Prediction of Air Pressure System (APS) Failures at Scania Trucks

Venkata Narasimha Durga Rao Pottella

11/21/2019

Executive summary:

The project report addresses the application of machine learning techniques in manufacturing and automobile industry. This project is based on the real time data obtained from sensors of the Scania trucks to predict the failures due to air pressure systems.

The sensor data was given by experts and they didn't mention any attribute names for proprietary reasons. So, we got the data with no attribute names from UCI machine learning repository. The data has 171 attributes lot of missing values, high correlation between the variables. There is also a cost matrix for misclassification and the goal is to reduce the cost.

We used unsupervised ML techniques like PCA to reduce the dimensionality and correlation between the variables. Then, we trained the data using 5 different advanced machine learning techniques like SVM with linear, radial and polynomial kernel, Random forest, XGBoost algorithm and calculated the cost based on the confusion matrix generated by the models. Out of all, XGBoost algorithm minimized the cost much lesser than the rest of the algorithms.

We can use this ML methods in the real time to produce the same results that we got here.

Objective:

The main purpose of this project is to classify the failures of the Scania trucks is whether by the components related to APS (Air pressure system) or by the other components that are not related to APS. The data has only 2 classes i.e. positive or negative. So, this is a binary classification problem.

There is also misclassification cost associated with this problem.

```
##       pred act_post act_neg
## 1 positive        - Cost_1,
## 2 negative   Cost_2       -
```

Here, Cost_1 refers to the cost that an unnecessary check needs to be done by a mechanic at the workshop and Cost_1 is 10, while Cost_2 refer to the cost of missing a faulty truck, which may cause a breakdown and Cost_2 is 500.

```
Total_cost = Cost_1 x No_Instances + Cost_2 x No_Instances.
```

Here, our goal is to minimize the total cost by reducing the errors especially type 2 errors i.e false negatives.

Description of the Dataset:

The dataset was founded in UCI machine learning repository. The source of the data was from Scania Trucks, Sweden. The dataset consists of data collected from heavy Scania trucks in everyday usage. The system in focus is the Air Pressure system (APS) which generates pressurized air that are utilized in various functions in a truck, such as braking and gear changes.

The dataset consists of 171 attributes for which names of them have been anonymized for proprietary reasons. Out of those 171, one of the attributes is Class which consists of two variables i.e. positive or negative. The positive class consists of component failures for a specific component of the APS system. The negative class consists of trucks with failures for components not related to the APS.

The data obtained from UCI has two files, one is train and the other one is test. The train file consists of 60000 observations with 171 attributes. 70 of these attributes belong to 7 histograms with ten bins each. Out of the 60000 observations, 59000 belongs to negative class and 1000 belongs to positive class. Here, the ratio of negative to positive class is 59:1. This tells us that the negative class outnumbers positive class by a large proportion and the data also exhibits an unequal distribution between its classes. So, the dataset is imbalanced.

Loading the dataset:

```
train = read.csv("aps_failure_training_set.csv", header = T, skip = 20, na.strings = 'na')
test = read.csv("aps_failure_test_set.csv", header = T, skip = 20, na.strings = 'na')
dim(train) # The size of train dataset
```

```
## [1] 60000   171
```

```
dim(test) # The size of test dataset
```

```
## [1] 16000   171
```

Now, let's look at the 171 attributes names. These attributes doesn't have any proper attrbute names since the data was extracted from sensors. The below are the names of 171 attributes.

```
##   [1] "class"  "aa_000" "ab_000" "ac_000" "ad_000" "ae_000" "af_000"
##   [8] "ag_000" "ag_001" "ag_002" "ag_003" "ag_004" "ag_005" "ag_006"
##  [15] "ag_007" "ag_008" "ag_009" "ah_000" "ai_000" "aj_000" "ak_000"
##  [22] "al_000" "am_0"   "an_000" "ao_000" "ap_000" "aq_000" "ar_000"
##  [29] "as_000" "at_000" "au_000" "av_000" "ax_000" "ay_000" "ay_001"
##  [36] "ay_002" "ay_003" "ay_004" "ay_005" "ay_006" "ay_007" "ay_008"
##  [43] "ay_009" "az_000" "az_001" "az_002" "az_003" "az_004" "az_005"
##  [50] "az_006" "az_007" "az_008" "az_009" "ba_000" "ba_001" "ba_002"
##  [57] "ba_003" "ba_004" "ba_005" "ba_006" "ba_007" "ba_008" "ba_009"
##  [64] "bb_000" "bc_000" "bd_000" "be_000" "bf_000" "bg_000" "bh_000"
##  [71] "bi_000" "bj_000" "bk_000" "bl_000" "bm_000" "bn_000" "bo_000"
##  [78] "bp_000" "bq_000" "br_000" "bs_000" "bt_000" "bu_000" "bv_000"
##  [85] "bx_000" "by_000" "bz_000" "ca_000" "cb_000" "cc_000" "cd_000"
##  [92] "ce_000" "cf_000" "cg_000" "ch_000" "ci_000" "cj_000" "ck_000"
##  [99] "cl_000" "cm_000" "cn_000" "cn_001" "cn_002" "cn_003" "cn_004"
## [106] "cn_005" "cn_006" "cn_007" "cn_008" "cn_009" "co_000" "cp_000"
```

```
## [113] "cq_000" "cr_000" "cs_000" "cs_001" "cs_002" "cs_003" "cs_004"
## [120] "cs_005" "cs_006" "cs_007" "cs_008" "cs_009" "ct_000" "cu_000"
## [127] "cv_000" "cx_000" "cy_000" "cz_000" "da_000" "db_000" "dc_000"
## [134] "dd_000" "de_000" "df_000" "dg_000" "dh_000" "di_000" "dj_000"
## [141] "dk_000" "dl_000" "dm_000" "dn_000" "do_000" "dp_000" "dq_000"
## [148] "dr_000" "ds_000" "dt_000" "du_000" "dv_000" "dx_000" "dy_000"
## [155] "dz_000" "ea_000" "eb_000" "ec_00"  "ed_000" "ee_000" "ee_001"
## [162] "ee_002" "ee_003" "ee_004" "ee_005" "ee_006" "ee_007" "ee_008"
## [169] "ee_009" "ef_000" "eg_000"
```

Exploratory Data Analysis:

Dealing with histograms and bins:

From the description of data, there are 70 attributes belong to 7 histograms with ten bins each. Let's first identify and deal with those 70 attributes. Adding up all the 10 variables in each bin across the columns and combining all 7 bins into dataframe and inspecting the head of the dataframe.

```
##        b1      b2      b3      b4      b5      b6      b7
## 1 6167850 6167850 6167850 6167850 6167850 6167850 6167850
## 2 2940714 2940714 2940714 2942850 2942850 2942850 2940714
## 3 2560566 2560566 2560566 2560566 2560566 2560566 2560566
## 4    7634    7634    7634    7634    7634    7634    7634
## 5 3946944 3946944 3946944 3946944 3946944 3946944 3946944
## 6 2663042 2663042 2663042 2663042 2663042 2663042 2663042
```

Looks like all the 7 bins have same information. So, now replacing all the bins with single bin in the main dataset so that we can reduce the attributes which have redundant information.

Now, the 171 attributes have reduced to 102 attributes.

