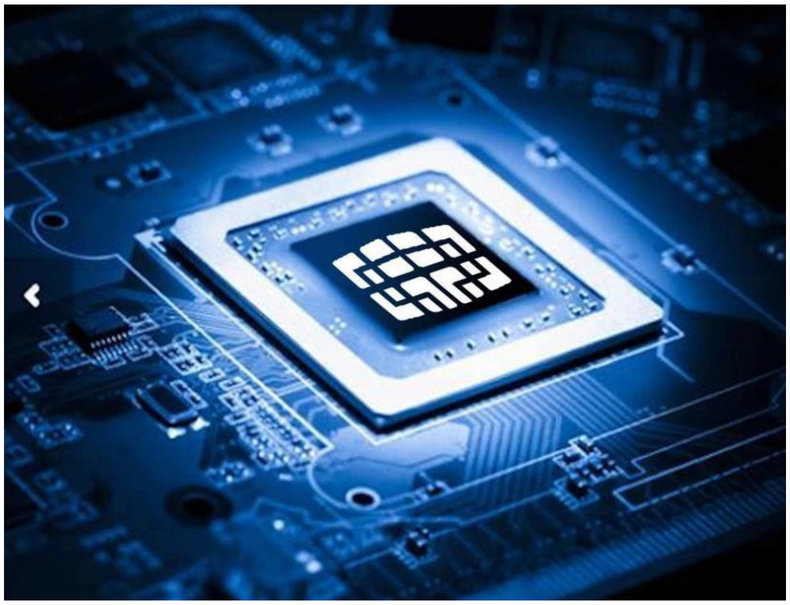
Analysis of semi-conductor process dataset using PCA and Classification methods



Apoorva Modali

Kaushik Arunapuram

Pallav Anand

Priyanka Srikanth

PROBLEM STATEMENT:

(i) Detect and monitor pass/fail signals from sensors of a semiconductor manufacturing dataset using different analytical methods.

(ii) Learn new methods of solving classification problems and dimensionally reducing a large dataset.

DATASET:

SECOM data set (Link: http://archive.ics.uci.edu/ml/datasets/SECOM)

The data consists of 2 files the dataset file SECOM consisting of 1567 examples each with 591 features a
1567 x 591 matrix and a labels file containing the classifications and date time stamp for each example. As with any real life data situations this data contains null values varying in intensity depending on the individual features. This needs to be taken into consideration when investigating the data either through pre-processing or within the technique applied. The data is represented in a raw text file each line representing an individual example and the features separated by spaces.

LITERATURE REVIEW:

Research Paper 1

In this paper, a new adaptive sub statistical PCA model is proposed for semiconductor manufacturing process monitoring. By employing this method, several drawbacks of the conventional method are addressed. Time complexities and correlations among ASSP, MMPCA, and MPCA are analysed in detail. Advantages of the proposed method are that 1) cross-information between data blocks is modelled, while low time complexity is needed; 2) future value estimation is avoided; and 3)no Gaussian assumption of process data is needed. However, the proposed method is limited in linear and stationary cases. Consideration of how to extend it for nonlinear and dynamic process monitoring is still under way.

Research Paper 2

The paper analyses fault detection accuracy in semiconductor manufacturing process. The sensors give 590 features which can be used for fault prediction. The paper proposes a feature section method to filter out irrelevant predictors. They also propose a boosting technique to handle the skewness of pass versus fail tests. The wafer fabrication data collected from 590 sensors with the last feature is a label stating pass or fail state. The 1567 observations contain 1463 pass cases and only 104 fail cases. The features which have constant value or have more than 55% missing values are removed first. In the cleaning step, we remove 137 features that contain a single value and lots of missing values. Further feature selection is done using statistical based analysis such as chi-square and principal component analysis (PCA) and information theoretical based such as gain ratio. A cluster based technique called MeanDiff is also used to analyse discrimination power of each feature.

From the remaining 454 features, we select the best 168 features (to maintain around 95% of variances) by means of principal component analysis (PCA), Chi-square test, gain ratio computation, and our own MeanDiff method. Before applying models it uses "case boosting" to reduce the skewness of pass vs fail cases.

Four methods are used to induce the fault-detection model namely decision tree, naïve Bayes, logistic regression, and k-nearest neighbour. For assessing the model performance four metrics are used: true positive rate (TP rate or recall), precision, F-measure, and false positive rate (FP rate or false alarm). We want the model that shows the highest values of TP rate, precision, and F-measure, but the lowest value in FP rate. WEKA software was used to perform a series of experiments.

