Project Report: SECOM Data Set

ISEN 613: Engineering Data Analytics

Shivam Raj Solanki -	20%
Mayank Mishra -	20%
Shirish Pandagare -	20%
Divyank Garg -	20%
Randhir Nalage -	20%

Contents

Introduction:

This is a data from a semi-conductor manufacturing process. It had 1567 Number of Instances & 591 number of instances. Semiconductor manufacturing is a highly complex process with hundreds of steps involved in it. Each of these steps has a feedback signal coming from the sensors. Since semiconductor manufacturing is expensive so even a small failure in one of the steps might result in catastrophic loss hence the large number of sensors to detect any anomaly. Process engineers are required to detect an anomaly during the manufacturing as soon as possible because these products are manufactured in an environment to achieve great precision on a scale of nanometer but the existing univariate and multivariate control charts fail to do so with such high volume of data taken during the manufacturing process. Industries already have a Planned Maintenance schedule in the manufacturing sector which informs us when to carry out the maintenance of machineries involved in manufacturing depending on the running hours of that particular machinery or the time between overhauls but we need to have a predictive maintenance scheduling (especially in the case of semiconductor manufacturing) which after learning from the data available, predicts about the failure before its occurrence and thus reducing the time & money lost during the process of manufacturing. In other words, if we are provided with observations that has been generated by an unknown stochastic dependency then our goal is to infer a law that will be able to predict the future observations correctly based on the same dependency in order to maintain a high process yield in manufacturing. This is a classification problem of predicting a pass or fail for the manufactured semiconductor wafer based on the feedback coming from the sensors. There are total 590 sensors and one more feature of pass and fail state. Using these 590 sensors total 1567 observations are collected and classification model is made using the binary response of these sensors.

First, we performed imputation method to clean the data like- removing non-value column or rows, removing the non-available value cell with certain value by using different imputation method.

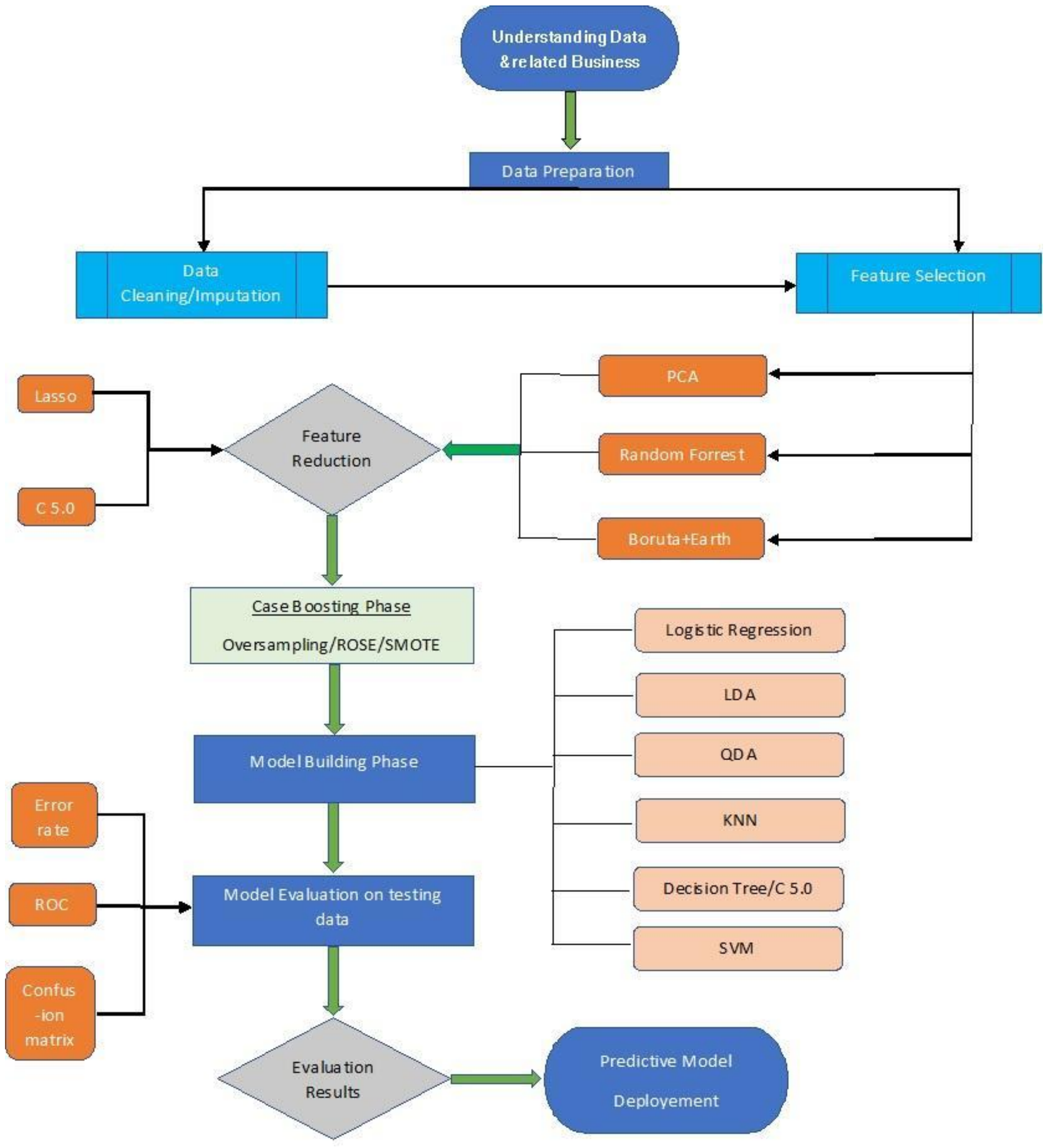
Second, the total observed data contains 1,463 pass cases with only 104 fail cases. so to equalize this two different cases a boosting technique is devised to deal with highly imbalance between the pass and fail cases.

Third, feature selection was done using different models for selecting the important sensor for designing model.

Fourth, the different methods were made for making different models and then best model was implemented.

Project Approach:

Flow Chart



We were provided with the data from a semiconductor manufacturing with a goal to classify the product in pass or fail class by creating a predictive model that follows the same dependency as the observations provided to us. With all the tools we have learned in and outside the class, we have approached in the following manner to tackle this problem: -

1. Data & Related Business Understanding:
   The most important step in data analytics/machine learning is the first one, to understand the data and the underlying Business problem related to it. We know that cost, quality, and delivery time are key factors for firms to attain long term competition and specifically speaking, semiconductor industry is a capital-intensive market sector so the stakes in this case are high and there is a little margin for error (high precision required and high downtime penalty to be paid in case of failure). Therefore, creating an automatics & advanced process control method is required so that process engineer is aware of the failure that is yet to happen and take corrective action well before it occurs in order to reduce the downtime error and maintain a high process yield thus improving business technique.

2. Data Preparation:
   The SECOM manufacturing data originally contained 1567 examples taken from a wafer fabrication production line with 590 manufacturing operation variables and 1 quality variable. There are only 104 fail cases among 1567 examples which is in a ratio of 1:14 so the classes are highly imbalanced.
   Also, almost 4.5% of the values are missing so we cannot proceed further without cleaning our data. The Columns with more than 55% missing values are eliminated using “is.na” & “data.frame”. After removing these columns, we are left with a data set containing 3% of the missing values so we carried out data imputation techniques such as mean imputation, KNN imputation and MICE imputation in order to fill those missing values with randomly generated data by regressing on the data set multiple times.

3. Feature Selection:
   With 590 features in the dataset, not all of these signals coming from the sensors are equally valuable in a specific monitoring system. It contains relevant information as well as noise. To eliminate noise, it is important to carry out feature selection. Following steps were carried out for dimensionality reduction/feature selection:-
   - All the features with more than 50% missing values were eliminated
   - Principal Component Analysis was carried out on the imputed data and it was observed that only less than 100 features could explain most of the variances for the given data. Precisely speaking, 168 features explained 95% of the variance
   - Random Forest was used on the data set and it was observed that only 27 features were important.
   - Boruta is an all relevant feature selection wrapper algorithm, capable of working with any classification method that output variable importance measure (VIM); by default,

Boruta uses Random Forest. The method performs a top-down search for relevant features by comparing original attributes' importance with importance achievable at random, estimated using their permuted copies, and progressively eliminating irrelevant features to stabilize that test. This method provided us with 20 important features.

Reference:- https://cran.r-project.org/web/packages/Boruta/Boruta.pdf

1. Feature Reduction:

   After knowing the important features, we need to select those features:-

   Lasso:- This shrinkage method was used to equate the coefficients of those features which were not important equal to zero. For obtaining the optimum λ (lambda) value, cross validation was carried out and the result of cross validation was used as an input for Lasso's lambda value. The output of the Lasso shrinkage method will be used directly for model building as it would be cleaned, imputed as well as dimensionally reduced data set with much less noise.

