiran and Anit expect when the classes are not represented adequately? Which classification models should be used to handle these problems?

The extraordinary quantity of data will lead to problems like having a highly imbalanced dataset as the multi-class variables being highly sparse. This can be solved using ensemble models for multinomial classification problem and not just using a single model.

(d) VMW has identified more than 600 predictor variables. The data is also highly imbalanced (why?). Do you think that techniques such as logistic regression can be applied when the number of variables is large? Justify your answer. If your answer is no, what variable reduction techniques can be used?

The decisions made by every customer on every step is different so the data is therefore imbalanced and the the variables are very scattered. We can use the variance inflation factor (VIF) and set a threshold x. For every variable that exceeds this threshold, we can eliminate the variable in a step-wise stage. However, Using a logistic regression model on a multinomial classification problem where the classes are higher than 3 will lead to class masking problems and our results will be biased. So, we can use regularization techniques like Lasso to perform variable selection.

(e) What are the efficient meta-algorithms that you can use to aggregate the model? Why do you think the model performs better than the individual models?

Some of the efficient meta-algorithms are Bagging, Boosting – Random Forests, Ada-Boost, Gradient Boost that we can use to aggregate the models. They perform better than individual models because they are additive models and they consist of bunch of individual models which helps lower the model bias and variance.

(f) Use the sample data provided to develop a Random Forest model. Comment on the model development and accuracy of the model.

Steps:

1. Combined both training and validation set to maintain consistency in data pre-processing.
2. Checking the number of missing values and removing the columns that have more than 90% of the missing values.
3. Converting all the variables with character class to factor class.
4. Keeping the rest of the missing values in the data as random forest will take care of the missing values by considering it as another class.
5. Splitting the data into Train and Test. Training the random forest model on the train data.
6. Using the trained model, predicting the results on test data and finding the accuracy of our model.

```
setwd("C:/Users/shahv/OneDrive/Desktop/MSBA/IDS 572/Assignments/Assignment_3")

library(SmartEDA)
library(readr)
library(randomForest)
library(zoo)
library(mice)
library(e1071)
library(dplyr)
library(tidyr)
library(factoextra)
library(PCAmixdata)
library(caret)
library(glmnet)

set.seed(1234)
```

1. Random Forest:

```
TD = read_csv("Training.csv")
VD = read_csv("Validation.csv")

RFD = rbind.data.frame(TD, VD)

RFD$target = as.factor(RFD$target)

RFD = RFD[, -which(colMeans(is.na(RFD)) > 0)]

RFD %>% mutate_if(is.character, factor)
```

```r
I = sample(2, nrow(RFD), replace = T, prob = c(0.6, 0.4))

T_D = RFD[I == 1, ]
Test_D = RFD[I == 2, ]

Model = randomForest(target ~ ., data = T_D, mtry = 5, ntree = 100, importance = T)

summary(Model)
```

```
##                 Length Class  Mode
## call                 6 -none- call
## type                 1 -none- character
## predicted        59868 factor numeric
## err.rate           700 -none- numeric
## confusion           42 -none- numeric
## votes           359208 matrix numeric
## oob.times        59868 -none- numeric
## classes              6 -none- character
## importance        5168 -none- numeric
## importanceSD      4522 -none- numeric
## localImportance      0 -none- NULL
## proximity            0 -none- NULL
## ntree                1 -none- numeric
## mtry                 1 -none- numeric
## forest              14 -none- list
## y                59868 factor numeric
## test                 0 -none- NULL
## inbag                0 -none- NULL
## terms                3 terms  call
```

```r
Predict_RF_Train = predict(Model, newdata = T_D, type = "class")

confusionMatrix(Predict_RF_Train, as.factor(T_D$target))
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction     0     1     2     3     4     5
##          0 58190   390     3    11    48   445
##          1     0   566     0     2     0     0
##          2     0     0     6     0     0     0
##          3     0     0     0     6     0     0
##          4     0     0     0     0    57     0
##          5     0     0     0     0     0   144
##
## Overall Statistics
##
##                Accuracy : 0.985
##                  95% CI : (0.984, 0.9859)
##     No Information Rate : 0.972
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.6295
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: 0 Class: 1  Class: 2  Class: 3  Class: 4
## Sensitivity            1.0000 0.592050 0.6666667 0.3157895 0.5428571
## Specificity            0.4654 0.999966 1.0000000 1.0000000 1.0000000
## Pos Pred Value         0.9848 0.996479 1.0000000 1.0000000 1.0000000
## Neg Pred Value         1.0000 0.993423 0.9999499 0.9997828 0.9991975
## Prevalence             0.9720 0.015968 0.0001503 0.0003174 0.0017539
## Detection Rate         0.9720 0.009454 0.0001002 0.0001002 0.0009521
## Detection Prevalence   0.9870 0.009488 0.0001002 0.0001002 0.0009521
## Balanced Accuracy      0.7327 0.796008 0.8333333 0.6578947 0.7714286
##                      Class: 5
## Sensitivity          0.244482
## Specificity          1.000000
## Pos Pred Value       1.000000
## Neg Pred Value       0.992549
## Prevalence           0.009838
## Detection Rate       0.002405
## Detection Prevalence 0.002405
## Balanced Accuracy    0.622241
```

```r
Predict_TEST = predict(Model, newdata = Test_D)

C_M = confusionMatrix(Predict_TEST, Test_D$target)

Accuracy = C_M$overall[1]
```

```
Accuracy
```

```
##  Accuracy
## 0.9750149
```

We get an accuracy of ~97% on the test data.

(g) How different are regularized logistic regression models from standard logistic regression models? When should L1, L2 regularization be used to model the data? Develop a regularized logistic regression model on the given sample data. What insights do you obtain from this model?

Unlike standard logistic regression models, Regularized Logistic regression imposes a penalty factor on the beta co-efficients. This means that if the size of the beta co-efficient is large, it will charge a higher penalty. This will help constrain the size of the beta co-efficient.

When there are large number of predictors and if we can sense that not all variables will be useful to predict the response variable. In such a case, we might need to perform some variable selection and hence we will use LASSO regression i.e. L1 regularization. But in the case when all the predictor variables are useful to predict the response variable then we will use the RIDGE Regression i.e. L2 regularization.

Steps:

1. Splitting the combined data into Train and validation set.
2. In order to apply Lasso, our predictors must be in a matrix form. So converted our variables into matrix form.
3. Performed cross validation to find the optimal value for lambda based on our training data.
4. Predicted the results on the test set using the optimal value of lambda and finding the accuracy.

2. Lasso method for Regularization:

```
TD = read_csv("Training.csv")
VD = read_csv("Validation.csv")

RD = rbind.data.frame(TD, VD)
```

```
RD = RD[, -which(colMeans(is.na(RD)) > 0)]

RD[sapply(RD, is.factor)] = data.matrix(RD[sapply(RD, is.factor)])

RD %>% mutate_if(is.character, factor)
```

```
RD[sapply(RD, is.character)] = lapply(RD[sapply(RD, is.character)], as.factor)

ExpData(RD, 1)
```

```
##                                  Descriptions        Obs
## 1                          Sample size (Nrow)     100012
## 2                    No. of Variables (Ncol)        647
## 3                   No. of Numeric Variables        644
## 4                    No. of Factor Variables          3
## 5                      No. of Text Variables          0
## 6                   No. of Logical Variables          0
## 7                      No. of Date Variables          0
## 8   No. of Zero variance Variables (Uniform)        123
## 9      %. of Variables having complete cases 100% (647)
## 10 %. of Variables having <50% missing cases     0% (0)
## 11 %. of Variables having >50% missing cases     0% (0)
## 12 %. of Variables having >90% missing cases     0% (0)
```

Applying lasso for variable selection:

Finding best parameter value for lambda using cross-validation and predicting on the validation set:

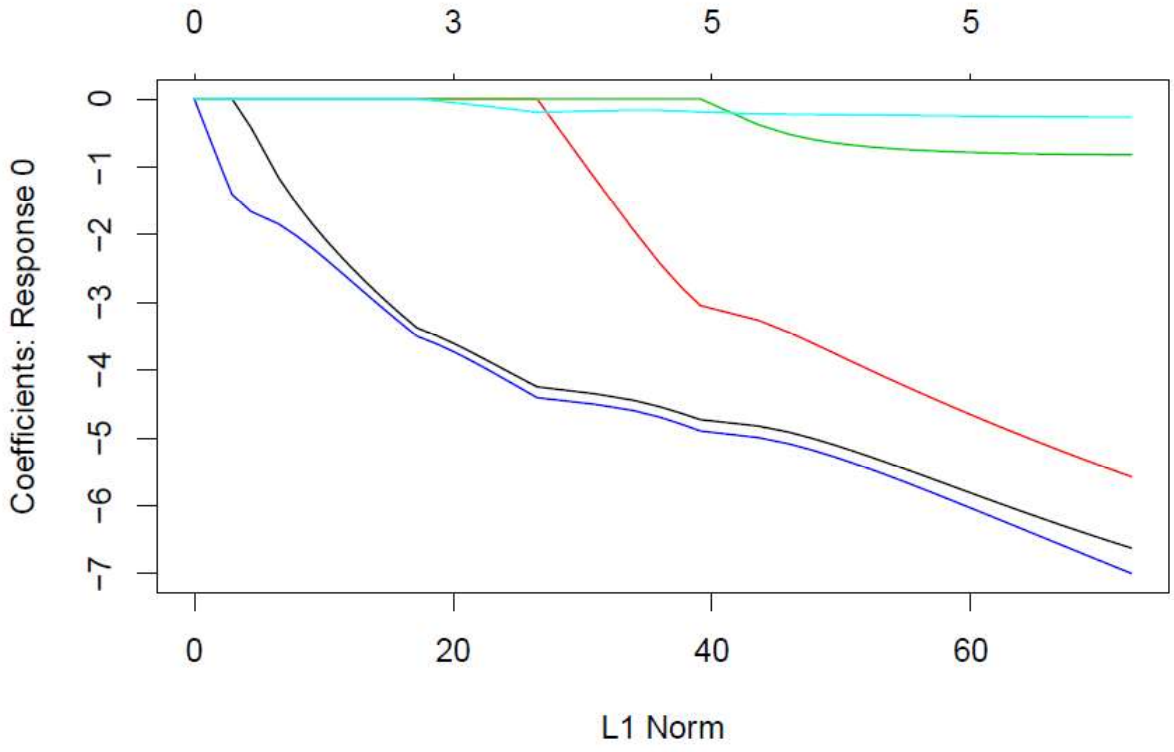
```
RD$target = as.factor(RD$target)

Index = sample(2, nrow(RD), replace = T, prob = c(0.6, 0.4))
Train = RD[Index == 1, ]
Validation = RD[Index == 2, ]

X = model.matrix(Train$target ~ ., data = Train)

V = model.matrix(Validation$target ~ ., data = Validation)
```