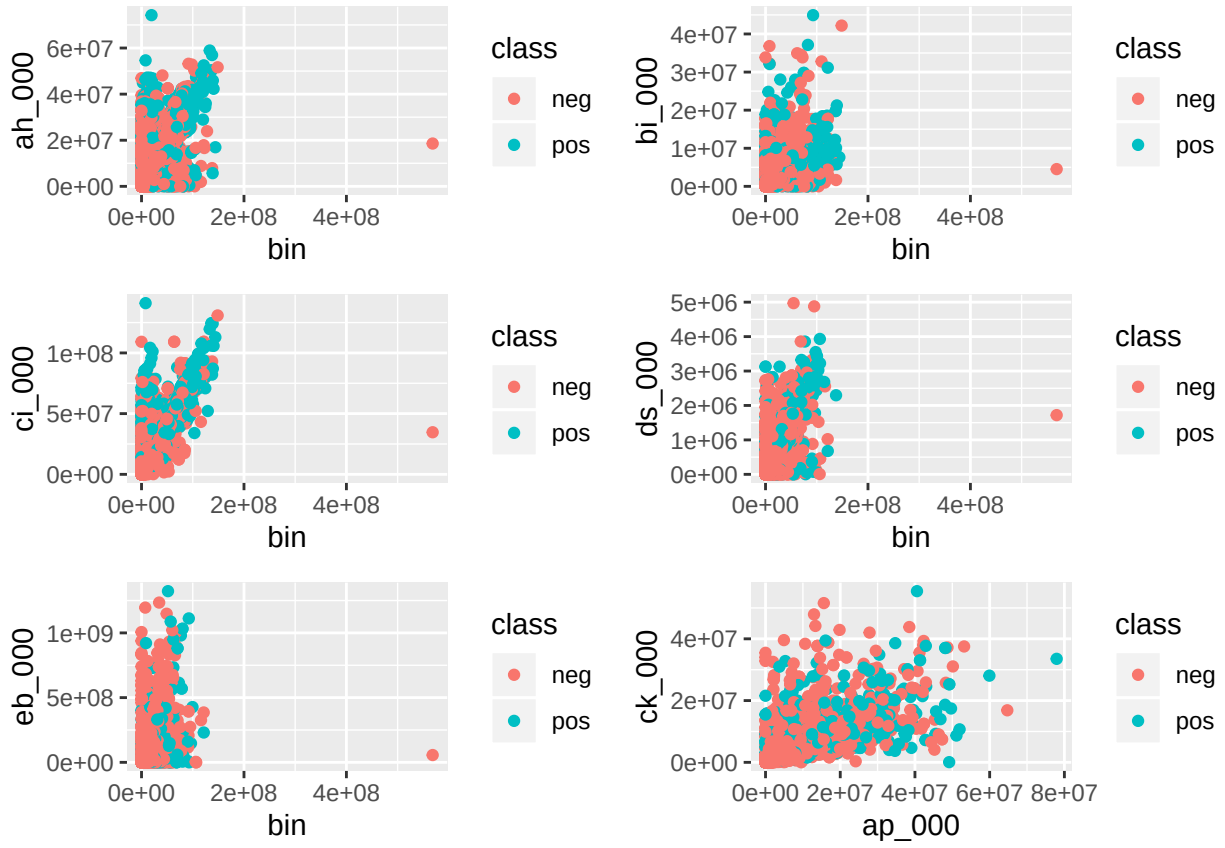
```
## [1] 60000   102
```

The attribute "cd_000" has same values i.e 1209600 and there are some missing values. Since, there is no variation in this attribute, we are removing this attribute from the dataset. The below is the summary of this:

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.    NA's
## 1209600 1209600 1209600 1209600 1209600 1209600     676
```

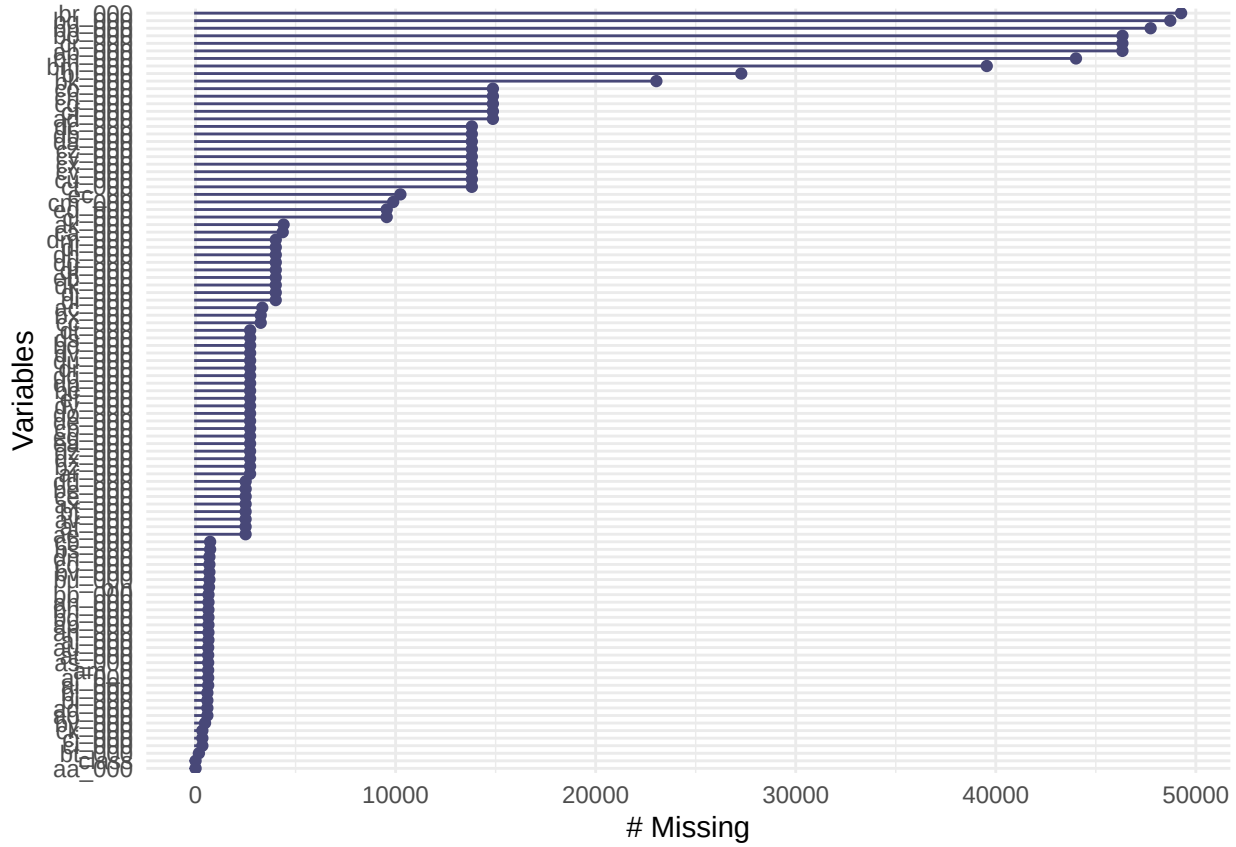
Plotting graphs:

As the attributes doesn't have any names, we cannot infer anything from graphs. But, let's see how the class variable is distributed w.r.t various attributes.



Dealing with missing values:

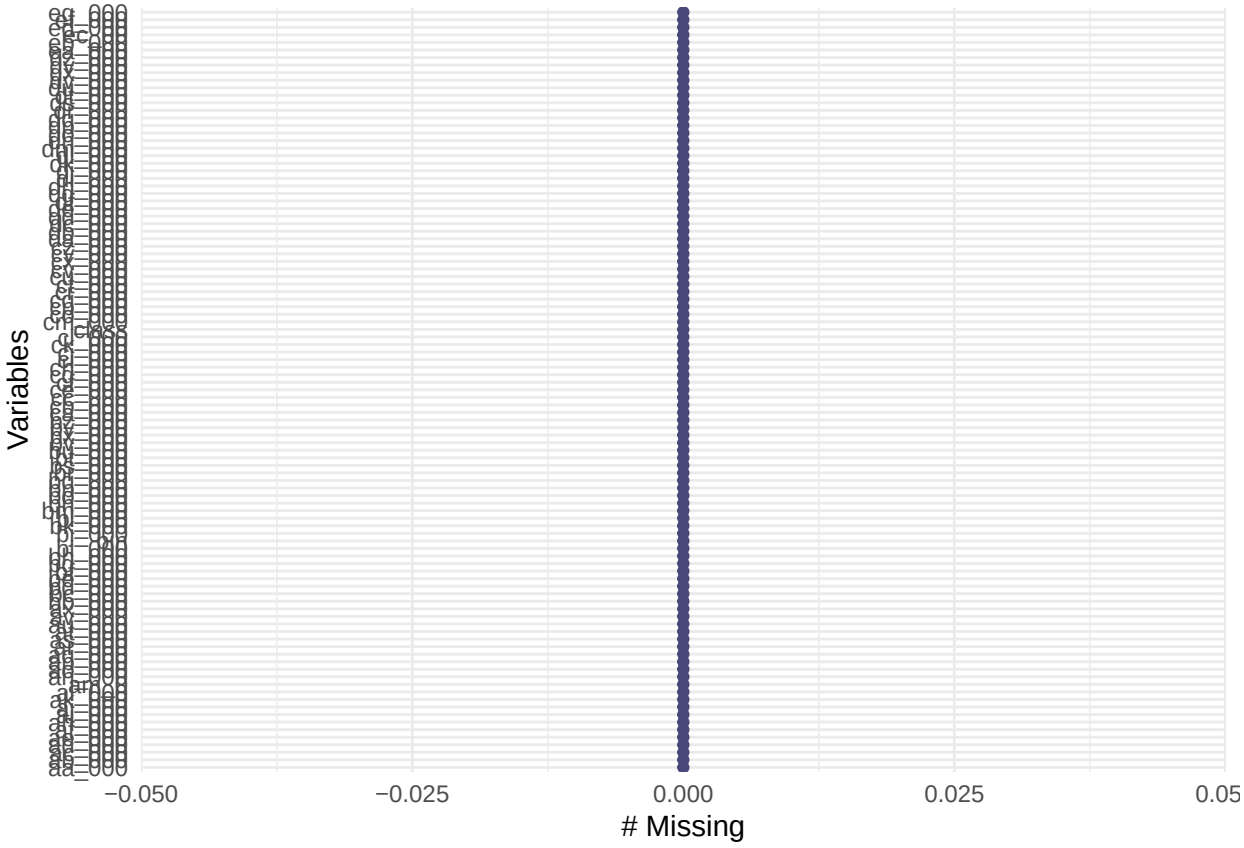
The data has lot of missing values of 0.2% to upto 82%. we choose to impute them with median of the corresponding variables as the data is from sensors and doesn't have any attribute names. The below plot depicts the missing values of each variables.



This shows clearly which attributes have how many missing values.As, we can see the attributes “br_000”, “bq_000”, “bp_000”, “bo_000”, “bn_000” have more that 70% of the missing values. Out of 101 attributes, 99 attributes have missing values.

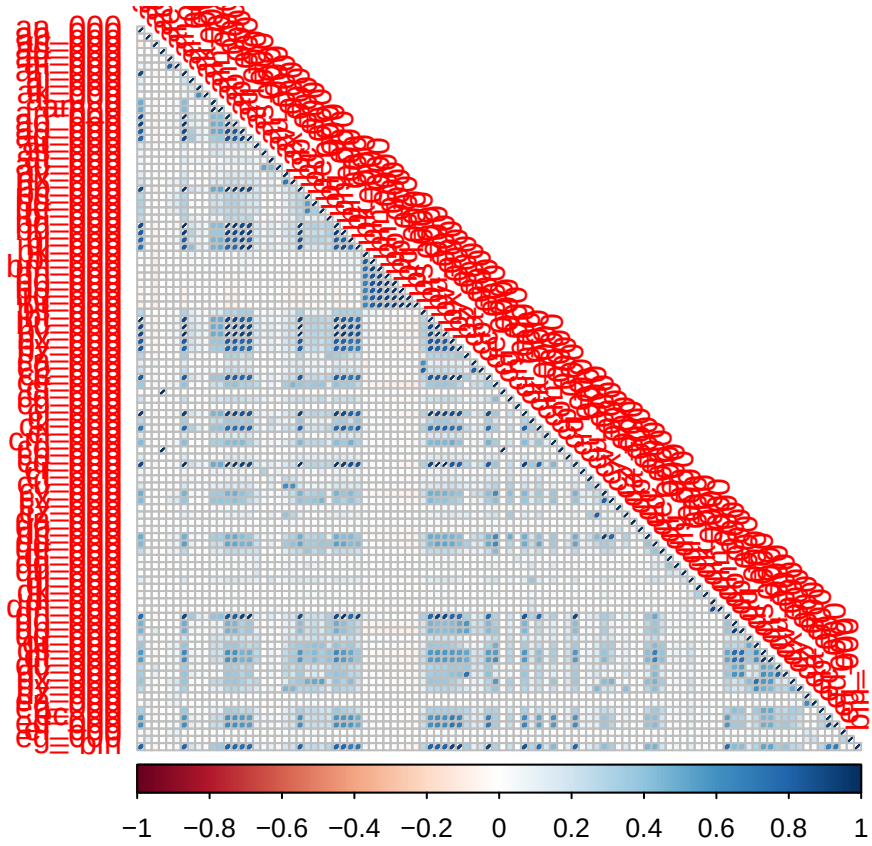
```
##  class aa_000 ab_000 ac_000 ad_000 ae_000 af_000 ah_000 ai_000 aj_000
##      0      0  46329   3335  14861   2500   2500    645    629    629
## ak_000 al_000   am_0 an_000 ao_000 ap_000 aq_000 ar_000 as_000 at_000
##   4400    642    629    642    589    642    589   2723    629    629
## au_000 av_000 ax_000 bb_000 bc_000 bd_000 be_000 bf_000 bg_000 bh_000
##    629   2500   2501    645   2725   2727   2503   2500    642    642
## bi_000 bj_000 bk_000 bl_000 bm_000 bn_000 bo_000 bp_000 bq_000 br_000
##    589    589  23034  27277  39549  44009  46333  47740  48722  49264
## bs_000 bt_000 bu_000 bv_000 bx_000 by_000 bz_000 ca_000 cb_000 cc_000
##    726    167    691    691   3257    473   2723   4356    726   3255
## ce_000 cf_000 cg_000 ch_000 ci_000 cj_000 ck_000 cl_000 cm_000 co_000
##   2502  14861  14861  14861    338    338    338   9553   9877  14861
## cp_000 cq_000 cr_000 ct_000 cu_000 cv_000 cx_000 cy_000 cz_000 da_000
##   2724    691  46329  13808  13808  13808  13808  13808  13808  13808
## db_000 dc_000 dd_000 de_000 df_000 dg_000 dh_000 di_000 dj_000 dk_000
##  13808  13808   2503   2724   4008   4008   4008   4006   4007   4007
## dl_000 dm_000 dn_000 do_000 dp_000 dq_000 dr_000 ds_000 dt_000 du_000
##   4008   4009    691   2724   2726   2726   2726   2727   2727   2726
## dv_000 dx_000 dy_000 dz_000 ea_000 eb_000  ec_00 ed_000 ef_000 eg_000
##   2726   2723   2724   2723   2723   4007  10239   9553   2724   2723
##    bin
##    671
```

Now, After imputing the missing the values with median of the respective columns, there are no missing values in the dataset.



Correlation Analysis:

Let's look at the correlation between the attributes. The below plot depicts the correlation between the variables. From the plot, we can see that there is either zero or positive correlation between the variables.



Among all the attributes, let us see the most correlated attributes i.e the correlation above 0.75.