   C 5.0 This Package fit classification tree models or rule-based models using Quinlan's C5.0 algorithm and therefore helps us in identifying important predictors.

   Reference:- https://cran.r-project.org/web/packages/C50/C50.pdf

1. Case Boosting Phase:-

   This data presents us with an unusual form of an imbalanced case with a ratio of 1:14. In such cases models work fairly well on the prediction of a majority class while fails on the prediction of a minority class. Therefore, we need to boost the minority class or in other words, we need to make the ratio as close to 1 as possible. For this case boosting, we use the following techniques: -

i.Oversampling:

   Oversampling duplicates data from the minority class by randomly sampling data with replacement from the same class

ii. SMOTE(Synthetic Minority Over-sampling Technique):

   To shift the classifier bias towards the minority class, it artificially generates random set of minority class observations. For generating artificial data, bootstrapping and k-nearest neighbors are used.

iii. ROSE(Random Over Sampling Exercise):

   ROSE also generates synthetic balanced samples which shifts the classifier bias towards minority class. The underlying concept of ROSE is also bootstrapping which aids classification in the presence of minority class.

After case boosting, we have a balanced data set on which we can start building our classification model.

1. Model Building:-
   Since, we have a cleaned and balanced data set divided into training and testing and we know the important features, we can start building our classification model by using the following techniques we have learned in class:
   i)Logistic Regression:
   Since this is a binary classification problem so the intuition of using Logistic regression for classifying pass/fail was quite obvious. The data set was already divided into training and testing data. Logistic regression was applied on the training data and the results were tested on testing data. It shows an output with AUC 0.74
   ii)LDA (Linear Discriminant Analysis) :
   Applying LDA to fit the model of pass/fail class on the signal coming from sensors (only the important features were used for generating this model) produced almost the same results giving us the idea that the variances of both the classes are almost equal and that the assumption that it follows normal distribution holds true.
   iii)QDA (Quadratic Discriminant Analysis) :
   QDA was applied and the resulting models were compared on the basis of confusion matrix/accuracy rate, ROC area and the classification error rate. It was observed that the LDA model outperforms QDA.
   iv)Decision tree/C5.0:
   It provided a result that can be interpreted well in a way that important features were quite obvious from the pruned tree but the classification accuracy was not as high as compared to other classification technology
   v)SVM (Support Vector Machine) :
   When SVM classification was applied for model building, it outperformed all the other classification techniques when the margin was varied and the optimum margin was selected for the model, i.e a separating hyperplane with an optimum margin/cost/budget was created in the hidden feature space using polynomial programming (polynomial kernel) to find a unique solution.

   All the models were compared, cross-validated and the best model was selected on the following criteria:-
   a) Confusion Matrix/Accuracy rate
   b) ROC area (AUC)
   c) Classification error rate

Description of New Techniques:

1. **IMPUTATION:**

The dataset was obtained from a Semiconductor manufacturing process which has too many Missing Values and Redundant data. Hence to make the dataset usable we use a technique called Imputation. We used 3 types of imputation techniques on our dataset:

- ❖ KNN Imputation [9]
- ❖ MICE imputation
- ❖ MissForest Imputation

KNN Imputation: KNN imputation is used to substitute the missing data with its closest k neighbors provided the data is continuous, discrete, ordinal or categorical. Certain properties have to be kept in mind when using KNN imputation:

- The value of k: A lower value of k will increase the noise and the model won't be that generalizable. But, a higher value of k will tend to diminish the properties of local neighbors which is exactly what we're looking for.
- The assigning method: kNN imputation uses Mean, Median and Mode of the k neighbors for numerical data and just the Mode for categorical data.

Usage:

```
library(DMwR)
data_Imputed  = knnImputation(data_cleaned)
```

#this command creates a dataset data Imputed which contains the new imputed dataset obtained using knnImputation.

MICE Imputation: MICE stands for Multivariate Imputation by Chained Equations. This method uses chained equations to perform imputation on missing data. It performs better while creating multiple imputations as compared to a single imputation. It assumes that all missing values are Missing at Random (MAR), i.e. the missing values can be predicted using the observed values. It imputes the dataset on the basis of variable by variable and provides an imputation output for each of them. MICE can tackle continuous, discrete, binary variables.

The different methods used by this package are:

1. PMM (Predictive Mean Matching) – For numeric variables
2. Logreg (Logistic Regression) – For Binary Variables(with 2 levels)
3. Polyreg (Bayesian polytomous regression) – For Factor Variables (>= 2 levels)

4. Proportional odds model (ordered, >= 2 levels)

Usage:

```
library(mice)
data_IIpmm <- mice(data_c, m=1, maxit = 1, method = 'pmm', seed = 500)
data_II <- complete(data_IIpmm,1)
```

Here is an explanation of the parameters used:

1. m – Refers to 1 imputed data set
2. maxit – Refers to no. of iterations taken to impute missing values
3. method – Refers to method used in imputation. we used predictive mean matching(PMM).

MissForest Imputation: It uses random forest model which is trained on the observed value to predict the missing values in the data set. MissForest imputation is used when the data is of mixed type. random forest naturally constitutes a multiple imputation scheme by averaging over many unpruned regression or classification tree.

One of the advantage of MissForest imputation is that it yields an estimated out-of-bag(OOB) imputation error eliminating the need of a test set or an elaborate cross-validation technique. It also saves time by running in parallel. Additionally, MissForest exhibits attractive computational efficiency and can cope with high-dimensional data. we can impute continuous/categorical data having complex interactions or nonlinear relations.

Usage:

```
data_miss <- missForest(data_c)
data_III <- data_miss$ximp
```

2. CASE BOOSTING:

The total observed data contains 1,463 pass cases with only 104 fail cases. so to equalize these two different cases a boosting technique is devised to deal with highly imbalance between the pass and fail cases. We have performed 3 method for case sampling-

- ROSE (Random Over Sampling Exercise)
- SMOTE (Synthetic Minority Over-sampling Technique)
- Oversampling/ Undersampling

ROSE Boosting- The function ROSE generates synthetic balanced samples and allows to strengthen the subsequent estimation of any binary classifier. ROSE (Random Over-Sampling Examples) is a bootstrap-based technique which aids the task of binary classification in the presence of rare classes. It handles both continuous and categorical data by generating synthetic examples from a conditional density estimate of the two classes.

Usage:
hacide.rose <- ROSE(cls ~ ., data=hacide.train, seed=123)$data #generating new balance data by ROSE boosting
table(hacide.rose$cls) #check imbalance of new data

SMOTE- SMOTE is a well-known algorithm to fight the problem of unbalance cases in datasets. To counter such problem SMOTE algorithm creates artificial data based on feature space (rather than data space) similarities from minority samples. It generates a random set of minority class observations to shift the classifier learning bias towards minority class.
To generate artificial data, it uses bootstrapping and k-nearest neighbors. It basically takes the difference between feature vectors and it nearest neighbors and multiply the difference by random number in between 0 and 1 and add it to feature vector under consideration. This result to the selection of random point along the line segment between two specific features.

Usage:
train$Class = as.factor(train$Class)
train = SMOTE(Class ~ ., train, perc.over = 300, perc.under=100) #generating new balance data by ROSE boosting
table(train$Class) #check imbalance of new data

UNDERSAMPLING- This method reduces the number of observations from those class which are in majority in the data set so to balance the cases. This method mostly works better when the data is huge in size and reducing the number of training samples helps to reduce the modelling run time and storage troubles.
There are two types of Undersampling methods:

- Random
- Informative

Usage:
under_sampled_data = ovun.sample(Class ~ ., data = train, method = "under")$data # to reduce the number of observations from majority class data set.
table(under_sampled_data$Class) # check the size of table

OVERSAMPLING- In an dataset, the class having the lowest number of observation is taken into account and all existing observations are taken and copied and extra observations are added by randomly sampling with replacement from this class. It is just opposite to undersampling.
Usage:
over_sampled_data = ovun.sample(Class~ ., data =train, method = "over")$data # to increase the number of observations of minority class data set.
table(over_sampled_data$Class) # check the size of table

3. FEATURE SELECTION AND MODEL DEVELOPING

C5.0- C5.0 algorithm is a successor of C4.5 algorithm also developed by Quinlan (1994). To understand the working of C5.0, we must know how C4.5, a descendant of CLS and ID3, generates classifiers that can be used for the classification problems. It extends the ID3 algorithm by dealing with both continuous and discrete attributes, missing values and pruning tress after construction.