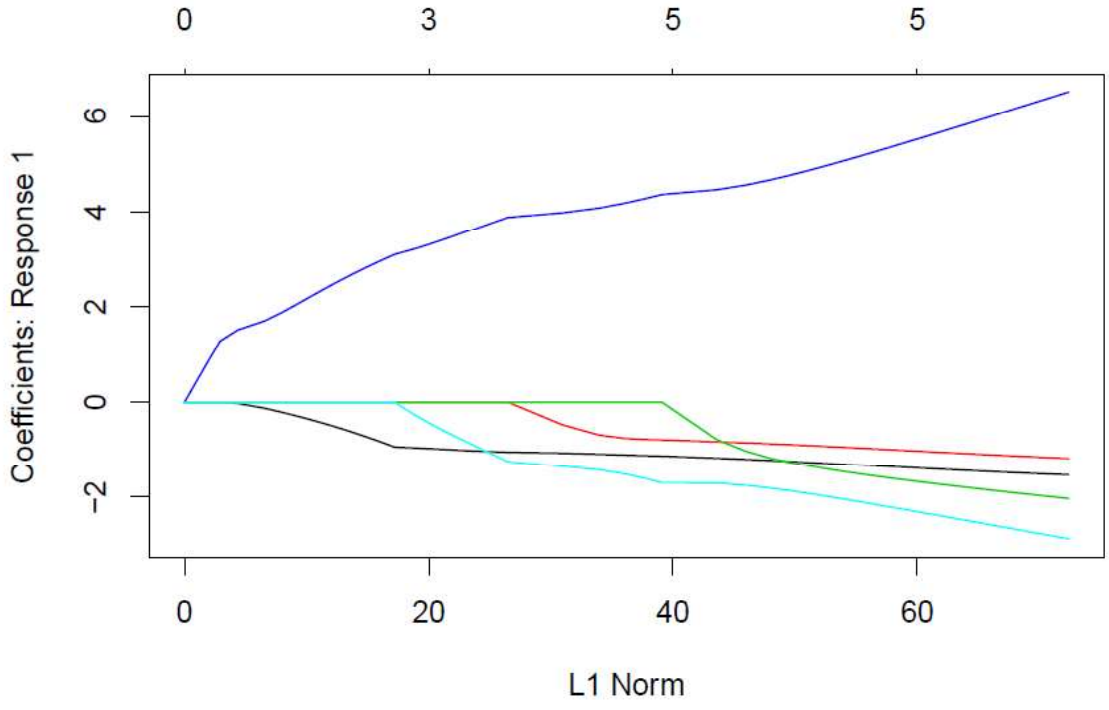
```
cvfit = cv.glmnet(X, Train$target, family = "multinomial", type.multinomial = "grouped")

coef(cvfit, s = "lambda.min")
```

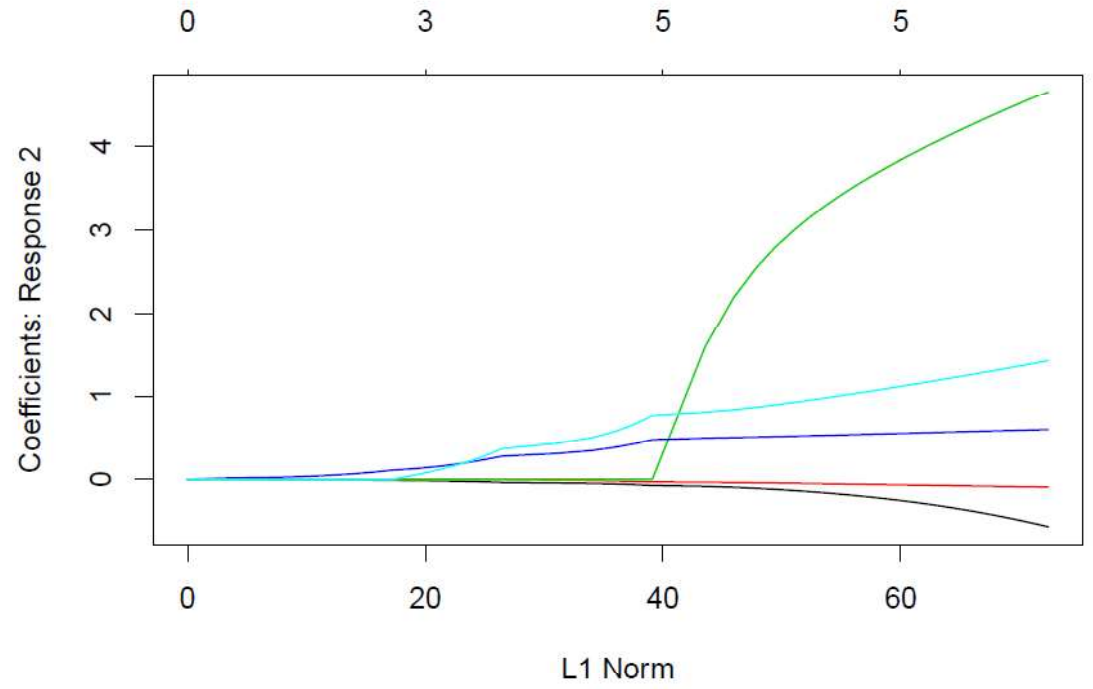
```
Model_Lasso = glmnet(X, Train$target, alpha = 1, family = "multinomial", type.multinomial = "grouped")

Predict_Lasso = predict(cvfit, newx = V, s = "lambda.min", type = "class")

Result = confusionMatrix(as.factor(Predict_Lasso), Validation$target)

Result$overall
```

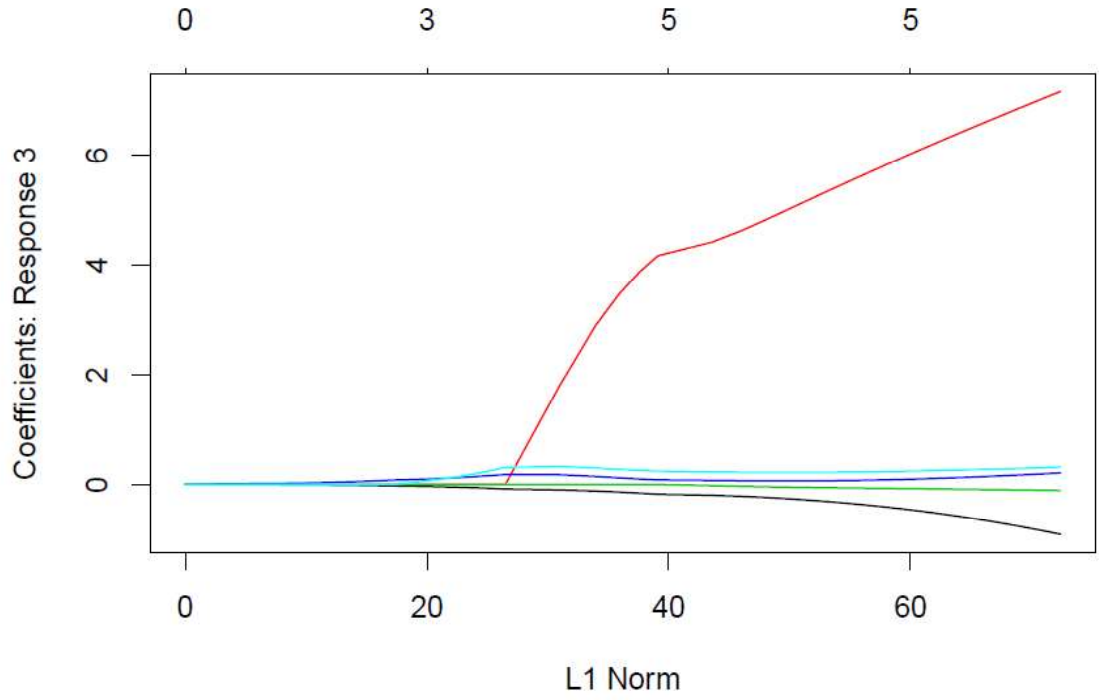
```
##       Accuracy          Kappa  AccuracyLower  AccuracyUpper   AccuracyNull
##      1.0000000      1.0000000      0.9999081      1.0000000      0.9735203
## AccuracyPValue  McnemarPValue
##      0.0000000            NaN
```