```
##  [1] "af_000" "ah_000" "am_0"   "an_000" "ao_000" "bb_000" "bg_000"
##  [8] "bh_000" "bm_000" "bn_000" "bo_000" "bp_000" "bq_000" "br_000"
## [15] "bt_000" "bu_000" "bv_000" "bx_000" "cb_000" "cc_000" "cf_000"
## [22] "ci_000" "cq_000" "dc_000" "dm_000" "dn_000" "dp_000" "ds_000"
## [29] "dv_000" "ed_000" "bin"    "aa_000" "ap_000" "aq_000" "bk_000"
## [36] "bs_000" "bj_000" "ad_000" "cv_000" "cu_000"
```

Data Preprocessing:

Principal Component Analysis:

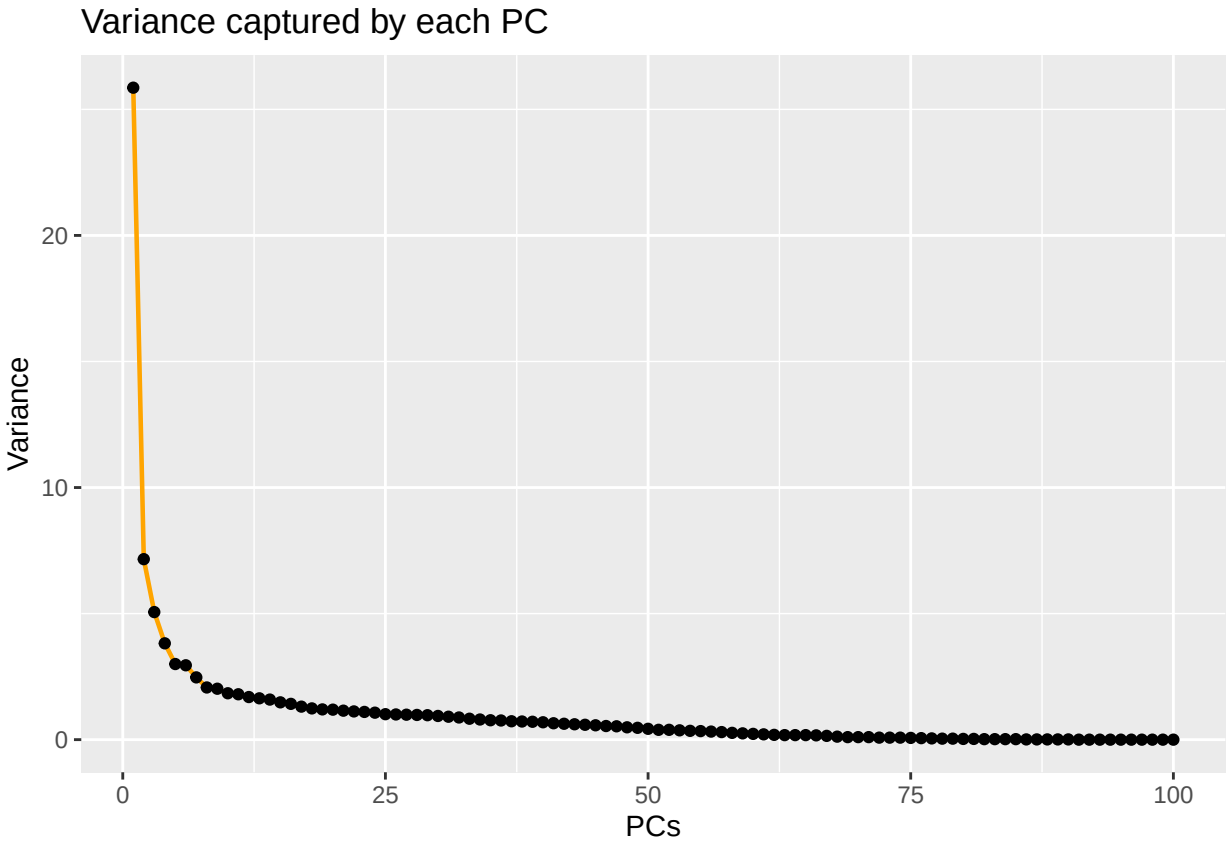
We have lot of attributes i.e 101 and lot of correlated variables. So, the best feature engineering is Prinicpal Component Analysis(PCA).