The algorithm for the C5.0 is presented as follows: -

Input: - A set of set S (continuous or discrete attributes) each belonging to one class.

Output: - A decision tree or a set of rules that assigns a class to a new case.

Algorithm: -

1. Check for the base case
2. Find the attribute with the highest information gain.
3. Partition S into S1, S2, S3…. according to the values of the attribute selected in Step 2
4. Repeat the steps for S1, S2, S3….

The base cases are the following

- All the examples from the training set belong to the same class (a tree leaf labelled with that class is returned).
- The training set is empty (returns a tree leaf called failure).
- The attribute list is empty (return a leaf labelled with the most frequent class or the disjunction of all the classes)

The attribute with the highest information gain is computed using the following formulas

Entropy: $E(S) = \sum_{=1}^{n} \quad - Pr\,(Ci) * log_2\, Pr(Ci)$

Informational Gain: $G(S,A) = E(S) - \sum_{i=1}^{m} \blacksquare\, Pr\,(Ai)\, E(Si)$

E(S) => Information entropy of S

G(S,A) =>gain of S after a split on attribute A

N =>Number of classes in S

Pr(Ci) => frequency of class Ci in S

m => Number of values of attribute A in S

Pr(Ai) => frequency of classes that have Ai value in S

E(Si) => Subset of S with items that have Ai value

The C4.5 algorithm improves the ID3 algorithm by allowing numerical attributes, permitting missing values and performing Tree pruning.

C4.5 was superseded by See5/C5.0 (or C5.0 for short). The changes encompass new capabilities as well as much-improved efficiency, and include:

1. A variant of boosting, which constructs an ensemble of classifiers that are then voted to give a final classification. Boosting often leads to a dramatic improvement in predictive accuracy.
2. New data types (e.g., dates), “not applicable” values, variable misclassification costs, and mechanisms to pre-filter attributes.
3. Unordered rulesets—when a case is classified, all applicable rules are found and voted. This improves both the interpretability of rulesets and their predictive accuracy
4. Greatly improved scalability of both decision trees and (particularly) rulesets. Scalability is enhanced by multi-threading; C5.0 can take advantage of computers with multiple CPUs and/or cores.

Usage:-
```
model1=C5.0(data1,data2)
model1
```

Code: - **C5.0(X,Y)** where X is the predicting features while the Y is the Response.

The entries of the Y should be the factor to run the code, Hence **as.factor** is used to covert the entries into factor.

Implementation Details:

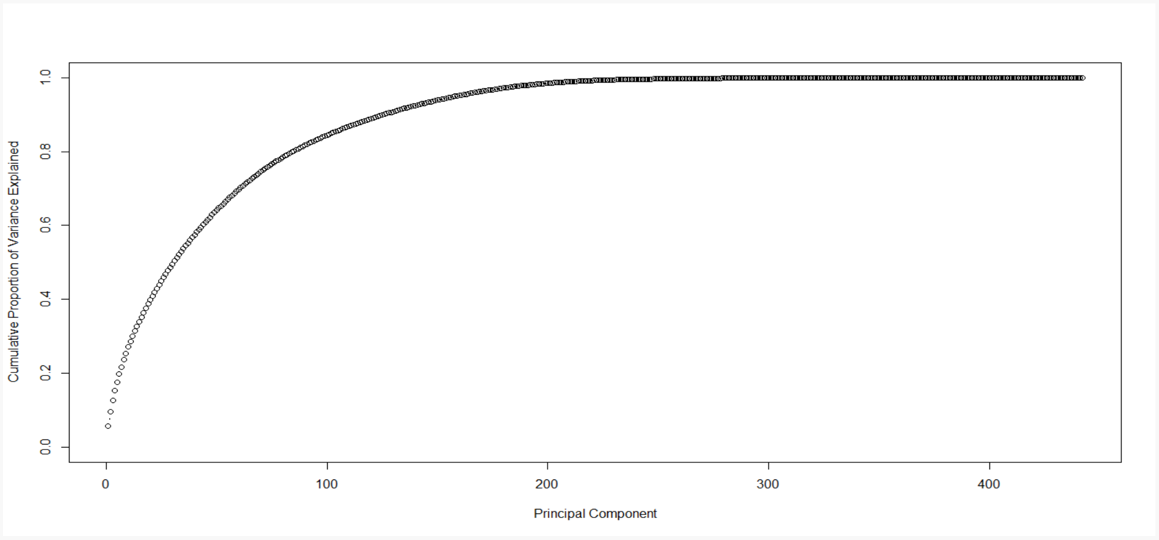
```
"Data Set Input and initial analysis
The dataset we started working on was obtained from the UCI
repository. It initially had 1567 observations (rows) which were
outputs of 590 sensor measurements (columns/variables) and a label of
Yield Pass/Fail."

#Loading data to workspace

library(data.table)
feature=fread("C:/Users/Randhir/Desktop/ISEN613 project/SECOM
UCI/secom.data.txt", data.table = F)
label = fread("C:/Users/Randhir/Desktop/ISEN613 project/SECOM
UCI/secom_labels.data.txt", data.table = F)
data = cbind(label,feature)#Combining the variates with their Labels
colnames(data) = c("Class", "Time", paste0(rep("Feature", ncol(feature)),
seq(1,ncol(feature))))
data$Class = factor(data$Class, labels = c("pass", "fail"))
data$Time =  as.POSIXct(data$Time, format = "%d/%m/%Y %H:%M:%S", tz = "GMT")
dim(data)

## [1] 1567  592

"On close observation we find that there are many missing values and
equal values which need to be tackled before any further operation on
the Dataset."

sum(is.na(data))

## [1] 41951

Here there are 41951 missing values in the table.

#Step 1: Removing Redundant data and Missing Values

# Here the Time variable is redundant for our study so we need to
remove it.
index_1 = which(colnames(data) == "Time")
```

```
# Remove columns with equal value as those features won’t help in
training.
equal = apply(data, 2, function(x) max(na.omit(x)) == min(na.omit(x)))
index_2 = which(equal == T)
# We remove the features that have more than 40% of values missing
rowsNA = apply(data, 1, function(x) sum(is.na(x))/ncol(data))
colsNA = apply(data, 2, function(x) sum(is.na(x))/nrow(data))
indexm3 = which(col_NA > 0.4)

"index1, index2, index3 might have common column numbers.unique
function is used to remove the repeating column numbers"

data_cleaned = data[,-unique(c(index1, index2, index3))]

#We have reduced the number of features to 443 after above steps.
dim(data_cleaned)

## [1] 1567  443

#Initially the missing values in the data set was 41951 which we have
reduced to 8008 now.
sum(is.na(data_cleaned))

## [1] 8008

"Now we cannot delete all the rows or columns with missing values.
That would lead to loss of
data and our prediction would be biased. So our next step is to fill
the missing values.

#Step 2: Imputation using different techniques

#Imputation

#Knn Imputation
library(DMwR)

## Loading required package: lattice

## Loading required package: grid

data_Imputed  = knnImputation(data_cleaned)
sum(is.na(data_Imputed))

## [1] 0     #no missing values
```

```
dim(data_Imputed)

## [1] 1567  443   #all observations are retained

fix(data_Imputed)


#MICE Imputation
install.packages("mice")
library(mice)
data_mice_pmm <- mice(data_cleaned, m=1, maxit = 1, method = 'pmm', seed =
500)
data_mice <- complete(data_mice_pmm,1)"

"we are getting a dataset with 117 missing values using mice.We will
store this data into another dataframe and use knn to fill remaining
117 values"


data_miceII=data_mice
sum(is.na(data_miceII)) #117 missing values

## [1] 117

data_mice= knnImputation(data_miceII)
sum(is.na(data_mice))

## [1] 0    #zero missing values after applying knnImputation

#MissForest Imputation
install.packages("missForest")
library(missForest)
data_miss = missForest(data_cleaned)
data_III = data_miss$ximp"



#Step 3: Splitting into training & testing data
set.seed(2)
index = sample(1:nrow(data_Imputed), nrow(data_Imputed)/10)
train = data_Imputed[-index,] #training data set
test = data_Imputed[index,]   #testing data set


#Step 4: Feature Selection techniques
```