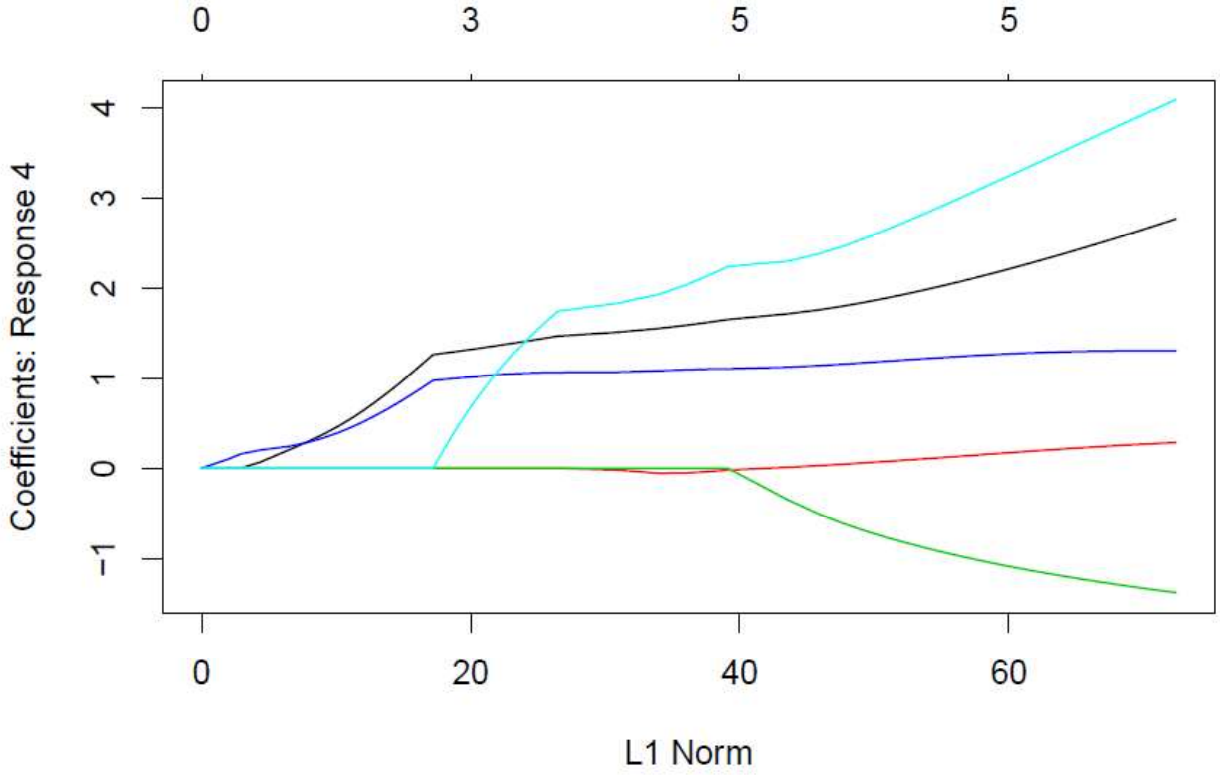
```
plot(Model_Lasso)
```

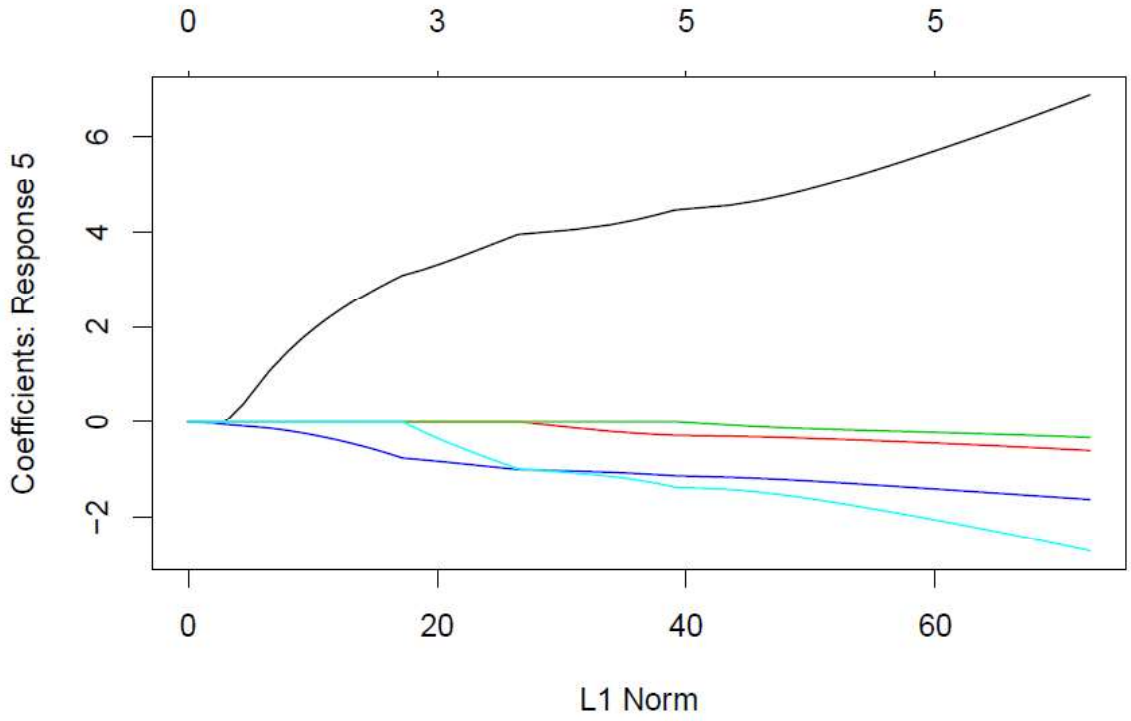





0
3
5
5
Coefficients: Response 2
0
1
2
3
4
0
20
40
60
L1 Norm


0
3
5
5
Coefficients: Response 3
0
2
4
6
0
20
40
60
L1 Norm

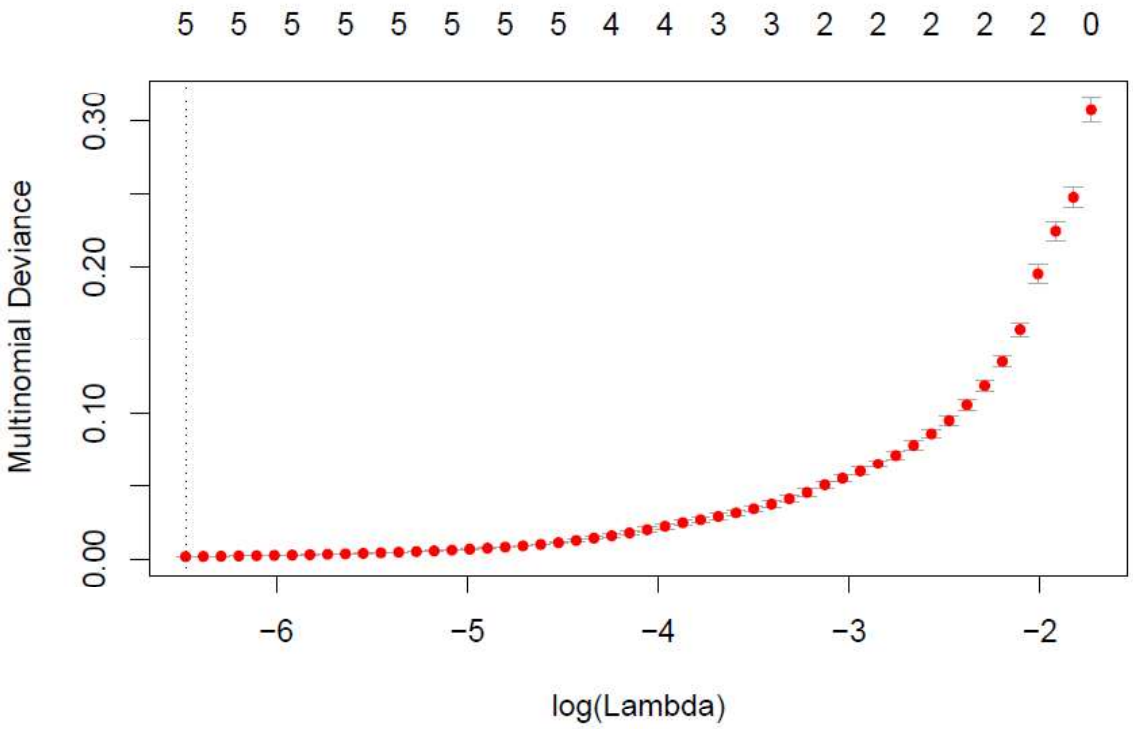

0
3
5
5
Coefficients: Response 4
−1
0
1
2
3
4
0
20
40
60
L1 Norm


0
3
5
5
Coefficients: Response 5
−2
0
2
4
6
0
20
40
60
L1 Norm

```
print(Model_Lasso)
```

```
##
## Call:  glmnet(x = X, y = Train$target, family = "multinomial", alpha = 1,      type.multinomial = "g
##
##        Df      %Dev   Lambda
##  [1,]  0 7.533e-12 0.177200
##  [2,]  1 2.004e-01 0.161400
##  [3,]  1 2.733e-01 0.147100
##  [4,]  2 3.691e-01 0.134000
##  [5,]  2 4.913e-01 0.122100
##  [6,]  2 5.638e-01 0.111300
##  [7,]  2 6.168e-01 0.101400
##  [8,]  2 6.587e-01 0.092380
##  [9,]  2 6.933e-01 0.084180
## [10,]  2 7.225e-01 0.076700
## [11,]  2 7.479e-01 0.069890
## [12,]  2 7.701e-01 0.063680
## [13,]  2 7.898e-01 0.058020
## [14,]  2 8.073e-01 0.052870
## [15,]  2 8.230e-01 0.048170
## [16,]  2 8.371e-01 0.043890
## [17,]  3 8.544e-01 0.039990
## [18,]  3 8.683e-01 0.036440
## [19,]  3 8.801e-01 0.033200
## [20,]  3 8.902e-01 0.030250
## [21,]  3 8.992e-01 0.027560
## [22,]  3 9.071e-01 0.025120
## [23,]  3 9.143e-01 0.022880
## [24,]  3 9.207e-01 0.020850
## [25,]  4 9.317e-01 0.019000
## [26,]  4 9.397e-01 0.017310
## [27,]  4 9.458e-01 0.015770
## [28,]  4 9.509e-01 0.014370
## [29,]  4 9.554e-01 0.013100
## [30,]  5 9.612e-01 0.011930
## [31,]  5 9.652e-01 0.010870
## [32,]  5 9.686e-01 0.009906
## [33,]  5 9.716e-01 0.009026
## [34,]  5 9.742e-01 0.008224
## [35,]  5 9.766e-01 0.007494
## [36,]  5 9.787e-01 0.006828
## [37,]  5 9.806e-01 0.006221
## [38,]  5 9.823e-01 0.005669
## [39,]  5 9.838e-01 0.005165
## [40,]  5 9.852e-01 0.004706
## [41,]  5 9.865e-01 0.004288
## [42,]  5 9.877e-01 0.003907
## [43,]  5 9.887e-01 0.003560
## [44,]  5 9.897e-01 0.003244
## [45,]  5 9.905e-01 0.002956
## [46,]  5 9.913e-01 0.002693
## [47,]  5 9.921e-01 0.002454
## [48,]  5 9.927e-01 0.002236
## [49,]  5 9.933e-01 0.002037
## [50,]  5 9.939e-01 0.001856
## [51,]  5 9.944e-01 0.001691
## [52,]  5 9.948e-01 0.001541
```

```
plot(cvfit)
```


5 5 5 5 5 5 5 5 4 4 3 3 2 2 2 2 2 0
Multinomial Deviance
0.00
0.10
0.20
0.30
−6
−5
−4
−3
−2
log(Lambda)

(h) Develop a couple of extreme gradient boosting models with different values for parameters (depth, eta, etc.) Discuss how the models differ from each other.