```
## Importance of components:
##                             PC1     PC2     PC3     PC4   PC5     PC6
## Standard deviation     5.0853 2.67541 2.25029 1.95480 1.732 1.71712
## Proportion of Variance 0.2586 0.07158 0.05064 0.03821 0.030 0.02948
## Cumulative Proportion  0.2586 0.33018 0.38082 0.41903 0.449 0.47852
##                             PC7     PC8    PC9    PC10    PC11    PC12
## Standard deviation     1.57122 1.43745 1.4211 1.35478 1.34129 1.29840
## Proportion of Variance 0.02469 0.02066 0.0202 0.01835 0.01799 0.01686
```

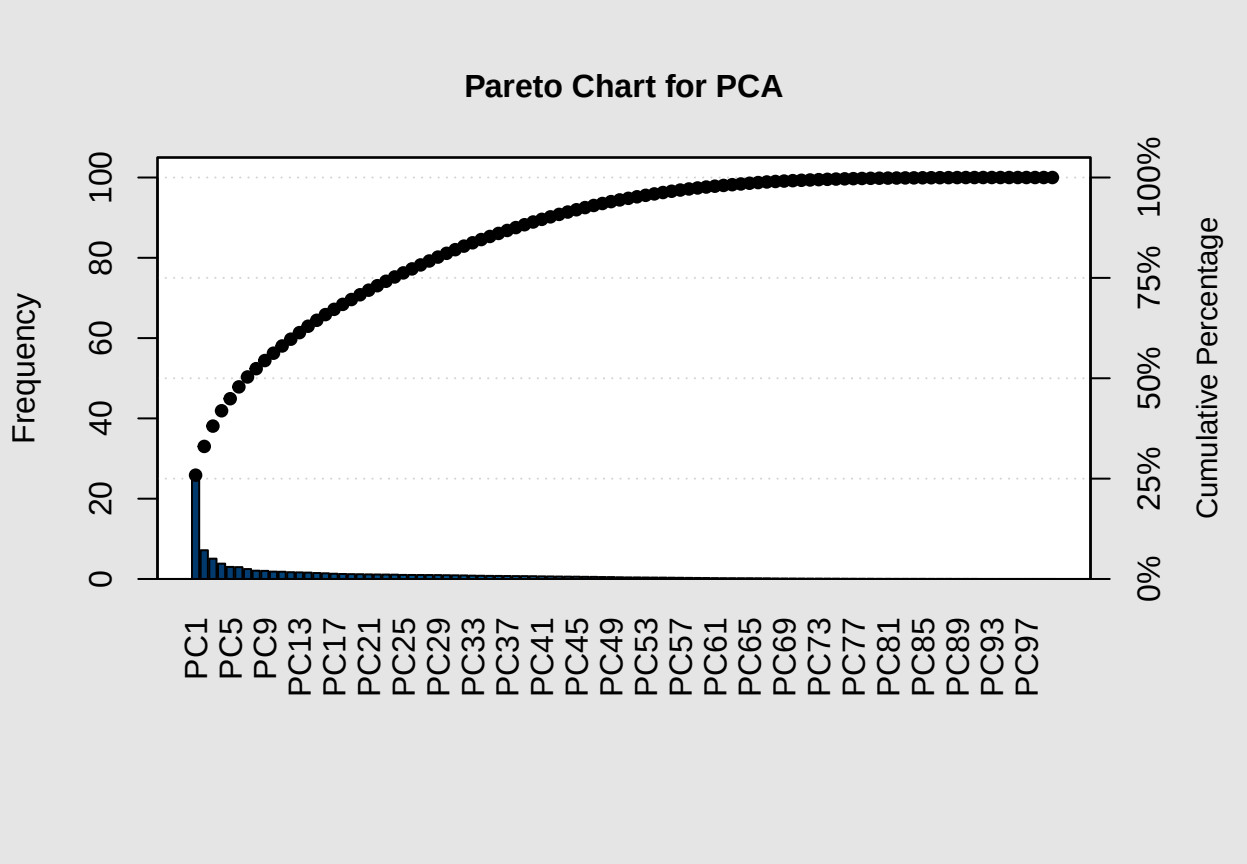
```
## Cumulative Proportion  0.50321 0.52387 0.5441 0.56242 0.58041 0.59727
##                           PC13    PC14    PC15    PC16    PC17    PC18
## Standard deviation     1.28000 1.26025 1.21760 1.19018 1.14498 1.11400
## Proportion of Variance 0.01638 0.01588 0.01483 0.01417 0.01311 0.01241
## Cumulative Proportion  0.61365 0.62953 0.64436 0.65852 0.67163 0.68404
##                           PC19    PC20    PC21    PC22   PC23    PC24
## Standard deviation     1.09609 1.08872 1.07008 1.05722 1.0487 1.03661
## Proportion of Variance 0.01201 0.01185 0.01145 0.01118 0.0110 0.01075
## Cumulative Proportion  0.69606 0.70791 0.71936 0.73054 0.7415 0.75228
##                           PC25    PC26    PC27    PC28    PC29    PC30
## Standard deviation     1.00525 0.99945 0.99360 0.99187 0.98629 0.96746
## Proportion of Variance 0.01011 0.00999 0.00987 0.00984 0.00973 0.00936
## Cumulative Proportion  0.76239 0.77238 0.78225 0.79209 0.80181 0.81117
##                           PC31    PC32    PC33    PC34   PC35    PC36
## Standard deviation     0.95271 0.93881 0.91260 0.89536 0.8774 0.87100
## Proportion of Variance 0.00908 0.00881 0.00833 0.00802 0.0077 0.00759
## Cumulative Proportion  0.82025 0.82906 0.83739 0.84541 0.8531 0.86069
##                           PC37   PC38    PC39    PC40    PC41   PC42
## Standard deviation     0.85729 0.8483 0.84197 0.82953 0.80808 0.7935
## Proportion of Variance 0.00735 0.0072 0.00709 0.00688 0.00653 0.0063
## Cumulative Proportion  0.86804 0.8752 0.88233 0.88921 0.89574 0.9020
##                           PC43    PC44    PC45    PC46    PC47    PC48
## Standard deviation     0.77887 0.76624 0.75554 0.73388 0.72498 0.69781
## Proportion of Variance 0.00607 0.00587 0.00571 0.00539 0.00526 0.00487
## Cumulative Proportion  0.90810 0.91397 0.91968 0.92507 0.93032 0.93519
##                           PC49    PC50    PC51    PC52    PC53    PC54
## Standard deviation     0.68759 0.65220 0.62658 0.62153 0.60525 0.59465
## Proportion of Variance 0.00473 0.00425 0.00393 0.00386 0.00366 0.00354
## Cumulative Proportion  0.93992 0.94417 0.94810 0.95196 0.95563 0.95916
##                           PC55    PC56    PC57   PC58    PC59    PC60
## Standard deviation     0.58449 0.56444 0.54607 0.5199 0.50183 0.48048
## Proportion of Variance 0.00342 0.00319 0.00298 0.0027 0.00252 0.00231
## Cumulative Proportion  0.96258 0.96577 0.96875 0.9715 0.97397 0.97628
##                           PC61    PC62    PC63    PC64    PC65    PC66
## Standard deviation     0.45444 0.43671 0.42999 0.42682 0.42203 0.40860
## Proportion of Variance 0.00207 0.00191 0.00185 0.00182 0.00178 0.00167
## Cumulative Proportion  0.97834 0.98025 0.98210 0.98392 0.98570 0.98737
##                           PC67    PC68    PC69    PC70    PC71   PC72
## Standard deviation     0.39207 0.34394 0.32246 0.32020 0.30822 0.2820
## Proportion of Variance 0.00154 0.00118 0.00104 0.00103 0.00095 0.0008
## Cumulative Proportion  0.98891 0.99009 0.99113 0.99216 0.99311 0.9939
##                           PC73    PC74    PC75    PC76    PC77    PC78
## Standard deviation     0.28083 0.27547 0.26115 0.23470 0.21833 0.20727
## Proportion of Variance 0.00079 0.00076 0.00068 0.00055 0.00048 0.00043
## Cumulative Proportion  0.99469 0.99545 0.99613 0.99668 0.99716 0.99759
##                           PC79    PC80    PC81    PC82   PC83    PC84
## Standard deviation     0.20374 0.16298 0.16192 0.14952 0.1425 0.13810
## Proportion of Variance 0.00042 0.00027 0.00026 0.00022 0.0002 0.00019
## Cumulative Proportion  0.99800 0.99827 0.99853 0.99875 0.9990 0.99915
##                           PC85    PC86    PC87    PC88    PC89    PC90
## Standard deviation     0.13508 0.12181 0.11673 0.10980 0.09635 0.08572
## Proportion of Variance 0.00018 0.00015 0.00014 0.00012 0.00009 0.00007
## Cumulative Proportion  0.99933 0.99948 0.99962 0.99974 0.99983 0.99990
##                           PC91    PC92    PC93    PC94    PC95     PC96
```