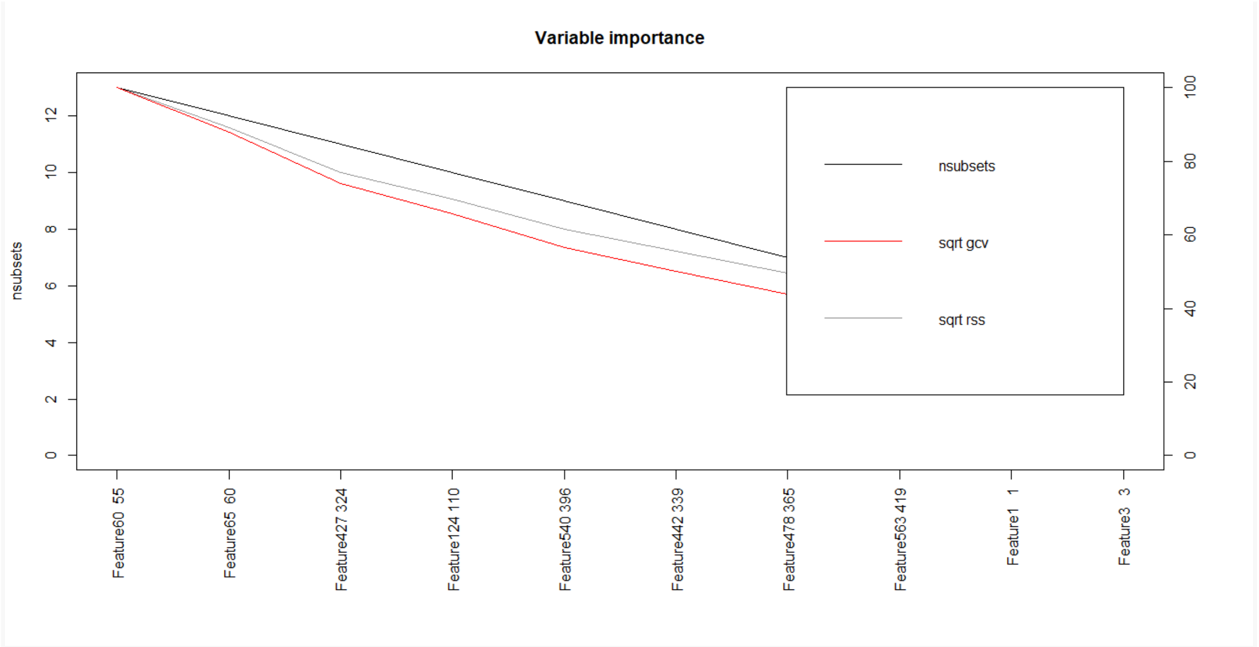
```
#We use PCA, Random Forest, Boruta, Mars and Lasso for our feature selection
out of the Multivariate data, Lasso gave us the best results but we have
shown the code for rest too.


#Principal Component Analysis

train_pca = train[,-1] #removing of the classification column (PASS/FAIL)

test_pca = test[,-1]   #removing of the classification column (PASS/FAIL)



pr.out=prcomp(train_pca, scale=TRUE)

names(pr.out)


#mean of variables

pr.out$center

#standard deviation of variables

pr.out$scale

#principal component loading vector

pr.out$rotation

dim(pr.out$x)


#scree analysis


pr.out$sdev

pr.var=pr.out$sdev^2

pr.var

#proportion of variance explained
```

```
pve=pr.var/sum(pr.var)

plot(cumsum(pve), xlab="Principal Component", ylab="Cumulative Proportion of Variance Explained", ylim=c(0,1),type='b')
```



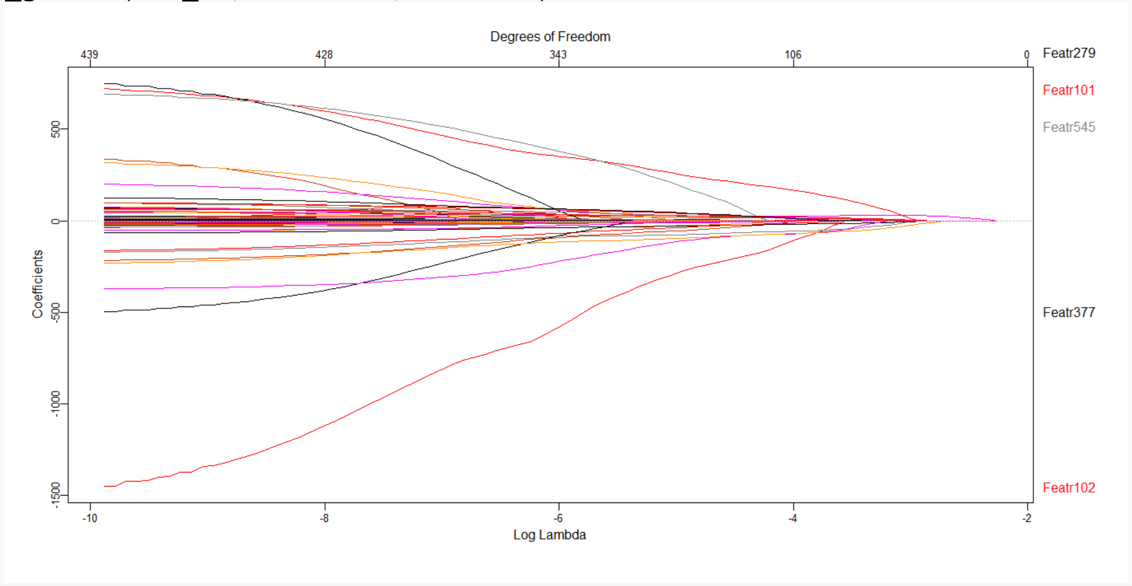
```
#we select first 170 components as the plot shows them to explain most of the variance

train_pca_1 = data.frame(Class=train$Class, pr.out$x)

train_pca_170 = train_pca_1[,1:170]

fix(train_pca_170)

test_pca__1 = predict(pr.out, newdata = test_pca)

test_pca__1 = as.data.frame(test_pca__1)


test_pca__1 <- test_pca__1[,1:170]

#The output Principle Component were not supporting Case Boosting and since it is a crucial part of our Model Building we skip PCA

#Random Forest
library(randomForest)
## randomForest 4.6-12
## Type rfNews() to see new features/changes/bug fixes.
```

```
rf.features =randomForest(Class~., data_Imputed, mtry=20, importance =TRUE)
rf.features
##
## Call:
##  randomForest(formula = Class ~ ., data = data_Imputed, mtry =
20,      importance = TRUE)
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 20
##
##         OOB estimate of  error rate: 6.7%
## Confusion matrix:
##      pass fail class.error
## pass 1462    1 0.000683527
## fail  104    0 1.000000000
importance(rf.features)
##                    pass         fail MeanDecreaseAccuracy MeanDecreaseGini
## Feature1   -1.621137816 -0.538241617          -1.693710694       0.70635755
## Feature2   -0.145959777 -1.056935564          -0.366699802       0.24076111
## Feature3    3.167396473  1.248323050           3.216230603       0.68749187
## Feature4   -0.538262803  0.542425790          -0.390239953       0.17368695
## Feature5    0.280887786 -0.714568846           0.021586945       0.50319153
## Feature7    0.540277976 -0.902020167           0.197879190       0.20495385
## Feature8   -0.174423152 -0.500285128          -0.314123034       0.24448674
## Feature9    2.170411210 -1.999482651           1.464558675       0.21157951
## Feature10   0.773886913  0.982572333           1.017045424       0.32324400
## Feature11   0.790156954  0.006096465           0.757362663       0.36208535
## Feature12   2.302089035 -2.538175154           1.822217765       0.49785759
## Feature13  -0.192620068  0.078123616          -0.097269506       0.56628484
## Feature15   1.564633341  0.404922372           1.532258772       0.44674349
## Feature16   0.474937141 -0.808843839           0.174953395       0.29335527
## Feature17   1.348103233  2.324139630           2.051046463       0.80584631
## Feature18   1.199377010 -0.539118427           1.100387531       0.39303426



# Mars Feature Selection Method
library(earth)
## Loading required package: plotmo
## Loading required package: plotrix
## Loading required package: TeachingDemos
marsModel = earth(Class ~ .,data_Imputed) # build model
ev = evimp (marsModel)
# estimate variable importance
plot (ev)
```