Steps:

1. Converting our target variable into integer and giving a label.
2. Removing our target variable from our dataset and storing it in different vector.
3. Splitting our data into train and test set.
4. Transforming the two data sets into xgb.Matrix in order to perform gradient boosting as a requirement of the package.
5. Defining the parameters for multinomial classification of xgboost
6. Training our model based on the training set.
7. Predicting results on the test set based on our training model and finding our accuracy.

R Code

1st Model – depth =5, eta = 0.001, gamma =3:

```
Data$target = as.factor(Data$target)

Target <- Data$target
label <- as.integer(Data$target)-1
Data$target = NULL

Data <- Data[,-which(colMeans(is.na(Data))>0)]

Data[sapply(Data,is.factor)]=data.matrix(Data[sapply(Data,is.factor)])
Data%>%
  mutate_if(is.character,factor)
Data[sapply(Data, is.character)]= lapply(Data[sapply(Data, is.character)],as.factor)

n = nrow(Data)
train.index = sample(n,floor(0.75*n))
train.data = as.matrix(Data[train.index,])
train.label = label[train.index]
test.data = as.matrix(Data[-train.index,])
test.label = label[-train.index]

num_class = length(levels(Target))
params = list( booster="gbtree",  eta=0.001,  max_depth=5,  gamma=3,  colsample_bytree=1,
objective="multi:softmax",  eval_metric="mlogloss",  num_class=num_class )

xgb.fit=xgb.train(  params=params,  data=xgb.train,  nrounds=10000,  early_stopping_rounds=5,
watchlist=list(val1=xgb.train,val2=xgb.test),  verbose=0 )

xgb.pred = predict(xgb.fit,test.data,reshape=T)
xgb.pred = as.data.frame(xgb.pred)
```

```
> xgb.fit
##### xgb.Booster
raw: 16.5 Mb
call:
  xgb.train(params = params, data = xgb.train, nrounds = 10000,
    watchlist = list(val1 = xgb.train, val2 = xgb.test), verbose = 0,
    early_stopping_rounds = 10)
params (as set within xgb.train):
  booster = "gbtree", eta = "0.001", max_depth = "5", gamma = "3", subsample = "0.75",
colsample_bytree = "1", objective = "multi:softprob", eval_metric = "mlogloss", num_cl
ass = "6", silent = "1"
xgb.attributes:
  best_iteration, best_msg, best_ntreelimit, best_score, niter
callbacks:
  cb.evaluation.log()
  cb.early.stop(stopping_rounds = early_stopping_rounds, maximize = maximize,
    verbose = verbose)
# of features: 269
niter: 8346
```

```
best_iteration : 8336
best_ntreelimit : 8336
best_score : 0.009194
nfeatures : 269
evaluation_log:
    iter val1_mlogloss val2_mlogloss
       1      1.788791      1.788772
       2      1.785786      1.785797
---
    8345      0.004990      0.009194
    8346      0.004990      0.009194
```

```
colnames(xgb.pred) = levels(Target)
xgb.pred$prediction = apply(xgb.pred,1,function(x) colnames(xgb.pred)[which.max(x)])
xgb.pred$label = levels(Target)[test.label+1]
```

```
> xgb.pred
             0            1            2            3            4
1   0.99886501 0.0002564198 0.0001899631 0.0002283287 0.0002082506
2   0.99886501 0.0002564198 0.0001899631 0.0002283287 0.0002082506
3   0.99886501 0.0002564198 0.0001899631 0.0002283287 0.0002082506
4   0.99886501 0.0002564198 0.0001899631 0.0002283287 0.0002082506
5   0.99886501 0.0002564198 0.0001899631 0.0002283287 0.0002082506
6   0.99886501 0.0002564198 0.0001899631 0.0002283287 0.0002082506
7   0.99886501 0.0002564198 0.0001899631 0.0002283287 0.0002082506
8   0.99886501 0.0002564198 0.0001899631 0.0002283287 0.0002082506
9   0.99886501 0.0002564198 0.0001899631 0.0002283287 0.0002082506
10  0.99886501 0.0002564198 0.0001899631 0.0002283287 0.0002082506
11  0.01780964 0.9605309963 0.0046793702 0.0056425650 0.0051298467
12  0.99886501 0.0002564198 0.0001899631 0.0002283287 0.0002082506
13  0.99886501 0.0002564198 0.0001899631 0.0002283287 0.0002082506
14  0.99886501 0.0002564198 0.0001899631 0.0002283287 0.0002082506
15  0.99886501 0.0002564198 0.0001899631 0.0002283287 0.0002082506
16  0.99886501 0.0002564198 0.0001899631 0.0002283287 0.0002082506
17  0.99886501 0.0002564198 0.0001899631 0.0002283287 0.0002082506
18  0.99886501 0.0002564198 0.0001899631 0.0002283287 0.0002082506
19  0.99886501 0.0002564198 0.0001899631 0.0002283287 0.0002082506
20  0.99886501 0.0002564198 0.0001899631 0.0002283287 0.0002082506
21  0.99886501 0.0002564198 0.0001899631 0.0002283287 0.0002082506
22  0.99886501 0.0002564198 0.0001899631 0.0002283287 0.0002082506
23  0.99886501 0.0002564198 0.0001899631 0.0002283287 0.0002082506
24  0.99886501 0.0002564198 0.0001899631 0.0002283287 0.0002082506
25  0.99886501 0.0002564198 0.0001899631 0.0002283287 0.0002082506
26  0.99886501 0.0002564198 0.0001899631 0.0002283287 0.0002082506
27  0.99886501 0.0002564198 0.0001899631 0.0002283287 0.0002082506
28  0.99886501 0.0002564198 0.0001899631 0.0002283287 0.0002082506
29  0.99886501 0.0002564198 0.0001899631 0.0002283287 0.0002082506
30  0.99886501 0.0002564198 0.0001899631 0.0002283287 0.0002082506
31  0.99886501 0.0002564198 0.0001899631 0.0002283287 0.0002082506
32  0.99886501 0.0002564198 0.0001899631 0.0002283287 0.0002082506
33  0.99886501 0.0002564198 0.0001899631 0.0002283287 0.0002082506
34  0.99886501 0.0002564198 0.0001899631 0.0002283287 0.0002082506
35  0.99886501 0.0002564198 0.0001899631 0.0002283287 0.0002082506
36  0.99886501 0.0002564198 0.0001899631 0.0002283287 0.0002082506
37  0.99886501 0.0002564198 0.0001899631 0.0002283287 0.0002082506
38  0.99886501 0.0002564198 0.0001899631 0.0002283287 0.0002082506
39  0.99886501 0.0002564198 0.0001899631 0.0002283287 0.0002082506
40  0.99886501 0.0002564198 0.0001899631 0.0002283287 0.0002082506
41  0.99886501 0.0002564198 0.0001899631 0.0002283287 0.0002082506
42  0.99886501 0.0002564198 0.0001899631 0.0002283287 0.0002082506
43  0.99886501 0.0002564198 0.0001899631 0.0002283287 0.0002082506
44  0.99886501 0.0002564198 0.0001899631 0.0002283287 0.0002082506
45  0.99886501 0.0002564198 0.0001899631 0.0002283287 0.0002082506
46  0.99886501 0.0002564198 0.0001899631 0.0002283287 0.0002082506
47  0.99886501 0.0002564198 0.0001899631 0.0002283287 0.0002082506
48  0.99886501 0.0002564198 0.0001899631 0.0002283287 0.0002082506
49  0.99886501 0.0002564198 0.0001899631 0.0002283287 0.0002082506
50  0.99886501 0.0002564198 0.0001899631 0.0002283287 0.0002082506
51  0.99886501 0.0002564198 0.0001899631 0.0002283287 0.0002082506
52  0.99886501 0.0002564198 0.0001899631 0.0002283287 0.0002082506
53  0.99886501 0.0002564198 0.0001899631 0.0002283287 0.0002082506
54  0.99886501 0.0002564198 0.0001899631 0.0002283287 0.0002082506
55  0.99886501 0.0002564198 0.0001899631 0.0002283287 0.0002082506
56  0.99886501 0.0002564198 0.0001899631 0.0002283287 0.0002082506
57  0.99886501 0.0002564198 0.0001899631 0.0002283287 0.0002082506
58  0.99886501 0.0002564198 0.0001899631 0.0002283287 0.0002082506
```

```
59  0.99886501 0.0002564198 0.0001899631 0.0002283287 0.0002082506
60  0.99886501 0.0002564198 0.0001899631 0.0002283287 0.0002082506
61  0.15505612 0.0053074691 0.0039319214 0.0047260290 0.0158207994
62  0.99886501 0.0002564198 0.0001899631 0.0002283287 0.0002082506
63  0.99886501 0.0002564198 0.0001899631 0.0002283287 0.0002082506
64  0.99886501 0.0002564198 0.0001899631 0.0002283287 0.0002082506
65  0.99886501 0.0002564198 0.0001899631 0.0002283287 0.0002082506
66  0.99886501 0.0002564198 0.0001899631 0.0002283287 0.0002082506
67  0.99886501 0.0002564198 0.0001899631 0.0002283287 0.0002082506
68  0.99886501 0.0002564198 0.0001899631 0.0002283287 0.0002082506
69  0.99886501 0.0002564198 0.0001899631 0.0002283287 0.0002082506
70  0.99886501 0.0002564198 0.0001899631 0.0002283287 0.0002082506
71  0.99886501 0.0002564198 0.0001899631 0.0002283287 0.0002082506
72  0.99886501 0.0002564198 0.0001899631 0.0002283287 0.0002082506
73  0.99886501 0.0002564198 0.0001899631 0.0002283287 0.0002082506
74  0.99886501 0.0002564198 0.0001899631 0.0002283287 0.0002082506
75  0.99886501 0.0002564198 0.0001899631 0.0002283287 0.0002082506
76  0.99886501 0.0002564198 0.0001899631 0.0002283287 0.0002082506
77  0.99886501 0.0002564198 0.0001899631 0.0002283287 0.0002082506
78  0.99886501 0.0002564198 0.0001899631 0.0002283287 0.0002082506
79  0.05875704 0.2559243143 0.3652909994 0.0186157916 0.2809317410
80  0.99886501 0.0002564198 0.0001899631 0.0002283287 0.0002082506
81  0.99886501 0.0002564198 0.0001899631 0.0002283287 0.0002082506
82  0.99886501 0.0002564198 0.0001899631 0.0002283287 0.0002082506
83  0.99886501 0.0002564198 0.0001899631 0.0002283287 0.0002082506
84  0.99886501 0.0002564198 0.0001899631 0.0002283287 0.0002082506
85  0.99886501 0.0002564198 0.0001899631 0.0002283287 0.0002082506
86  0.99886501 0.0002564198 0.0001899631 0.0002283287 0.0002082506
87  0.99886501 0.0002564198 0.0001899631 0.0002283287 0.0002082506
88  0.99886501 0.0002564198 0.0001899631 0.0002283287 0.0002082506
89  0.99886501 0.0002564198 0.0001899631 0.0002283287 0.0002082506
90  0.99886501 0.0002564198 0.0001899631 0.0002283287 0.0002082506
91  0.99886501 0.0002564198 0.0001899631 0.0002283287 0.0002082506
92  0.99886501 0.0002564198 0.0001899631 0.0002283287 0.0002082506
93  0.99886501 0.0002564198 0.0001899631 0.0002283287 0.0002082506
94  0.99886501 0.0002564198 0.0001899631 0.0002283287 0.0002082506
95  0.99886501 0.0002564198 0.0001899631 0.0002283287 0.0002082506
96  0.99886501 0.0002564198 0.0001899631 0.0002283287 0.0002082506
97  0.99886501 0.0002564198 0.0001899631 0.0002283287 0.0002082506
98  0.99886501 0.0002564198 0.0001899631 0.0002283287 0.0002082506
99  0.99886501 0.0002564198 0.0001899631 0.0002283287 0.0002082506
100 0.99886501 0.0002564198 0.0001899631 0.0002283287 0.0002082506
101 0.99836701 0.0003689327 0.0002733156 0.0003285156 0.0002996274
102 0.99886501 0.0002564198 0.0001899631 0.0002283287 0.0002082506
103 0.99886501 0.0002564198 0.0001899631 0.0002283287 0.0002082506
104 0.99886501 0.0002564198 0.0001899631 0.0002283287 0.0002082506
105 0.99886501 0.0002564198 0.0001899631 0.0002283287 0.0002082506
106 0.99886501 0.0002564198 0.0001899631 0.0002283287 0.0002082506
107 0.99886501 0.0002564198 0.0001899631 0.0002283287 0.0002082506
108 0.99886501 0.0002564198 0.0001899631 0.0002283287 0.0002082506
109 0.99886501 0.0002564198 0.0001899631 0.0002283287 0.0002082506
110 0.99886501 0.0002564198 0.0001899631 0.0002283287 0.0002082506
111 0.99886501 0.0002564198 0.0001899631 0.0002283287 0.0002082506
112 0.99886501 0.0002564198 0.0001899631 0.0002283287 0.0002082506
113 0.99886501 0.0002564198 0.0001899631 0.0002283287 0.0002082506
114 0.99886501 0.0002564198 0.0001899631 0.0002283287 0.0002082506
115 0.99886501 0.0002564198 0.0001899631 0.0002283287 0.0002082506
116 0.99886501 0.0002564198 0.0001899631 0.0002283287 0.0002082506
117 0.99886501 0.0002564198 0.0001899631 0.0002283287 0.0002082506
118 0.99886501 0.0002564198 0.0001899631 0.0002283287 0.0002082506
119 0.99886501 0.0002564198 0.0001899631 0.0002283287 0.0002082506
120 0.99886501 0.0002564198 0.0001899631 0.0002283287 0.0002082506
121 0.99886501 0.0002564198 0.0001899631 0.0002283287 0.0002082506
122 0.99886501 0.0002564198 0.0001899631 0.0002283287 0.0002082506
123 0.99886501 0.0002564198 0.0001899631 0.0002283287 0.0002082506
124 0.99886501 0.0002564198 0.0001899631 0.0002283287 0.0002082506
125 0.99886501 0.0002564198 0.0001899631 0.0002283287 0.0002082506
              5 prediction label
1   0.0002520051          0     0
2   0.0002520051          0     0
3   0.0002520051          0     0
4   0.0002520051          0     0
5   0.0002520051          0     0
6   0.0002520051          0     0
7   0.0002520051          0     0
8   0.0002520051          0     0
```