The proposed MeanDiff method contributes the most to decision tree model, whereas the gain ratio method is the best feature selection method for the naive Bayes and logistic regression model building approaches. The k-nearest neighbour method (in which k was set to be one on our experiments because it yields the best result) needs a cleaned dataset without any other feature selection facility. Among the four model building methods, naïve Bayes model can detect fault cases at the success rate as high as 90%, but the false alarm (FP rate) is also as high as 80% as well.

Research Paper 3

Support vector machine is a well-known technique in the field of machine learning which is used for classification. Implementing nonlinear kernels in the SVM structure enables classification of nonlinear data which cannot be classified by simple linear classifiers. SVM algorithm is usually used for twoclass separation problems. In K-nearest neighbour classification, the class of each sample point is determined by its K neighbouring points in the training set. The point is assigned to the class with the majority of votes for class label amongst the K-neighbour points. The classifier is defined by its parameters. Setting parameter K depends on the data and effects the performance of the classifier. K must be large enough to reduce misclassification.

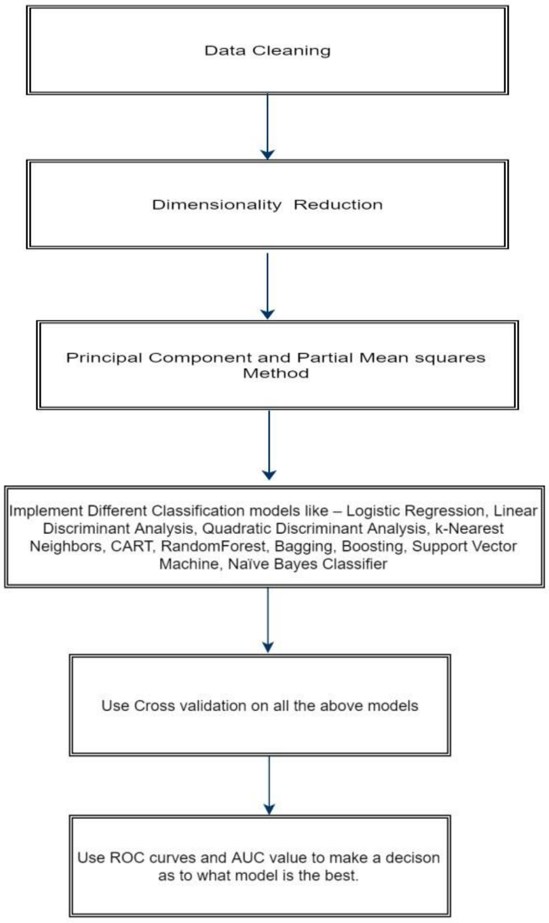
In this paper, the experiment data was conducted on the well-established Tennessee Eastman Process and Three tank system data sets. In every fault detection and diagnosis system, the FDI process includes detecting the fault in the process and then identifying the type of the fault. Here, the focus was on the diagnosis part of the FDI process and assume that fault detection has been accomplished. After fault detection stage in FDI, SVM was used for fault classification. The performance of the C-SVM compared to simple SVM with different kernels and also to K-nearest neighbour classifier is compared. Hence, a training and a testing data set is collected from the processes. The choices of different SVM depend on their parameters. Type of the kernel, value of C, width of the RBF kernel, polynomial kernel degree, and number of SVM to be used in the committee are such example parameters. Since there are many different combinations to choose, they only restricted the experiment to a simple case with three different kernels to be used in the SVM-classifier.

The procedure for building the classifier is as follows: For every two faults they trained a C-SVM classifier. Each classifier is a combination of three SVM with different kernels (linear, RBF, polynomial), trained with data that are a mixture of the two fault class data set. The output is simply the average of the three. It can be clearly seen that using a classifier that is a combination of all the three proves to be the best in this case. However, for further confirmation, the method should be tested on other different processes in order to achieve a comprehensive understanding of the proposed method.