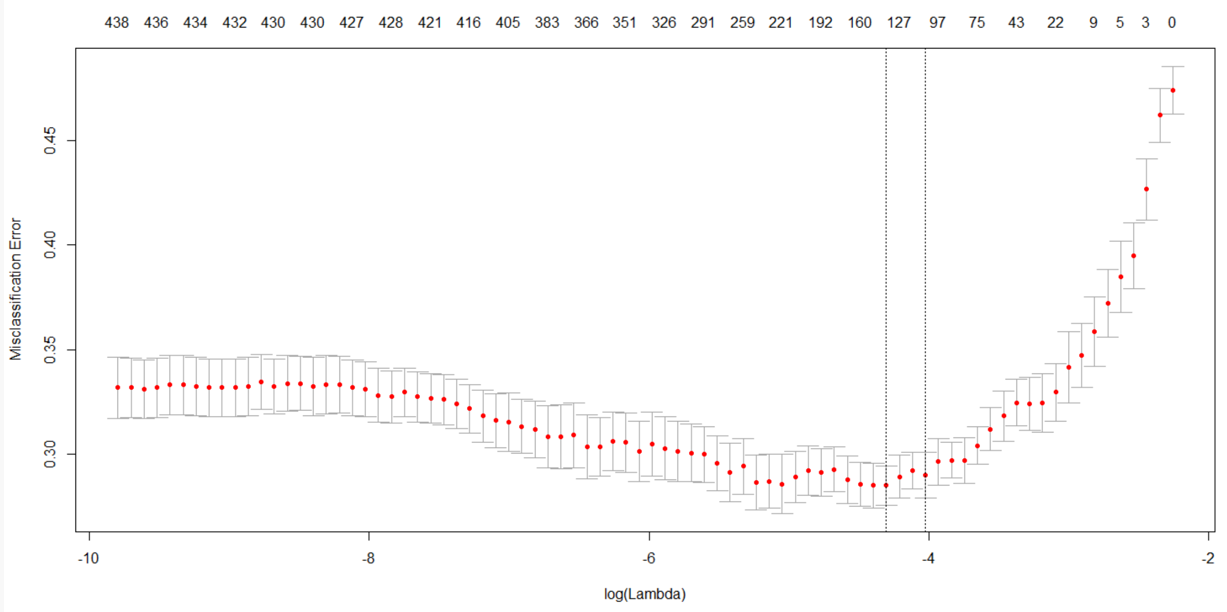


```
#The plot gives us the important features from MARS model which are limited
to 10 in number.
ev
##             nsubsets   gcv    rss
## Feature60        13 100.0  100.0
## Feature65        12  87.9   89.2
## Feature427       11  74.0   77.1
## Feature124       10  65.9   69.7
## Feature540        9  56.6   61.5
## Feature442        8  50.1   55.5
## Feature478        7  43.9   49.7
## Feature563        6  38.5   44.4
## Feature1          5  31.5   38.0
## Feature3          3  19.2   26.0
#Boruta Feature Selection Method
library(Boruta)
## Loading required package: ranger
##
## Attaching package: 'ranger'
## The following object is masked from 'package:randomForest':
##
##     importance
set.seed(123)
boruta.train = Boruta(Class ~ ., data_Imputed, doTrace = 2, ntree=500).
print(boruta.train)
final.boruta = TentativeRoughFix(boruta.train)
```

```
print(final.boruta)
getSelectedAttributes(final.boruta, withTentative = F)
##  [1] "Feature3"   "Feature17"  "Feature39"  "Feature41"  "Feature60"
##  [6] "Feature64"  "Feature65"  "Feature66"  "Feature79"  "Feature104"
## [11] "Feature154" "Feature170" "Feature198" "Feature268" "Feature289"
## [16] "Feature337" "Feature342" "Feature349" "Feature427" "Feature442"
## [21] "Feature478" "Feature511" "Feature540" "Feature563"

#Boruta gives us 24 important features



#Lasso
library(glmnet)
## Loading required package: Matrix
## Loading required package: foreach
## Loaded glmnet 2.0-13
fit_LS = glmnet(as.matrix(train[,-1]), train[,1], family="binomial", alpha=1)
plot_glmnet(fit_LS, "lambda", label=5)
```



```
library(plotmo)
fit_LS_cv = cv.glmnet(as.matrix(train[,-1]), as.matrix(as.numeric(train[,1])-
1), type.measure="class", family="binomial", alpha=1)
plot(fit_LS_cv)
```



#The Misclassification Error rate is minimum at the point where the number of predictors is around 127, we take out the important predictors from Lasso as shown below

```
coef = coef(fit_LS_cv, s = "lambda.min")
coef_df = as.data.frame(as.matrix(coef))
lasso_op = rownames(coef_df)[which(coef_df[,1] != 0)][-1]
```

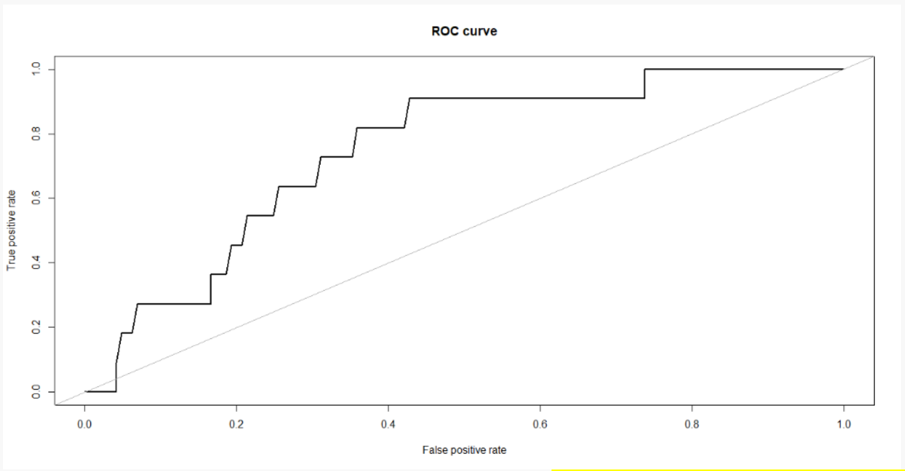
#Step 5: Case Boosting

As mentioned in the project approach we have to use a new technique called Case Boosting to have a unbiased training dataset. Currently the number of Pass and Fail are in the ratio of 1:14

#We use three techniques

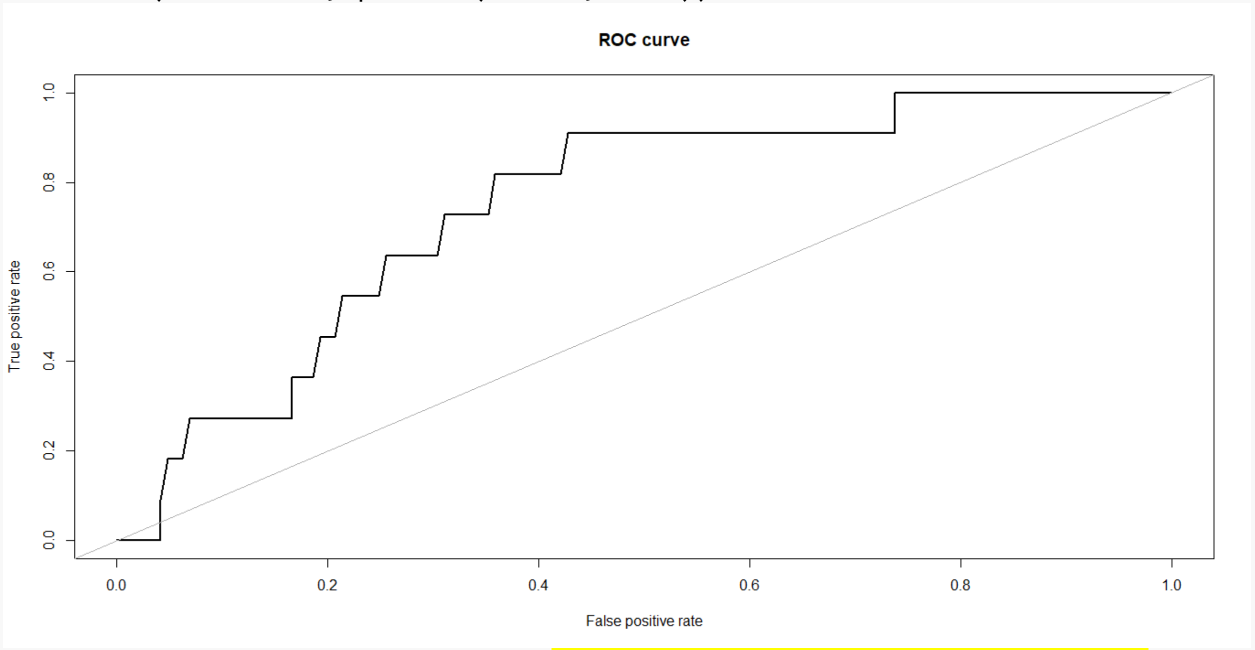
```
#1.OverSampling
library('ROSE')
## Loaded ROSE 0.0-3
over_sampled_data = ovun.sample(Class~ ., data =train, method = "over")$data
table(over_sampled_data$Class)
##
## pass fail
## 1318 1331
#2.ROSE
train_rose = ROSE(Class ~ ., data = train, seed = 1)$data
```

```
table(train_rose$Class)
##
## pass fail
##  741  670
#3.SMOTE
library(DMwR)
train$Class = as.factor(train$Class)
train = SMOTE(Class ~ ., train, perc.over = 300, perc.under=100)
table(train$Class)
##
## pass fail
##  279  372



#Step 6: MODEL BUILDING

#1.Logistic Regression
LR.fit = glm(Class ~ ., data=train_rose[,c("Class",lasso_op)], family =
"binomial")
pred_LR = factor(ifelse(predict(LR.fit, test, type = "response") > 0.7,
"fail", "pass"), levels = c("pass", "fail"))
table(test$Class, pred_LR)
     pred_LR
      pass fail
 pass  124   21                          #CONFUSION MATRIX
 fail    8    3
#Accuracy: 81.41
roc.curve(test$Class, predict(LR.fit, test))
```