9	0.0002520051	0	0
10	0.0002520051	0	0
11	0.0062076538	1	1
12	0.0002520051	0	0
13	0.0002520051	0	0
14	0.0002520051	0	0
15	0.0002520051	0	0
16	0.0002520051	0	0
17	0.0002520051	0	0
18	0.0002520051	0	0
19	0.0002520051	0	0
20	0.0002520051	0	0
21	0.0002520051	0	0
22	0.0002520051	0	0
23	0.0002520051	0	0
24	0.0002520051	0	0
25	0.0002520051	0	0
26	0.0002520051	0	0
27	0.0002520051	0	0
28	0.0002520051	0	0
29	0.0002520051	0	0
30	0.0002520051	0	0
31	0.0002520051	0	0
32	0.0002520051	0	0
33	0.0002520051	0	0
34	0.0002520051	0	0
35	0.0002520051	0	0
36	0.0002520051	0	0
37	0.0002520051	0	0
38	0.0002520051	0	0
39	0.0002520051	0	0
40	0.0002520051	0	0
41	0.0002520051	0	0
42	0.0002520051	0	0
43	0.0002520051	0	0
44	0.0002520051	0	0
45	0.0002520051	0	0
46	0.0002520051	0	0
47	0.0002520051	0	0
48	0.0002520051	0	0
49	0.0002520051	0	0
50	0.0002520051	0	0
51	0.0002520051	0	0
52	0.0002520051	0	0
53	0.0002520051	0	0
54	0.0002520051	0	0
55	0.0002520051	0	0
56	0.0002520051	0	0
57	0.0002520051	0	0
58	0.0002520051	0	0
59	0.0002520051	0	0
60	0.0002520051	0	0
61	0.8151576519	5	5
62	0.0002520051	0	0
63	0.0002520051	0	0
64	0.0002520051	0	0
65	0.0002520051	0	0
66	0.0002520051	0	0
67	0.0002520051	0	0
68	0.0002520051	0	0
69	0.0002520051	0	0
70	0.0002520051	0	0
71	0.0002520051	0	0
72	0.0002520051	0	0
73	0.0002520051	0	0
74	0.0002520051	0	0
75	0.0002520051	0	0
76	0.0002520051	0	0
77	0.0002520051	0	0
78	0.0002520051	0	0
79	0.0204801094	2	2
80	0.0002520051	0	0
81	0.0002520051	0	0
82	0.0002520051	0	0
83	0.0002520051	0	0
84	0.0002520051	0	0

```
85  0.0002520051          0     0
86  0.0002520051          0     0
87  0.0002520051          0     0
88  0.0002520051          0     0
89  0.0002520051          0     0
90  0.0002520051          0     0
91  0.0002520051          0     0
92  0.0002520051          0     0
93  0.0002520051          0     0
94  0.0002520051          0     0
95  0.0002520051          0     0
96  0.0002520051          0     0
97  0.0002520051          0     0
98  0.0002520051          0     0
99  0.0002520051          0     0
100 0.0002520051          0     0
101 0.0003625806          0     0
102 0.0002520051          0     0
103 0.0002520051          0     0
104 0.0002520051          0     0
105 0.0002520051          0     0
106 0.0002520051          0     0
107 0.0002520051          0     0
108 0.0002520051          0     0
109 0.0002520051          0     0
110 0.0002520051          0     0
111 0.0002520051          0     0
112 0.0002520051          0     0
113 0.0002520051          0     0
114 0.0002520051          0     0
115 0.0002520051          0     0
116 0.0002520051          0     0
117 0.0002520051          0     0
118 0.0002520051          0     0
119 0.0002520051          0     0
120 0.0002520051          0     0
121 0.0002520051          0     0
122 0.0002520051          0     0
123 0.0002520051          0     0
124 0.0002520051          0     0
125 0.0002520051          0     0
 [ reached 'max' / getOption("max.print") -- omitted 2375 rows ]
```

```
result = sum(xgb.pred$prediction==xgb.pred$label)/nrow(xgb.pred)
print(paste("Final Accuracy =",sprintf("%1.2f%%", 100*result)))
> print(paste("Final Accuracy =",sprintf("%1.2f%%", 100*result)))
[1] "Final Accuracy = 99.88%"
```