```
## Standard deviation     0.06208 0.05676 0.03533 0.03433 0.01571 0.005336
## Proportion of Variance 0.00004 0.00003 0.00001 0.00001 0.00000 0.000000
## Cumulative Proportion  0.99994 0.99997 0.99999 1.00000 1.00000 1.000000
##                            PC97      PC98      PC99     PC100
## Standard deviation     5.383e-05 2.308e-05 3.521e-06 1.097e-06
## Proportion of Variance 0.000e+00 0.000e+00 0.000e+00 0.000e+00
## Cumulative Proportion  1.000e+00 1.000e+00 1.000e+00 1.000e+00
```

The summary describes the importance of the PCs. The first row describe the standard deviation associated with each PC. The second row shows the proportion of the variance in the data explained by each component while the third row describe the cumulative proportion of explained variance. We can see there that the 42 PCs accounts for more than 90% of the variance of the data and 52 PCs accounts for 95% of the variance of the data.



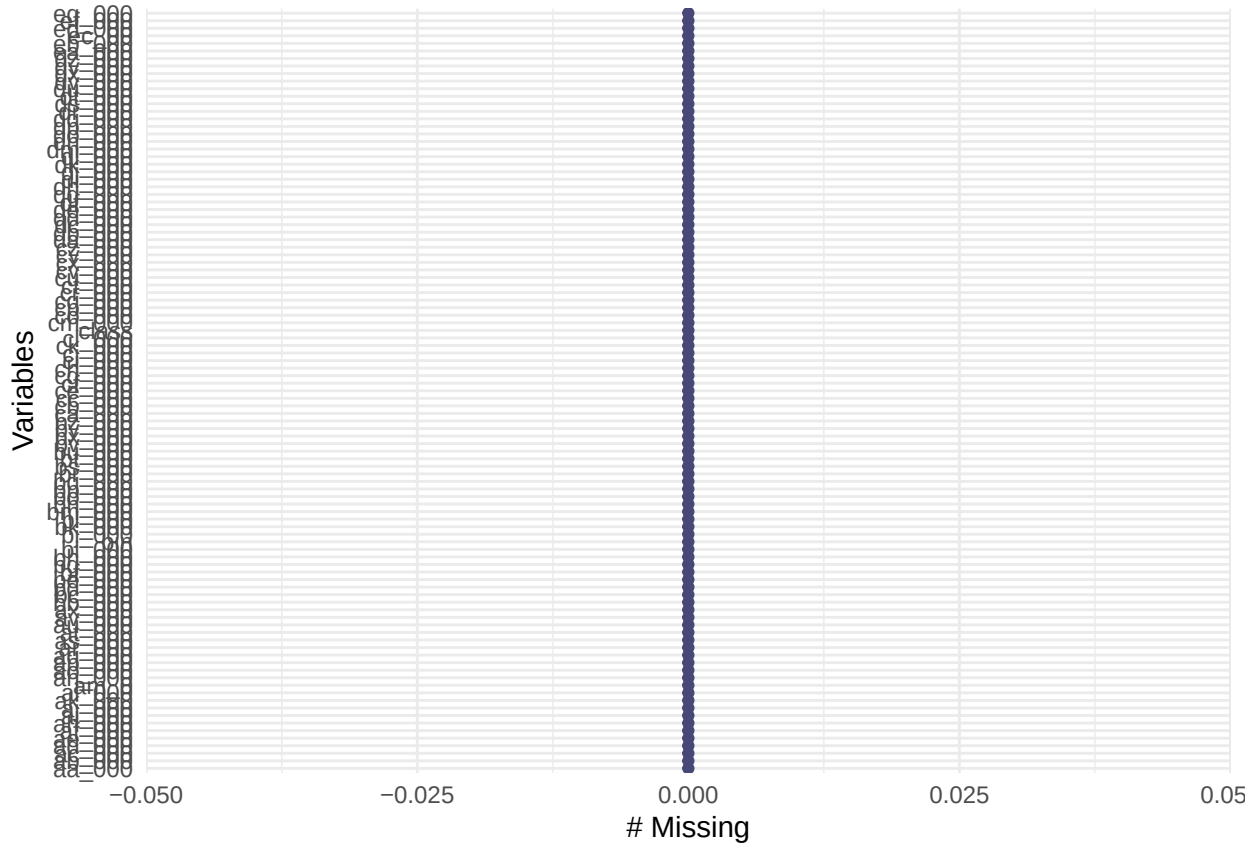
The above plot shows the variance captured by each PC. The plot depicts decrease in variation captured when we move to the extreme end as in PCA, the initial PCs capture lot of variation.



Since, 95% of the variation is capured by 52 PCs, we are using only 52 PCs as the independent variables for further analysis.

Preprocessing the test dataset:

Preforming all the above operations on test dataset also for evaluating and calculating the cost matrices. We imputed the missing values with median for test dataset also.



Using PCA of training set, predicting the PCA values for test dataset for evaluating in conjuction with train dataset.

Dealing wth unbalanced data:

```
##
##   neg   pos
## 59000  1000
```

Since the data is unbalanced, working on this will lead to reduction in accuracy and the models will get biased towards majority class. So, to balance the dataset, we considered the below 3 methods. • Under Sampling • Over Sampling • Synthetic minority oversampling technique (SMOTE) • Both Under and Over Sampling The data is modified using these 4 methods and can be tested on logisitc regression to test the best technique for further ananlysis.

Over Sampling:

```
##
##    neg    pos
##  59000 137703
```

```
## Confusion Matrix and Statistics
##
##           Reference
```

```
## Prediction    neg    pos
##        neg  49589   5692
##        pos   9411 132011
##
##                Accuracy : 0.9232
##                  95% CI : (0.922, 0.9244)
##     No Information Rate : 0.7001
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.8138
##
##  Mcnemar's Test P-Value : < 2.2e-16
##
##             Sensitivity : 0.8405
##             Specificity : 0.9587
##          Pos Pred Value : 0.8970
##          Neg Pred Value : 0.9335
##              Prevalence : 0.2999
##          Detection Rate : 0.2521
##    Detection Prevalence : 0.2810
##       Balanced Accuracy : 0.8996
##
##        'Positive' Class : neg
##
```

Here the accuracy is 0.9232 or 92.32%.

Under Sampling:

```
##
##  neg  pos
##  961 1000
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction neg pos
##        neg 807  56
##        pos 154 944
##
##                Accuracy : 0.8929
##                  95% CI : (0.8784, 0.9063)
##     No Information Rate : 0.5099
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.7853
##
##  Mcnemar's Test P-Value : 2.177e-11
##
##             Sensitivity : 0.8398
##             Specificity : 0.9440
##          Pos Pred Value : 0.9351
##          Neg Pred Value : 0.8597
```

```
##              Prevalence : 0.4901
##          Detection Rate : 0.4115
##    Detection Prevalence : 0.4401
##       Balanced Accuracy : 0.8919
##
##        'Positive' Class : neg
##
```

Here the accuracy is 0.8929 or 89.29%.

SMOTE:

```
##
##   neg   pos
## 40000 20000
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   neg   pos
##        neg 39041  2279
##        pos   959 17721
##
##                Accuracy : 0.946
##                  95% CI : (0.9442, 0.9478)
##     No Information Rate : 0.6667
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.8765
##
##  Mcnemar's Test P-Value : < 2.2e-16
##
##             Sensitivity : 0.9760
##             Specificity : 0.8861
##          Pos Pred Value : 0.9448
##          Neg Pred Value : 0.9487
##              Prevalence : 0.6667
##          Detection Rate : 0.6507
##    Detection Prevalence : 0.6887
##       Balanced Accuracy : 0.9310
##
##        'Positive' Class : neg
##
```

Here the accuracy is 0.946 or 94.6%.

Both(Under and Over Sampling):

```
##
##   neg   pos
## 29951 30049
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   neg   pos
##        neg 28325  2833
##        pos  1626 27216
##
##                Accuracy : 0.9257
##                  95% CI : (0.9236, 0.9278)
##     No Information Rate : 0.5008
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.8514
##
##  Mcnemar's Test P-Value : < 2.2e-16
##
##             Sensitivity : 0.9457
##             Specificity : 0.9057
##          Pos Pred Value : 0.9091
##          Neg Pred Value : 0.9436
##              Prevalence : 0.4992
##          Detection Rate : 0.4721
##    Detection Prevalence : 0.5193
##       Balanced Accuracy : 0.9257
##
##        'Positive' Class : neg
##
```

Here the accuracy is 0.9257 or 92.57%

So, the best technique for this unbalanced dataset is SMOTE technique which generates the neg to pos ration of 2:1. we will use SMOTE technique for further analysis.