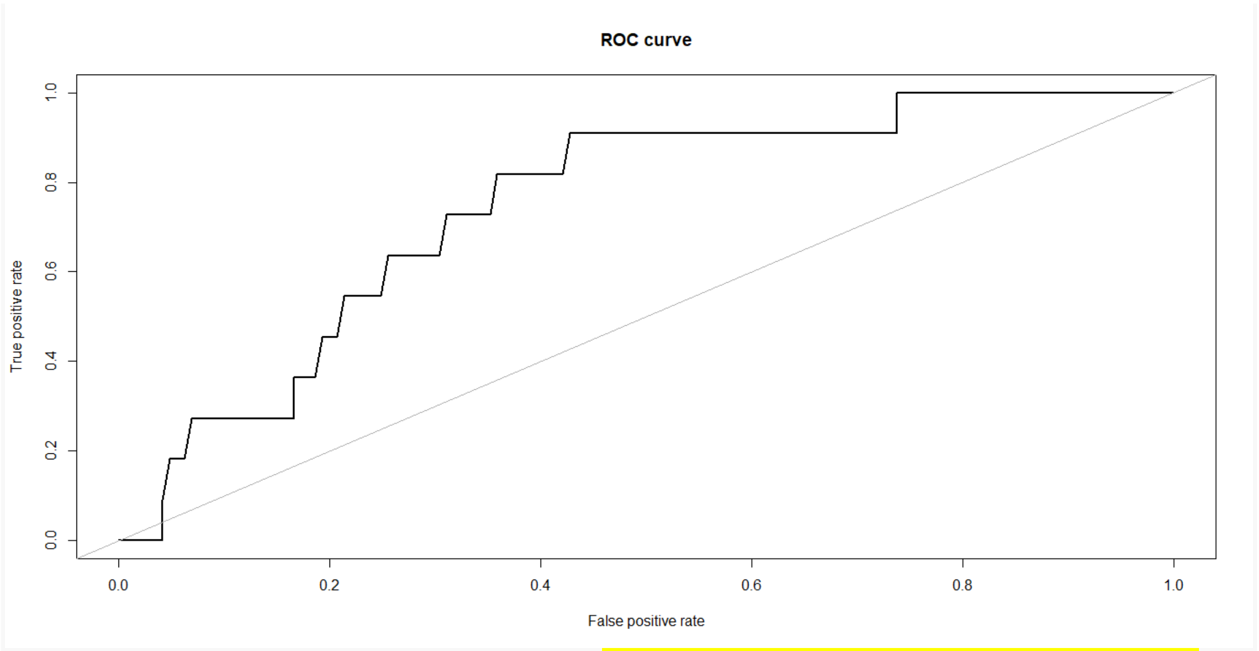


```
Area under the curve (AUC): 0.746 #Higher the AUC better the Classifier
```

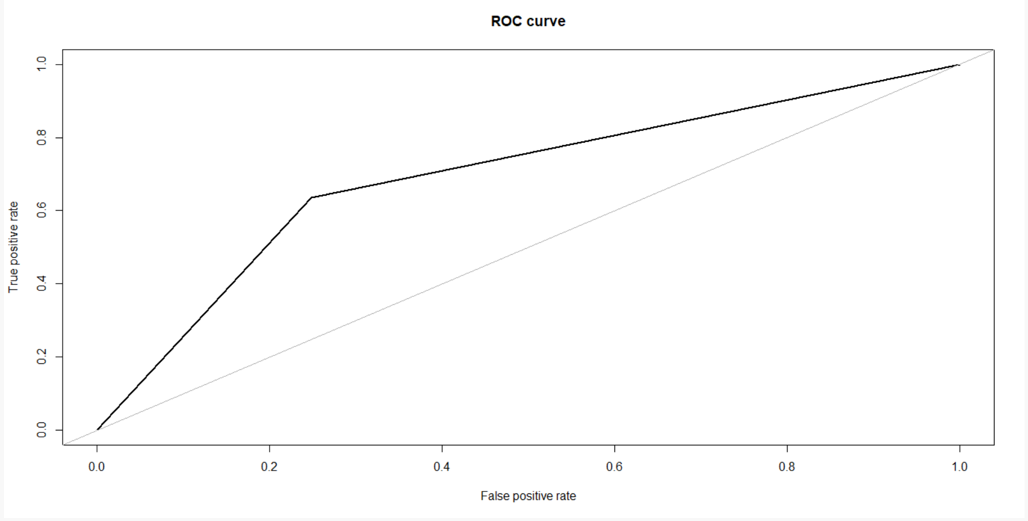
```
#Reducing threshold value to decrease the False Negative rate while keeping
#the total error optimum required Low False Negative Rate
pred_LR = factor(ifelse(predict(LR.fit, test, type = "response") > 0.6,
"fail", "pass"), levels = c("pass", "fail"))
table(test$Class, pred_LR)
     pred_LR
      pass fail
 pass  116   29                          #CONFUSION MATRIX
 fail    6    5
#Accuracy: 77.56
#Reducing threshold value to decrease the False Negative rate while keeping
#the total error optimum
roc.curve(test$Class, predict(LR.fit, test))
```



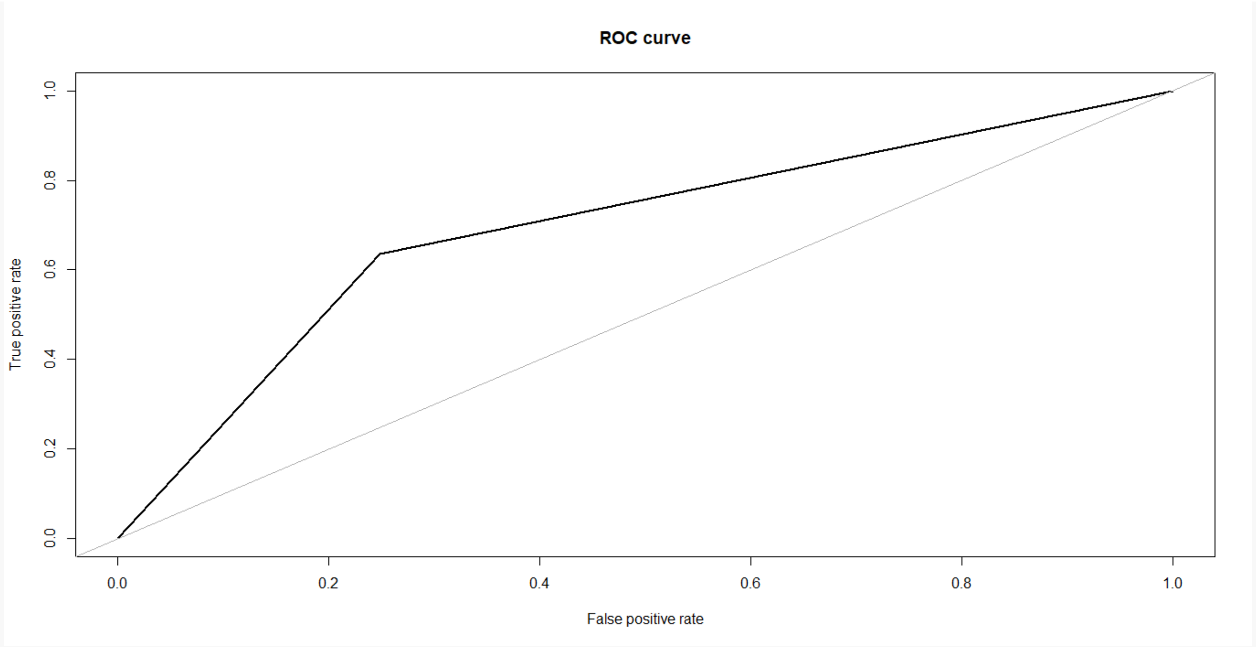
```
Area under the curve (AUC): 0.746 #Higher the AUC better the Classifier
pred_LR = factor(ifelse(predict(LR.fit, test, type = "response") > 0.5,
"fail", "pass"), levels = c("pass", "fail"))
table(test$Class, pred_LR)
     pred_LR
      pass fail
 pass  111   34                          #CONFUSION MATRIX
 fail    5    6
#Accuracy: 75
roc.curve(test$Class, predict(LR.fit, test))
```



```
## Area under the curve (AUC): 0.741 #Higher the AUC better the Classifier

#LDA
library(MASS)
lda.model = lda(Class ~ ., data=train_rose[,c("Class",lasso_op)])
lda.pred=predict(lda.model,test)
lda.class=lda.pred$class
table(lda.class,test$Class)
##
## lda.class pass fail
##      pass  109    5                    #CONFUSION MATRIX
##      fail   36    6
mean(lda.class==test$Class)
#Accuracy
## [1] 0.7371795
roc.curve(test$Class, predict(lda.model, test)$class)
```