 # 2nd Gradient Boosting Model – depth =3, eta = 0.005, gamma =5:

```
Params_2 = list( booster="gbtree",  eta=0.005,  max_depth=3,  gamma=5,  colsample_bytree=1,
objective="multi:softmax",  eval_metric="mlogloss",  num_class=num_class )
```

```
params_2 = list( booster="gbtree",  eta=0.005,  max_depth=3,  gamma=5,  colsample_bytree=1,
objective="multi:softmax",  eval_metric="mlogloss",  num_class=num_class )

xgb.fit_2=xgb.train(  params=params_2,  data=xgb.train,  nrounds=10000,  early_stopping_rounds=5,
watchlist=list(val1=xgb.train,val2=xgb.test),  verbose=0 )

xgb.fit_2
> xgb.fit_2
##### xgb.Booster
raw: 3.2 Mb
call:
  xgb.train(params = params_2, data = xgb.train, nrounds = 10000,
    watchlist = list(val1 = xgb.train, val2 = xgb.test), verbose = 0,
    early_stopping_rounds = 5)
params (as set within xgb.train):
  booster = "gbtree", eta = "0.005", max_depth = "3", gamma = "5", colsample_bytree =
"1", objective = "multi:softmax", eval_metric = "mlogloss", num_class = "6", silent =
"1"
xgb.attributes:
```

```
  best_iteration, best_msg, best_ntreelimit, best_score, niter
callbacks:
  cb.evaluation.log()
  cb.early.stop(stopping_rounds = early_stopping_rounds, maximize = maximize,
    verbose = verbose)
# of features: 269
niter: 1681
best_iteration : 1676
best_ntreelimit : 1676
best_score : 0.011652
nfeatures : 269
evaluation_log:
    iter val1_mlogloss val2_mlogloss
       1      1.776817      1.776854
       2      1.762152      1.762183
---
    1680      0.006702      0.011654
    1681      0.006702      0.011655
```

```
xgb.pred_2 = predict(xgb.fit_2,test.data,reshape=T)
xgb.pred_2
```

```
> xgb.pred_2
   [1] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
  [35] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
  [69] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 5 0 0 0 0 0 0
 [103] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0
 [137] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 [171] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 [205] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0
 [239] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 [273] 0 0 5 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 [307] 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 [341] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 5 0 0 0 0 0 0 0 0
 [375] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 5
 [409] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 4 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 [443] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 [477] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 5 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 [511] 0 0 0 0 0 1 0 0 0 0 1 0 0 0 0 0 0 0 0 5 0 0 0 0 0 0 5 0 0 0 0 0 0 0
 [545] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 4 0 0 0 0 0 0 0 0 0
 [579] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 [613] 0 0 0 0 0 0 0 0 0 0 0 0 5 0 0 0 0 0 0 0 0 0 0 0 0 0 5 0 0 0 0 0 0 0
 [647] 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 [681] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 4 0 0 0 0 0 0 0 0 0
 [715] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 [749] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 [783] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 1 0 0 0 0 0 0
 [817] 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 [851] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 [885] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 [919] 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 [953] 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 [987] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 [ reached getOption("max.print") -- omitted 2500 entries ]
```

```
xgb.pred_2 = as.data.frame(xgb.pred_2)
colnames(xgb.pred_2) = levels(Target)
xgb.pred_2
```

```
> xgb.pred_2
    xgb.pred_2 prediction label
1            0 xgb.pred_2     0
2            0 xgb.pred_2     0
3            0 xgb.pred_2     0
4            0 xgb.pred_2     0
5            0 xgb.pred_2     0
6            0 xgb.pred_2     0
7            0 xgb.pred_2     0
8            0 xgb.pred_2     0
9            0 xgb.pred_2     0
10           0 xgb.pred_2     0
11           0 xgb.pred_2     0
12           0 xgb.pred_2     0
13           0 xgb.pred_2     0
14           0 xgb.pred_2     0
```

```
15          1 xgb.pred_2     1
16          0 xgb.pred_2     0
17          0 xgb.pred_2     0
18          0 xgb.pred_2     0
19          0 xgb.pred_2     0
20          0 xgb.pred_2     0
21          0 xgb.pred_2     0
22          0 xgb.pred_2     0
23          0 xgb.pred_2     0
24          0 xgb.pred_2     0
25          0 xgb.pred_2     0
26          0 xgb.pred_2     0
27          0 xgb.pred_2     0
28          0 xgb.pred_2     0
29          0 xgb.pred_2     0
30          0 xgb.pred_2     0
31          0 xgb.pred_2     0
32          0 xgb.pred_2     0
33          0 xgb.pred_2     0
34          0 xgb.pred_2     0
35          0 xgb.pred_2     0
36          0 xgb.pred_2     0
37          0 xgb.pred_2     0
38          0 xgb.pred_2     0
39          0 xgb.pred_2     0
40          0 xgb.pred_2     0
41          0 xgb.pred_2     0
42          0 xgb.pred_2     0
43          0 xgb.pred_2     0
44          0 xgb.pred_2     0
45          0 xgb.pred_2     0
46          0 xgb.pred_2     0
47          0 xgb.pred_2     0
48          0 xgb.pred_2     0
49          0 xgb.pred_2     0
50          0 xgb.pred_2     0
51          0 xgb.pred_2     0
52          0 xgb.pred_2     0
53          0 xgb.pred_2     0
54          0 xgb.pred_2     0
55          0 xgb.pred_2     0
56          0 xgb.pred_2     0
57          0 xgb.pred_2     0
58          0 xgb.pred_2     0
59          0 xgb.pred_2     0
60          0 xgb.pred_2     0
61          0 xgb.pred_2     0
62          0 xgb.pred_2     0
63          0 xgb.pred_2     0
64          0 xgb.pred_2     0
65          0 xgb.pred_2     0
66          0 xgb.pred_2     0
67          0 xgb.pred_2     0
68          0 xgb.pred_2     0
69          0 xgb.pred_2     0
70          0 xgb.pred_2     0
71          0 xgb.pred_2     0
72          0 xgb.pred_2     0
73          0 xgb.pred_2     0
74          0 xgb.pred_2     0
75          0 xgb.pred_2     0
76          0 xgb.pred_2     0
77          0 xgb.pred_2     0
78          0 xgb.pred_2     0
79          0 xgb.pred_2     0
80          0 xgb.pred_2     0
81          0 xgb.pred_2     0
82          0 xgb.pred_2     0
83          0 xgb.pred_2     0
84          0 xgb.pred_2     0
85          0 xgb.pred_2     0
86          0 xgb.pred_2     0
87          0 xgb.pred_2     0
88          0 xgb.pred_2     0
89          0 xgb.pred_2     0
90          0 xgb.pred_2     0
```

```
91          0 xgb.pred_2     0
92          0 xgb.pred_2     0
93          1 xgb.pred_2     1
94          0 xgb.pred_2     0
95          0 xgb.pred_2     0
96          5 xgb.pred_2     5
97          0 xgb.pred_2     0
98          0 xgb.pred_2     0
99          0 xgb.pred_2     0
100         0 xgb.pred_2     0
101         0 xgb.pred_2     0
102         0 xgb.pred_2     0
103         0 xgb.pred_2     0
104         0 xgb.pred_2     0
105         0 xgb.pred_2     0
106         0 xgb.pred_2     0
107         0 xgb.pred_2     0
108         0 xgb.pred_2     0
109         0 xgb.pred_2     0
110         0 xgb.pred_2     0
111         0 xgb.pred_2     0
112         0 xgb.pred_2     0
113         0 xgb.pred_2     0
114         0 xgb.pred_2     0
115         0 xgb.pred_2     0
116         0 xgb.pred_2     0
117         0 xgb.pred_2     0
118         0 xgb.pred_2     0
119         0 xgb.pred_2     0
120         0 xgb.pred_2     0
121         0 xgb.pred_2     0
122         0 xgb.pred_2     0
123         0 xgb.pred_2     0
124         0 xgb.pred_2     0
125         0 xgb.pred_2     0
126         0 xgb.pred_2     0
127         0 xgb.pred_2     0
128         0 xgb.pred_2     0
129         0 xgb.pred_2     0
130         1 xgb.pred_2     2
131         0 xgb.pred_2     0
132         0 xgb.pred_2     0
133         0 xgb.pred_2     0
134         0 xgb.pred_2     0
135         0 xgb.pred_2     0
136         0 xgb.pred_2     0
137         0 xgb.pred_2     0
138         0 xgb.pred_2     0
139         0 xgb.pred_2     0
140         0 xgb.pred_2     0
141         0 xgb.pred_2     0
142         0 xgb.pred_2     0
143         0 xgb.pred_2     0
144         0 xgb.pred_2     0
145         0 xgb.pred_2     0
146         0 xgb.pred_2     0
147         0 xgb.pred_2     0
148         0 xgb.pred_2     0
149         0 xgb.pred_2     0
150         0 xgb.pred_2     0
151         0 xgb.pred_2     0
152         0 xgb.pred_2     0
153         0 xgb.pred_2     0
154         0 xgb.pred_2     0
155         0 xgb.pred_2     0
156         0 xgb.pred_2     0
157         0 xgb.pred_2     0
158         0 xgb.pred_2     0
159         0 xgb.pred_2     0
160         0 xgb.pred_2     0
161         0 xgb.pred_2     0
162         0 xgb.pred_2     0
163         0 xgb.pred_2     0
164         0 xgb.pred_2     0
165         0 xgb.pred_2     0
166         0 xgb.pred_2     0
```