```
##
##   neg   pos
## 40000 20000
```

Data Modelling:

Here, our goal is to calculate the cost associated with missclassification errors and find the best best model with minimum cost. So for calculating the cost, we are generating confusion matrix for each model and comparing them at the end.

SVM with Linear Kernel:

```
##
## Call:
## svm(formula = class ~ ., data = pca.best.smote, kernel = "linear",
##     cost = 0.01)
##
##
## Parameters:
```

```
##    SVM-Type:  C-classification
##  SVM-Kernel:  linear
##        cost:  0.01
##
## Number of Support Vectors:  9069
##
##  ( 4531 4538 )
##
##
## Number of Classes:  2
##
## Levels:
##  neg pos
```

From summary above, we found out that there are total of 9069 support vectors with 4531 for 'neg' class and 4538 for 'pos' class.

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   neg   pos
##        neg 15217    32
##        pos   408   343
##
##                Accuracy : 0.9725
##                  95% CI : (0.9698, 0.975)
##     No Information Rate : 0.9766
##     P-Value [Acc > NIR] : 0.9996
##
##                   Kappa : 0.5966
##
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 0.9739
##             Specificity : 0.9147
##          Pos Pred Value : 0.9979
##          Neg Pred Value : 0.4567
##              Prevalence : 0.9766
##          Detection Rate : 0.9511
##    Detection Prevalence : 0.9531
##       Balanced Accuracy : 0.9443
##
##        'Positive' Class : neg
##
```

The accuracy for test dataset for SVM model with linear kernel with cost=0.01 is 0.9725 or 97.25%. So, the error rate is 0.0275 or 2.75% which is less than the naive classification accuracy of 97.65%. The cost obtained from this model is

Total Cost = 408 x 10 + 32 x 500 = 20,080

SVM with Radial Kernel:

```
##
```

```
## Call:
## svm(formula = class ~ ., data = pca.best.smote, kernel = "radial",
##     gamma = 0.1, cost = 0.1)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  radial
##        cost:  0.1
##
## Number of Support Vectors:  10168
##
##  ( 4258 5910 )
##
##
## Number of Classes:  2
##
## Levels:
##  neg pos
```

From summary above, we found out that there are total of 10168 support vectors with 4258 for 'neg' class and 5910 for 'pos' class.

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   neg   pos
##        neg 14999    19
##        pos   626   356
##
##                Accuracy : 0.9597
##                  95% CI : (0.9565, 0.9627)
##     No Information Rate : 0.9766
##     P-Value [Acc > NIR] : 1
##
##                   Kappa : 0.508
##
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 0.9599
##             Specificity : 0.9493
##          Pos Pred Value : 0.9987
##          Neg Pred Value : 0.3625
##              Prevalence : 0.9766
##          Detection Rate : 0.9374
##    Detection Prevalence : 0.9386
##       Balanced Accuracy : 0.9546
##
##        'Positive' Class : neg
##
```

The accuracy for test dataset for SVM model with radial kernel with cost=0.1 is 0.9597 or 95.97%. So, the error rate is 0.0403 or 4.03% which is less than the naive classification accuracy of 97.65%. The cost obtained from this model is

Total Cost = 626 x 10 + 19 x 500 = 15,760

SVM with Polynomial Kernel:

```
##
## Call:
## svm(formula = class ~ ., data = pca.best.smote, kernel = "polynomial",
##     cost = 0.01, gamma = 0.1, degree = 2)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  polynomial
##        cost:  0.01
##      degree:  2
##      coef.0:  0
##
## Number of Support Vectors:  12870
##
##  ( 6452 6418 )
##
##
## Number of Classes:  2
##
## Levels:
##  neg pos
```

From summary above, we found out that there are total of 12870 support vectors with 6452 for 'neg' class and 6418 for 'pos' class.

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   neg   pos
##        neg 15400    75
##        pos   225   300
##
##                Accuracy : 0.9812
##                  95% CI : (0.979, 0.9833)
##     No Information Rate : 0.9766
##     P-Value [Acc > NIR] : 2.885e-05
##
##                   Kappa : 0.6573
##
##  Mcnemar's Test P-Value : < 2.2e-16
##
##             Sensitivity : 0.9856
##             Specificity : 0.8000
##          Pos Pred Value : 0.9952
##          Neg Pred Value : 0.5714
##              Prevalence : 0.9766
##          Detection Rate : 0.9625
##    Detection Prevalence : 0.9672
```

```
##       Balanced Accuracy : 0.8928
##
##        'Positive' Class : neg
##
```

The accuracy for test dataset for SVM model with polynomial kernel with cost=0.01 is 0.9812 or 98.12%. So, the error rate is 0.0188 or 1.88% which is higher than the naive classification accuracy of 97.65%. The cost obtained from this model is

Total Cost = 225 x 10 + 75 x 500 = 39,750

Here, the cost is high compared to other SVM models.

Random Forest:

Now, we will create a Random Forest model with default parameters i.e Ntree i.e Number of trees to grow is 500 and number of variables tried at each split i.e mtry is 6 in this case. Error rate is 0.86%.

```
##
## Call:
##  randomForest(formula = class ~ ., data = pca.best.smote, ntree = 500,      mtry = 6, importance = T
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 6
##
##         OOB estimate of  error rate: 0.88%
## Confusion matrix:
##       neg   pos class.error
## neg 39648   352     0.00880
## pos   177 19823     0.00885
```

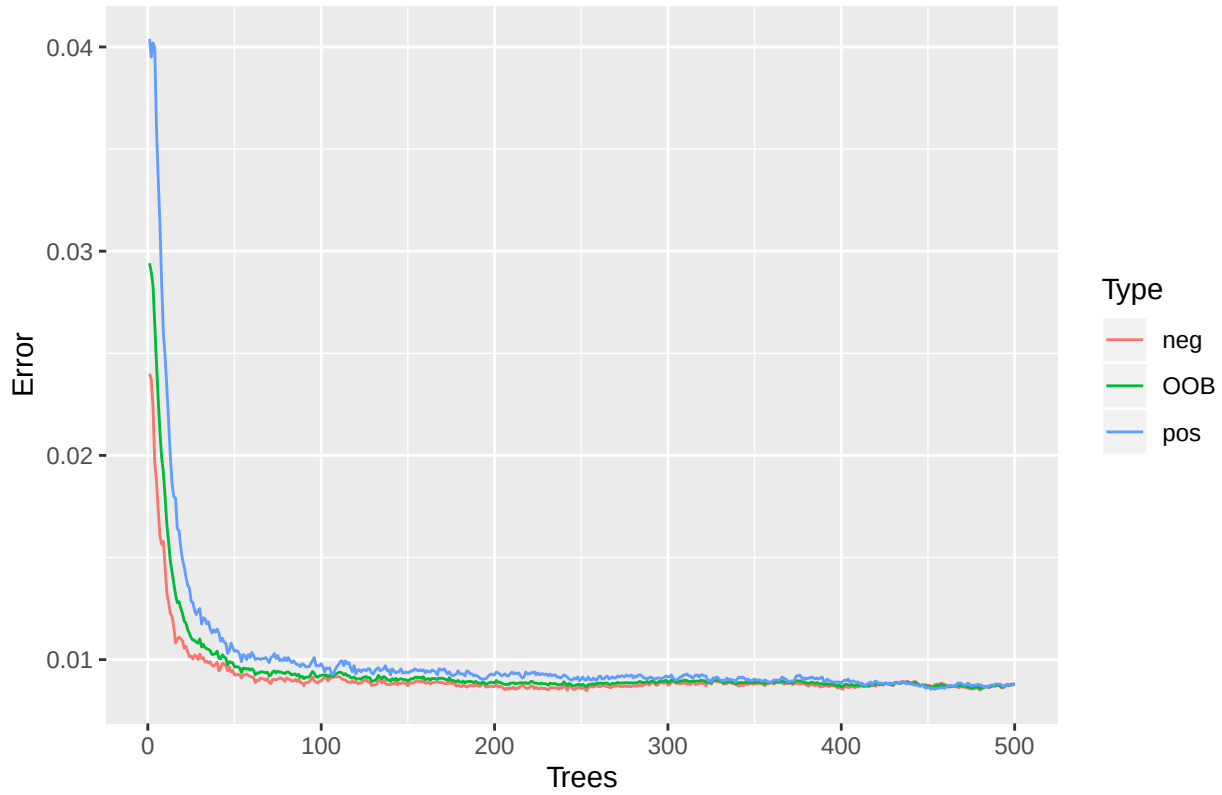
```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   neg   pos
##        neg 15364    41
##        pos   261   334
##
##               Accuracy : 0.9811
##                 95% CI : (0.9789, 0.9832)
##    No Information Rate : 0.9766
##    P-Value [Acc > NIR] : 4.617e-05
##
##                  Kappa : 0.6794
##
##  Mcnemar's Test P-Value : < 2.2e-16
##
##             Sensitivity : 0.9833
##             Specificity : 0.8907
##          Pos Pred Value : 0.9973
##          Neg Pred Value : 0.5613
##              Prevalence : 0.9766
##          Detection Rate : 0.9603
```

```
##    Detection Prevalence : 0.9628
##       Balanced Accuracy : 0.9370
##
##        'Positive' Class : neg
##
```