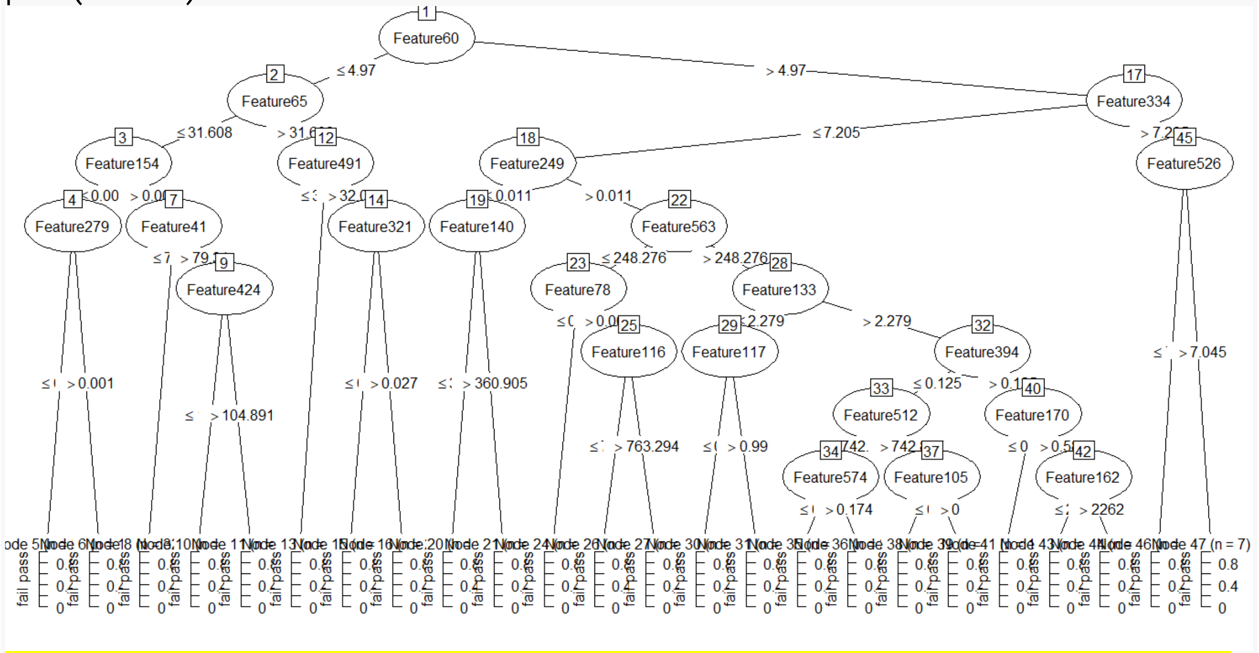


```
## Area under the curve (AUC): 0.649 #Higher the AUC better the Classifier


#QDA
library(MASS)
train_lasso=train_rose[,c("Class",lasso_op)]
qda.fit = qda(Class ~ .,train_lasso)
qda.pred=predict(qda.fit,test)
qda.class=qda.pred$class
table(qda.class,test$Class)
##
## qda.class pass fail
##      pass   89    3                        #CONFUSION MATRIX
##      fail   56    8
mean(qda.class==test$Class)
#Accuracy
## [1] 0.6217949
roc.curve(test$Class, qda.class)
```



```
## Area under the curve (AUC): 0.671

#C5.0
#Develops Multiple Decision Trees using all attributes and selects the best
model for making predictions.
library(C50)
attach(train)
dim(train)
## [1] 651 443
data1=train[1:651,-1]
data2=as.factor(train[1:651,1])
model1=C5.0(data1,data2)
model1
##
## Call:
## C5.0.default(x = data1, y = data2)
##
## Classification Tree
## Number of samples: 651
## Number of predictors: 442
##
## Tree size: 32
##
## Non-standard options: attempt to group attributes
summary(model1)
##
## Call:
```

```
## C5.0.default(x = data1, y = data2)
##
##
## C5.0 [Release 2.07 GPL Edition]      Sun Dec 10 12:37:00 2017
## -------------------------------
##
## Class specified by attribute `outcome'
##
## Read 651 cases (443 attributes) from undefined.data
##
## Time: 0.5 secs
plot(model1)
```

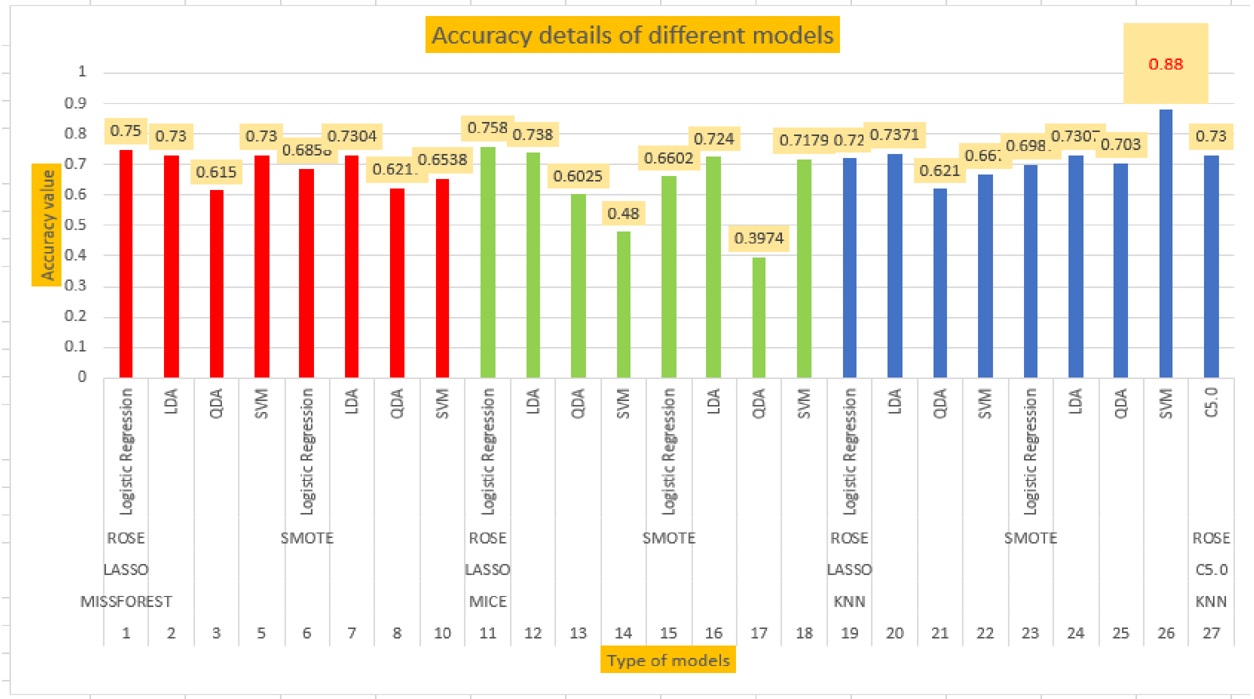


```
#Various decision trees were iterated and the best tree model explaining the
model is shown above
dim(test)
## [1] 156 443
test_result=predict(model1,test[1:156,-1])
summary(test_result)
## pass fail
##  109   47
test_result
##   [1] fail pass pass pass pass pass pass fail pass fail pass fail pass
fail
##  [15] pass fail fail pass pass pass pass pass pass pass pass pass fail
pass
##  [29] pass fail pass fail pass pass pass pass pass fail pass pass pass
fail
```

```
##  [43] fail fail pass pass pass pass fail pass fail pass pass pass pass
fail
##  [57] fail pass fail pass fail pass pass pass fail pass pass fail fail
pass
##  [71] fail pass pass fail fail fail fail fail pass pass pass pass fail
pass
##  [85] pass pass pass fail pass pass pass pass pass pass pass pass pass
pass
##  [99] fail pass fail pass pass pass fail pass pass pass fail pass pass
fail
## [113] pass pass pass fail pass fail pass fail fail fail fail pass pass
pass
## [127] pass pass pass pass pass pass fail pass pass pass pass pass fail
fail
## [141] fail pass pass pass pass pass fail pass pass pass pass pass pass
pass
## [155] pass pass
## Levels: pass fail
table(test_result,test[1:156,1])
##
## test_result pass fail
##        pass  106    3
##        fail   39    8
#Accuracy: 73.07


#SVM
library(e1071)
svm.fit=svm(Class~., data=train_rose[,c("Class",lasso_op)],
kernel="polynomial", cost=10,scale=FALSE)
svm.pred=predict(svm.fit,test)
summary(svm.fit)
## Call:
## svm(formula = Class ~ ., data = train_rose[, c("Class", lasso_op)],
##     kernel = "polynomial", cost = 10, scale = FALSE)
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  polynomial
##        cost:  10
##      degree:  3
##       gamma:  0.00625
##      coef.0:  0
## Number of Support Vectors:  604
##  ( 308 296 )
## Number of Classes:  2
## Levels:
##  pass fail
table(svm.pred,test$Class)
## svm.pred pass fail
##     pass   98    5
##     fail   47    6
mean(svm.pred==test$Class)
## [1] 0.8846154 #Accuracy



#Hence we finally selected SVM because of highest accuracy rate:
```