167	0	xgb.pred_2	0
168	0	xgb.pred_2	0
169	0	xgb.pred_2	0
170	0	xgb.pred_2	0
171	0	xgb.pred_2	0
172	0	xgb.pred_2	0
173	0	xgb.pred_2	0
174	0	xgb.pred_2	0
175	0	xgb.pred_2	0
176	0	xgb.pred_2	0
177	0	xgb.pred_2	0
178	0	xgb.pred_2	0
179	0	xgb.pred_2	0
180	0	xgb.pred_2	0
181	0	xgb.pred_2	0
182	0	xgb.pred_2	0
183	0	xgb.pred_2	0
184	0	xgb.pred_2	0
185	0	xgb.pred_2	0
186	0	xgb.pred_2	0
187	0	xgb.pred_2	0
188	0	xgb.pred_2	0
189	0	xgb.pred_2	0
190	0	xgb.pred_2	0
191	0	xgb.pred_2	0
192	0	xgb.pred_2	0
193	0	xgb.pred_2	0
194	0	xgb.pred_2	0
195	0	xgb.pred_2	0
196	0	xgb.pred_2	0
197	0	xgb.pred_2	0
198	0	xgb.pred_2	0
199	0	xgb.pred_2	0
200	0	xgb.pred_2	0
201	0	xgb.pred_2	0
202	0	xgb.pred_2	0
203	0	xgb.pred_2	0
204	0	xgb.pred_2	0
205	0	xgb.pred_2	0
206	0	xgb.pred_2	0
207	0	xgb.pred_2	0
208	0	xgb.pred_2	0
209	0	xgb.pred_2	0
210	0	xgb.pred_2	0
211	0	xgb.pred_2	0
212	0	xgb.pred_2	0
213	0	xgb.pred_2	0
214	0	xgb.pred_2	0
215	0	xgb.pred_2	0
216	0	xgb.pred_2	0
217	0	xgb.pred_2	0
218	0	xgb.pred_2	0
219	0	xgb.pred_2	0
220	0	xgb.pred_2	0
221	0	xgb.pred_2	0
222	1	xgb.pred_2	1
223	0	xgb.pred_2	0
224	0	xgb.pred_2	0
225	1	xgb.pred_2	1
226	0	xgb.pred_2	0
227	0	xgb.pred_2	0
228	0	xgb.pred_2	0
229	0	xgb.pred_2	0
230	0	xgb.pred_2	0
231	0	xgb.pred_2	0
232	0	xgb.pred_2	0
233	0	xgb.pred_2	0
234	0	xgb.pred_2	0
235	0	xgb.pred_2	0
236	0	xgb.pred_2	0
237	0	xgb.pred_2	0
238	0	xgb.pred_2	0
239	0	xgb.pred_2	0
240	0	xgb.pred_2	0
241	0	xgb.pred_2	0
242	0	xgb.pred_2	0

```
243          0 xgb.pred_2     0
244          0 xgb.pred_2     0
245          0 xgb.pred_2     0
246          0 xgb.pred_2     0
247          0 xgb.pred_2     0
248          0 xgb.pred_2     0
249          0 xgb.pred_2     0
250          0 xgb.pred_2     0
251          0 xgb.pred_2     0
252          0 xgb.pred_2     0
253          0 xgb.pred_2     0
254          0 xgb.pred_2     0
255          0 xgb.pred_2     0
256          0 xgb.pred_2     0
257          0 xgb.pred_2     0
258          0 xgb.pred_2     0
259          0 xgb.pred_2     0
260          0 xgb.pred_2     0
261          0 xgb.pred_2     0
262          0 xgb.pred_2     0
263          0 xgb.pred_2     0
264          0 xgb.pred_2     0
265          0 xgb.pred_2     0
266          0 xgb.pred_2     0
267          0 xgb.pred_2     0
268          0 xgb.pred_2     0
269          0 xgb.pred_2     0
270          0 xgb.pred_2     0
271          0 xgb.pred_2     0
272          0 xgb.pred_2     0
273          0 xgb.pred_2     0
274          0 xgb.pred_2     0
275          5 xgb.pred_2     5
276          0 xgb.pred_2     0
277          0 xgb.pred_2     0
278          0 xgb.pred_2     0
279          0 xgb.pred_2     0
280          0 xgb.pred_2     0
281          0 xgb.pred_2     0
282          0 xgb.pred_2     0
283          0 xgb.pred_2     0
284          0 xgb.pred_2     0
285          0 xgb.pred_2     0
286          0 xgb.pred_2     0
287          0 xgb.pred_2     0
288          0 xgb.pred_2     0
289          0 xgb.pred_2     0
290          0 xgb.pred_2     0
291          0 xgb.pred_2     0
292          0 xgb.pred_2     0
293          0 xgb.pred_2     0
294          0 xgb.pred_2     0
295          0 xgb.pred_2     0
296          0 xgb.pred_2     0
297          0 xgb.pred_2     0
298          0 xgb.pred_2     0
299          0 xgb.pred_2     0
300          0 xgb.pred_2     0
301          0 xgb.pred_2     0
302          0 xgb.pred_2     0
303          0 xgb.pred_2     0
304          0 xgb.pred_2     0
305          0 xgb.pred_2     0
306          0 xgb.pred_2     0
307          0 xgb.pred_2     0
308          0 xgb.pred_2     0
309          0 xgb.pred_2     0
310          1 xgb.pred_2     1
311          0 xgb.pred_2     0
312          0 xgb.pred_2     0
313          0 xgb.pred_2     0
314          0 xgb.pred_2     0
315          0 xgb.pred_2     0
316          0 xgb.pred_2     0
317          0 xgb.pred_2     0
318          0 xgb.pred_2     0
```

```
319          0 xgb.pred_2     0
320          0 xgb.pred_2     0
321          0 xgb.pred_2     0
322          0 xgb.pred_2     0
323          0 xgb.pred_2     0
324          0 xgb.pred_2     0
325          0 xgb.pred_2     0
326          0 xgb.pred_2     0
327          0 xgb.pred_2     0
328          0 xgb.pred_2     0
329          0 xgb.pred_2     0
330          0 xgb.pred_2     0
331          0 xgb.pred_2     0
332          0 xgb.pred_2     0
333          0 xgb.pred_2     0
 [ reached 'max' / getOption("max.print") -- omitted 3167 rows ]
```

```
xgb.pred_2$prediction = apply(xgb.pred_2,1,function(x) colnames(xgb.pred_2)[which.max(x)])
xgb.pred_2$label = levels(Target)[test.label+1]

result_2 = sum(xgb.pred_2$prediction==xgb.pred_2$label)/nrow(xgb.pred_2)
print(paste("Final Accuracy =",sprintf("%1.2f%%", 100*result_2)))
```

```
> print(paste("Final Accuracy =",sprintf("%1.2f%%", 100*result_2)))
[1] "Final Accuracy = 98.88%"
```

(i) Based on the different models results, what would be your final recommendation to create the propensity to respond model?

Our recommendation to VMWARE is to consider only the variables whose importance is of highest order after reducing the ones that are just large sets of data that have constant/redundant values and not influential on the target variable in a significant way. The data must be first checked for any obvious manual errors and cleansing the data to reduce a few variables are suggested. The best suggested technique as per our analysis is Gradient boosting which gave ~99% accuracy on test data taking less time to run as compared to Random forest model as efficiency is also a priority while fitting the best model.

(j) Discuss the possible deployment strategies for the model results so obtained.

The results obtained after testing various variable reduction techniques and modelling the VMWARE data can be summarized and deployed in the real-world scenario dynamically by real time prediction strategy. This is a significant way of handling deployment in a large organization such as VMWARE as it is required by the management team to decide on possible marketing strategies based on the results obtained and later on implement the best strategies to go on board with the plans.

(k) Discuss the business implications of the models developed here. What will be the value addition to the marketing department and the sales department of VMW because of this exercise?

The business implications to conduct the modelling techniques will be high level of data analysis and collection along with higher computational time required to conduct this exercise.
This exercise will help predict the next course of action of the consumer based on the actions taken till date. Also, this will help set a business rule if there are similar instances.
The unnecessary expenses on sales and marketing can be reduced by analysing the trends in the data and targeting the right set of audience.

In Addition to the above stated requirements, we have also conducted Principle Component Analysis on our data.

```
TD = read_csv("Training.csv")
VD = read_csv("Validation.csv")

Data = rbind.data.frame(TD, VD)
```

Finding missing values:

```
ExpData(TD, 1)
```

```
##                                Descriptions          Obs
## 1                       Sample size (Nrow)        50006
## 2                  No. of Variables (Ncol)          707
## 3                 No. of Numeric Variables          670
## 4                  No. of Factor Variables            0
## 5                    No. of Text Variables           29
## 6                 No. of Logical Variables            4
## 7                    No. of Date Variables            8
## 8   No. of Zero variance Variables (Uniform)          129
## 9      %. of Variables having complete cases 91.51% (647)
## 10 %. of Variables having <50% missing cases    0.71% (5)
## 11 %. of Variables having >50% missing cases   6.22% (44)
## 12 %. of Variables having >90% missing cases   1.56% (11)
```

There are 44 variables with missing values more than 50% and 11 variables with missing values more tha 90%. We will just remove these variables.