In case of prediction on test dataset, there is 0.0183 or 1.83% misclassification and 293 data points are misclassified and accuracy is 98.17% which is higher than the naive classification accuracy of 97.65%. The cost obtained from this model is

Total Cost = 249 x 10 + 44 x 500 = 24,490

Seems like SVM outperformed Random forest for this dataset w.r.t cost.



From the above plot, we can see that the error decreases as the number of trees increases. Here the error is high for "pos" class compared to "neg" class since the majority class in the data is "neg" class.

XGBoost Algorithm:

Now, we will train a XGBoost Algorithm with default parameters i.e eta(learning rate) is 0.1, max_depth that the trees will be generated is 6, no of iterations is 100. The objective function used here is mutli:softmax which is multiclassification using softmax objective and returns predicted class labels.The evaluation metrics is mlogloss which is multiclass logloss used in classification.

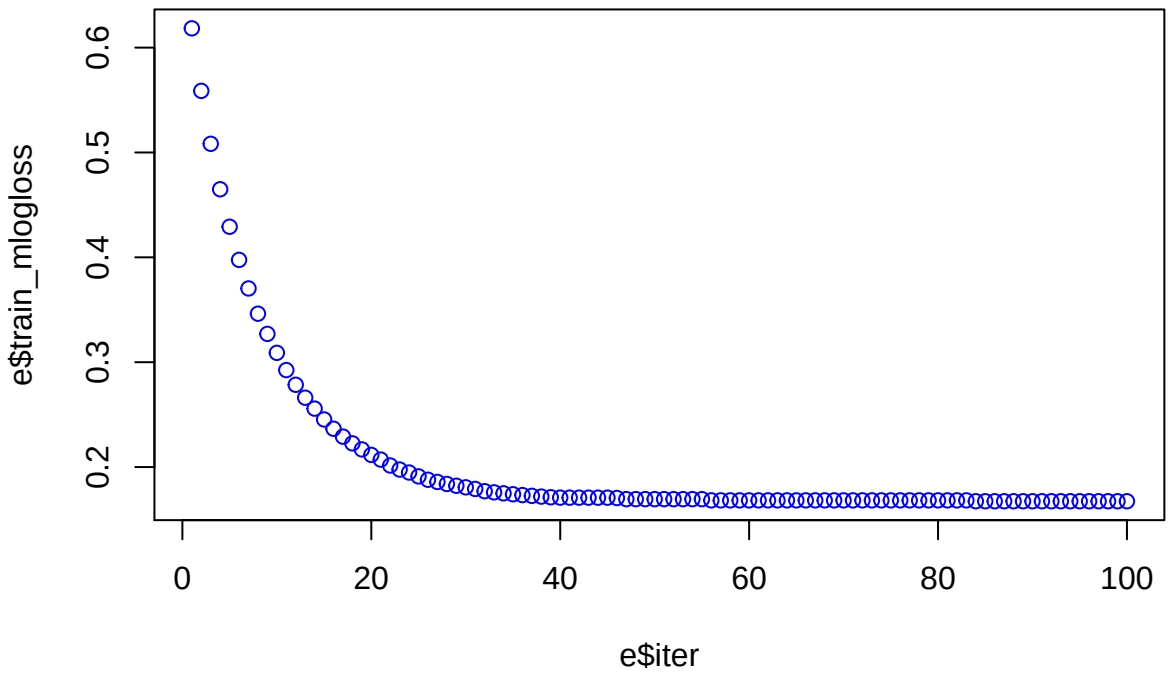
```
#pca.best.smote$class <- as.numeric(as.factor(pca.best.smote$class)) - 1
train_label = pca.best.smote[,'class']
train_label = as.numeric(as.factor(train_label)) - 1
```

```
train_matrix = xgb.DMatrix(data = as.matrix(pca.best.smote[-53]), label=train_label)


test_label = pca.test[,'class']
test_label = as.numeric(as.factor(test_label)) - 1
test_matrix = xgb.DMatrix(data = as.matrix(pca.test[-53]), label=test_label)
```

```
nc = length(unique(train_label))
xgb_params = list("objective" == "binary:logistic",
                  "eval_metric" == "mlogloss",
                  "num_class" == "nc")
watchlist = list(train = train_matrix, test= test_matrix)
set.seed(1111)
xgb <- xgboost(data = train_matrix,
               eta = 0.1,
               max_depth = 6,
               nround=100,
               subsample = 0.5,
               colsample_bytree = 0.5,
               seed = 1111, gamma = 100,
               eval_metric = "mlogloss",
               objective = "multi:softmax",
               num_class = 2,
               nthread = 3)
```

```
e <- data.frame(xgb$evaluation_log)
plot(e$iter, e$train_mlogloss, col = 'blue')
```



From the above plot, we can see that the error is reducing as no of iterations increses.

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   neg   pos
##        neg 14930    14
##        pos   695   361
##
##                Accuracy : 0.9557
##                  95% CI : (0.9524, 0.9588)
##     No Information Rate : 0.9766
##     P-Value [Acc > NIR] : 1
##
##                   Kappa : 0.4868
##
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 0.9555
##             Specificity : 0.9627
##          Pos Pred Value : 0.9991
##          Neg Pred Value : 0.3419
##              Prevalence : 0.9766
##          Detection Rate : 0.9331
##    Detection Prevalence : 0.9340
##       Balanced Accuracy : 0.9591
##
```

```
##        'Positive' Class : neg
##
```

In case of prediction on test dataset, there is 0.0443 or 4.43% misclassification and 709 data points are misclassified and accuracy is 95.57% which is less than the naive classification accuracy of 97.65%. The cost obtained from this model is

Total Cost = 695 x 10 + 14 x 500 = 13,950

Here the accuracy is less, but this algorithm reduced false negatives errors significantly. Thus we got the low cost compared to all the models.

Conclusions:

Dealing with a dataset with no attribute names is quite a difficult task in terms of understanding the data w.r.t the target variable. But, with a handful of latest machine learning algorithms, we can find a way to deal with this kind of datasets especially through ensemble techniques.

In this case, we used PCA to reduce the dimensionality and worked on 5 different advanced machine learning algorithms and here our goal is to minimize the cost associated with missclassification. So, in terms of the cost, the best algorithm we found for this data is XGBoost algorithm with a minimum cost of 13,950.

Future Approach:

We can try different other algorithms with advaced techniques like bagging and pasting etc and we can fine tune the hyperparameters in the best possible way to get less minimum cost. Here, due to long running algorithms for optimizing hyperparameters, we didn't go for optimizing the hyper parameters.