Comparison between different Models:

Sl.No	Imputation	Feature Selection	Case Boosting	Model	Prediction Accuracy
1	MISSFOREST	**LASSO**	ROSE	Logistic Regression	0.75
2				LDA	0.73
3				QDA	0.615
5				SVM	0.73
6			SMOTE	Logistic Regression	0.6858
7				LDA	0.7304
8				QDA	0.6217
10				SVM	0.6538
11	MICE	**LASSO**	ROSE	Logistic Regression	0.758
12				LDA	0.738
13				QDA	0.6025
14				SVM	0.48
15			SMOTE	Logistic Regression	0.6602
16				LDA	0.724
17				QDA	0.3974
18				SVM	0.7179
19	KNN	**LASSO**	ROSE	Logistic Regression	0.72
20				LDA	0.7371
21				QDA	0.621
22				SVM	0.667
23			SMOTE	Logistic Regression	0.6987
24				LDA	0.7307
25				QDA	0.703
26				SVM	**0.88**
27	KNN	C5.0	ROSE	C5.0	0.73



We finally selected the SVM model using KNN imputation for our Predictions which uses LASSO for feature selection and ROSE for Case Boosting because out of different models' accuracy results we found that this model has the highest accuracy of 88%. The data was first imputed using KNN, MICE and MISSFOREST and for each imputation we then did the feature selection using LASSO, RandomForest, PCA and then case boosting was done using ROSE and SMOTE. Using these all permutations and combinations we got different datasets for individual process path and then model was made using Logistic Regression, LDA, QDA, SVM and C5.0 to find out the best model with highest accuracy.

Executive Summary:

This project deals with making predictive models for equipment fault detection in semiconductor manufacturing process. Semiconductor manufacturing is one of the most technologically complicated process and involves many variables that are monitored during manufacturing. In past many Machine Learning algorithms like multivariate analysis have been deployed for creating predictive models to detect faults. As per the semiconductor industry statistics the equipment usually suffer an 8% unscheduled downtime. If we can make a predictive model for preventive maintenance to reduce this 8% unscheduled downtime, it will improve the productivity significantly. Predictive Maintenance is the process of identifying when equipment needs maintenance to make sure that failure of equipment is avoided.

Our Objective is to make a model that classifies manufactured semiconductor wafer as yield pass/fail. We are training this model on a data set that consists of 590 sensor attributes and 1567 observations. These values are recorded by different sensors during the manufacturing process.

The first problem we faced was the missing values in the dataset. We had to first clean the data set which can be identified as the data cleaning phase. Columns with more than 40% missing values and columns consisting same data in all observations were removed. After initial cleaning of data, we learned new imputation techniques like Mice, KNN, missForest and applied them to fill the missing values. Another issue we faced while training the model was that the ratio of Yield Pass and Fails was 14:1 [1463 Pass and only 104 fails]. Training the model on this data would create a biased model. To encounter this issue, we used Case Boosting techniques such as ROSE and Smote that would equalize the pass/fail cases reducing the bias.

Monitoring all the variables/sensors is not a practical solution to decide if a wafer is pass/fail. After boosting we then do feature selecting using different models for selecting the important sensors for designing model. Out of all the models we got best result by doing KNN imputation, Lasso feature selection followed by ROSE Case boosting and finally modeling using SVM. This model has a prediction accuracy of 88%. We can use this model for preventive maintenance and predict with an accuracy of 88% to determine if a semiconductor wafer will pass or fail. This will reduce the 8% unscheduled downtime considerably.

Conclusion:

Semiconductor manufacturing is a highly complex process and the industry is one of the most capital-intensive industry. Process excursion during manufacturing can significantly impact product yield and manufacturing cost as well. So, for such processes, real time data is available and our job is to build a predictive classification model since Detection of root cause gives process Engineers action plan to improve manufacturing robustness and prevent future process excursions. With such a large number of features in the manufacturing process, the most important step is the causal feature selection for effective monitoring & process control as this causal feature selection informs process engineers that which of features (sensor signals) are most important for the particular process so that preventive/predictive maintenance of the related machinery/equipment is carried out well before time in order to reduce any downtime that is supposed to occur because of that breakdown.

Therefore, we synthesized the classification model building techniques along with the feature selection methods to ensure that only the most important features (excluding the noises) are used for building classification model which will be used for prediction of pass/fail class to the semiconductor wafer manufactured. After data cleaning and imputation, we have tried various feature selection methods such as Principal Component analysis, Lasso, C5.0 and Random Forest. The best results obtained from Lasso was then used to build a classification model using various techniques such as Logistic Regression, LDA, QDA, KNN, Decision tree & SVM. After comparing on the basis of confusion matrix/accuracy rate, AUC & Classification error rate and cross validating all the models, SVM was found to be the best among the above mentioned.

Hence, the SVM classification model was deployed and the prediction model was built. This would help the process Engineers in the semiconductor manufacturing industry to improve the yield by preventing breakdown thus reducing downtime and improving the business model.

References:

1. Author: Gareth James, Daniela Witten, Trevor Hastie, Robert Tibshirani
   Title: An Introduction to Statistical Learning
   Edition: 7th edition
   URL: http://www-bcf.usc.edu/~gareth/ISL/ISLR%20First%20Printing.pdf
2. Online site URL: https://www.analyticsvidhya.com/blog/tag/statistics/
3. Author: Sathyan Munirathinam and Balakrishnan Ramadoss
   Journal: IACSIT International Journal of Engineering and Technology, Vol. 8, No. 4, August 2016
   Title: Predictive Models for Equipment Fault Detection in the Semiconductor Manufacturing Process
   URL: http://www.ijetch.org/vol8/898-T10023.pdf.
4. Author: P.S. Frankwicz, S. E. Romano and T. Moutinho
   Title: Process Excursion Detection using Statistical Analysis Methodologies in High Volume Semiconductor Production
   URL: http://www.lexjansen.com/nesug/nesug09/po/PO11.pdf
5. Author: Kittisak Kerdprasop and Nittaya Kerdprasop
   Journal: INTERNATIONAL JOURNAL OF MECHANICS
   Title: A Data Mining Approach to Automate Fault Detection Model Development in the Semiconductor Manufacturing Process
   URL: http://www.naun.org/main/NAUN/mechanics/17-220.pdf
6. Author: Michael McCann, Yuhua Li, Liam Maguire,Adrian Johnston Journal:Workshop and Conference Proceedings 6: 277-288 Title:Causality Challenge: Benchmarking relevant signal components for effective monitoring and process control
   URL: http://proceedings.mlr.press/v6/mccann10a/mccann10a.pdf
7. Online site URL: https://stackoverflow.com/questions/13114812/imputation-in-r
8. KNN Imputation: https://towardsdatascience.com/the-use-of-knn-for-missing-values-cf33d935c637
9. R-programming
   Online site URL: https://www.datacamp.com/
10. http://archive.ics.uci.edu/ml/datasets/SECOM
11. Reference:- https://www.r-bloggers.com/imputing-missing-data-with-r-mice-package/
12. Reference:- https://cran.r-project.org/web/packages/Boruta/Boruta.pdf