Removing the target variable and columns with unknown values:

```
Data = Data[, -which(colMeans(is.na(Data)) > 0)]

Data_1 = Data[, !(names(Data) %in% c("target", "db_industry", "gu_emp_segment_desc",
    "idc_verticals"))]

which(is.na(Data_1))
```

```
## integer(0)
```

```
ExpData(Data_1, 1)
```

```
##                                Descriptions        Obs
## 1                        Sample size (Nrow)     100012
## 2                   No. of Variables (Ncol)        643
## 3                  No. of Numeric Variables        643
## 4                   No. of Factor Variables          0
## 5                     No. of Text Variables          0
## 6                  No. of Logical Variables          0
## 7                     No. of Date Variables          0
## 8   No. of Zero variance Variables (Uniform)        123
## 9      %. of Variables having complete cases 100% (643)
## 10 %. of Variables having <50% missing cases     0% (0)
## 11 %. of Variables having >50% missing cases     0% (0)
## 12 %. of Variables having >90% missing cases     0% (0)
```

Conducting Principle component analysis:

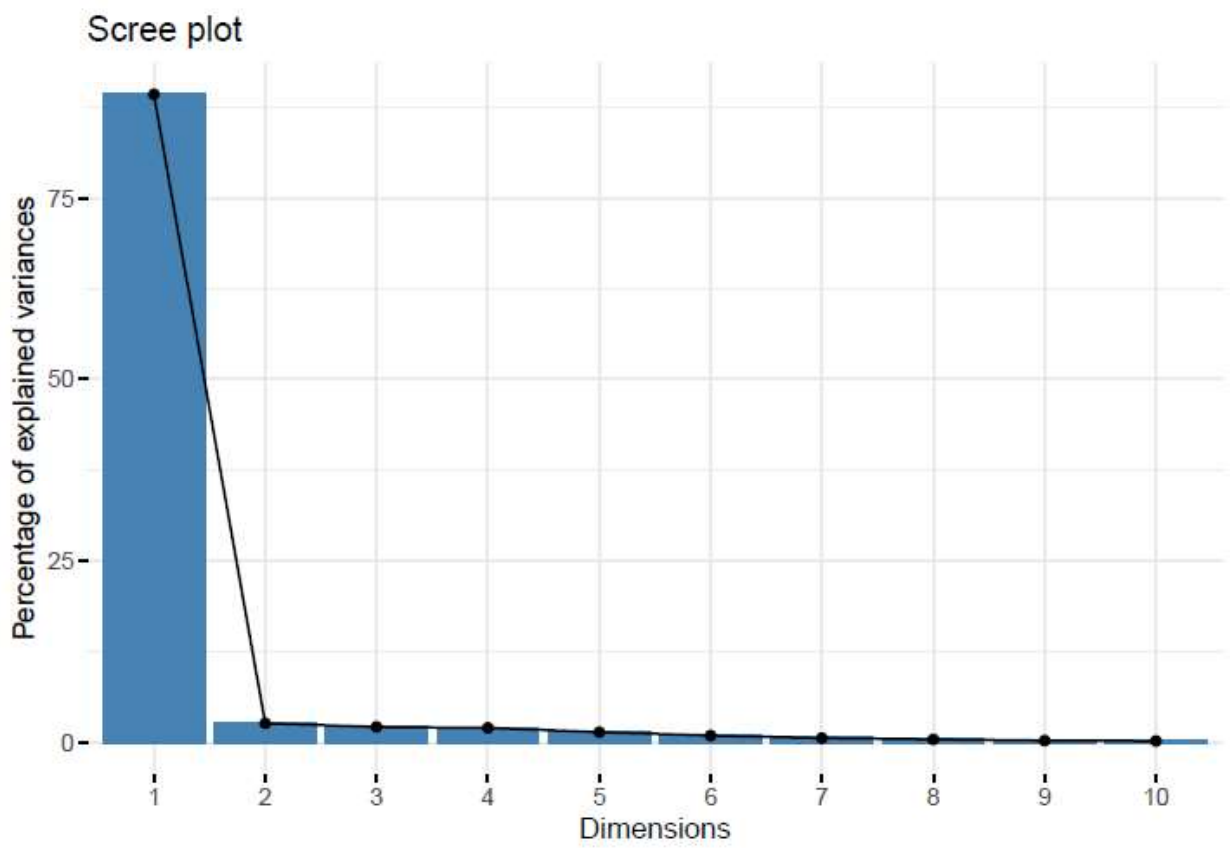
```
PCA = prcomp(Data_1)

names(PCA)
```

```
## [1] "sdev"     "rotation" "center"   "scale"    "x"
```

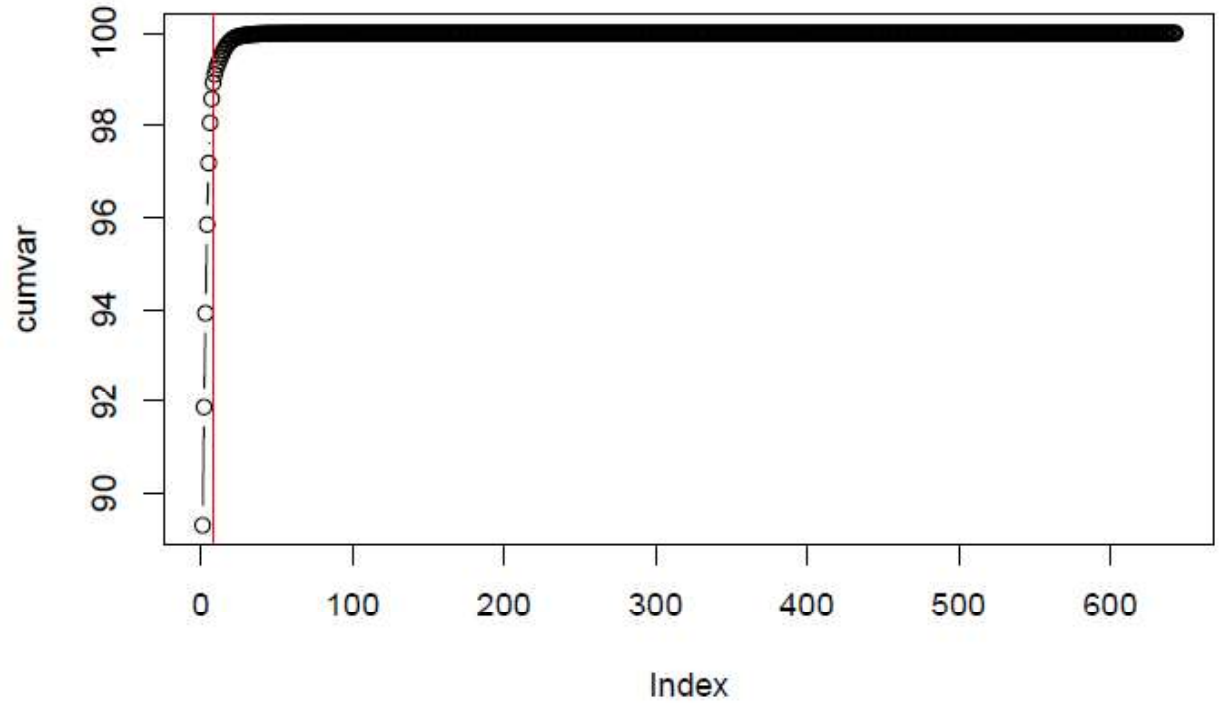
```
# Eigen Values:

Eig = (PCA$sdev)^2

# Variances in percentage

Variance = Eig * 100/sum(Eig)

# Cumulative variances
cumvar = cumsum(Variance)

New_Data = data.frame(eigenvalues = Eig, variance = Variance, cumulative_variance = cumvar)
head(New_Data)
```

```
##    eigenvalues   variance cumulative_variance
## 1 3.775216e+12 89.3047516            89.30475
## 2 1.080458e+11  2.5558807            91.86063
## 3 8.701280e+10  2.0583341            93.91897
## 4 8.133366e+10  1.9239910            95.84296
## 5 5.654686e+10  1.3376460            97.18060
## 6 3.692757e+10  0.8735412            98.05414
```

```
fviz_screeplot(PCA, ncp = 10)
```




```
plot(cumvar, type = "b")
abline(h = 0.975, col = "red", v = 8)
```



Building a random forest model on the PCA data:

```
# Using training and test split:
set.seed(1234)
PCA_Data = data.frame(target = Data$target, PCA$x)
Ind = sample(2, nrow(PCA_Data), replace = T, prob = c(0.6, 0.4))

Train_data = PCA_Data[Ind == 1, ]
Test_data = PCA_Data[Ind == 2, ]

Train_data$target = as.factor(Train_data$target)
Test_data$target = as.factor(Test_data$target)

Model_RF = randomForest(target ~ ., data = Train_data, mtry = 5, ntree = 100,
    importance = T, replace = T)

Pred = predict(Model_RF, newdata = Train_data)

confusionMatrix(Pred, Train_data$target)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction     0     1     2     3     4     5
##          0 58190     0     0     0     0     0
##          1     0   956     0     0     0     0
##          2     0     0     9     0     0     0
##          3     0     0     0    19     0     0
##          4     0     0     0     0   105     0
##          5     0     0     0     0     0   589
##
## Overall Statistics
##
##                Accuracy : 1
##                  95% CI : (0.9999, 1)
##     No Information Rate : 0.972
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 1
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: 0 Class: 1  Class: 2  Class: 3 Class: 4
## Sensitivity             1.000  1.00000 1.0000000 1.0000000 1.000000
## Specificity             1.000  1.00000 1.0000000 1.0000000 1.000000
## Pos Pred Value          1.000  1.00000 1.0000000 1.0000000 1.000000
## Neg Pred Value          1.000  1.00000 1.0000000 1.0000000 1.000000
## Prevalence              0.972  0.01597 0.0001503 0.0003174 0.001754
## Detection Rate          0.972  0.01597 0.0001503 0.0003174 0.001754
## Detection Prevalence    0.972  0.01597 0.0001503 0.0003174 0.001754
```

```
## Balanced Accuracy       1.000  1.00000 1.0000000 1.0000000 1.000000
##                      Class: 5
## Sensitivity          1.000000
## Specificity          1.000000
## Pos Pred Value       1.000000
## Neg Pred Value       1.000000
## Prevalence           0.009838
## Detection Rate       0.009838
## Detection Prevalence 0.009838
## Balanced Accuracy    1.000000
```

```
Pred_Test = predict(Model_RF, newdata = Test_data)

Confusion_Matrix = confusionMatrix(Pred_Test, Test_data$target)

Accuracy_RF = Confusion_Matrix$overall[1]

Accuracy_RF
```

```
##  Accuracy
## 0.9928756
```

The accuracy is ~99% for test data and 100% in train data after applying PCA.