Research Paper 4

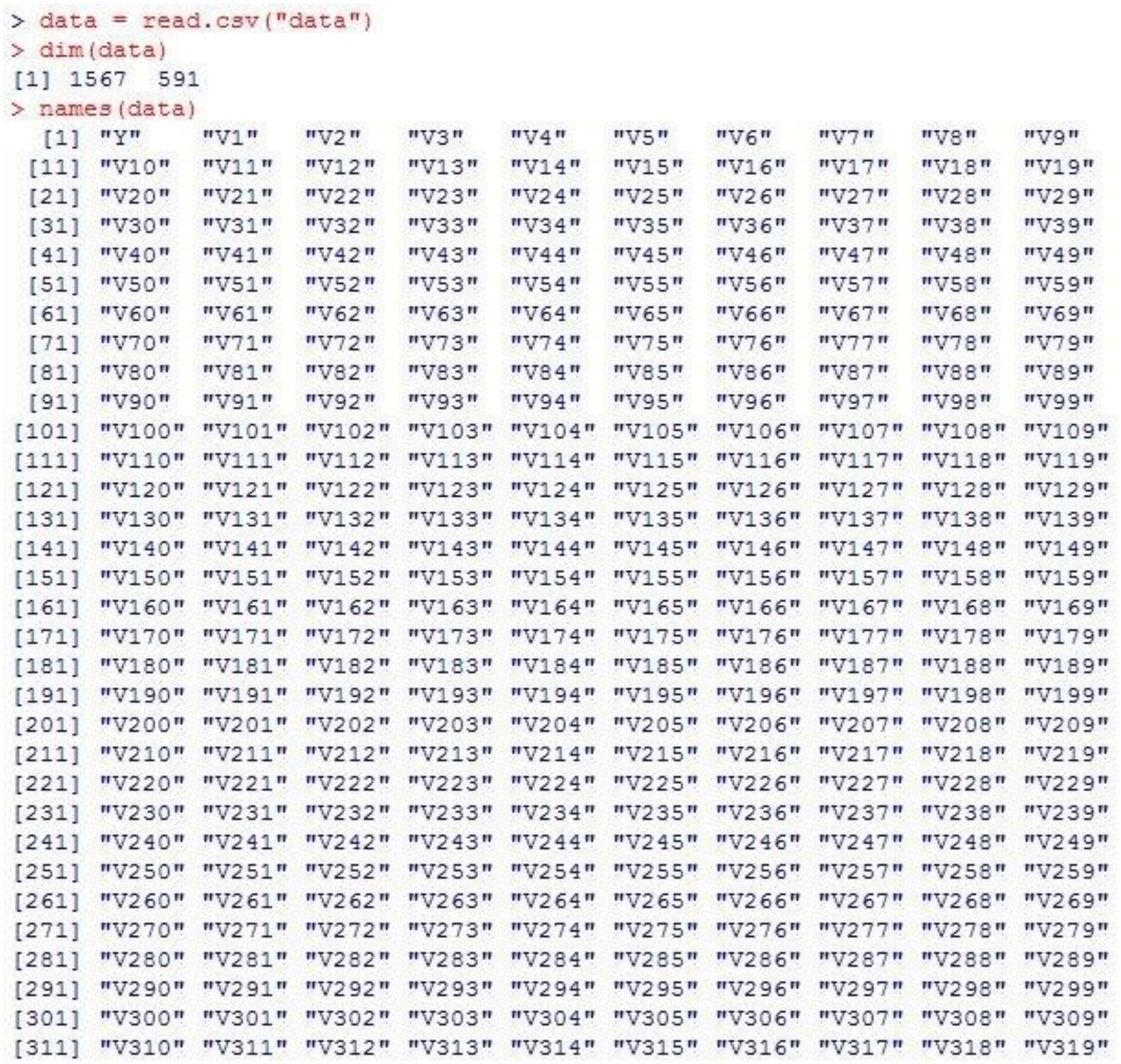
In this paper, a fault detection method has been proposed by combining random projection and k nearest neighbour rule. The proposed method not only can reduce the computational complexity and storage space, but also approximately guarantee the advantages of kNN rule in dealing with the problems of multimode batch trajectories and nonlinearity that often coexist in semiconductor processes. It is worth noting that the idea of using random projection for reducing computation of kNN classification is not new. However, the idea of combining random projection with kNN for fault detection (one-class classification) of semiconductor processes, to our knowledge, should have no publication before. The proposed method should be seen as an alternative fault detection method. It is not superior to PCA in all cases. Some interesting future work: To exploit the ability of kNN for isolating faulty variables and estimating the fault magnitude. Once a fault is detected, the next step is to identify faulty variables which are attributed to this fault. One can also define the variable contribution for the kNN distance in the light of the idea of contribution plots. And the estimation of fault magnitude is useful for fault tolerant control. The distances from normal samples are, to a certain extent related with fault magnitude. To apply RPkNN to other batch or continuous processes. The proposed method is not limited to semiconductor processes, it can also be applied to other high dimensional batch processes and continuous processes.

APPROACH


Data Cleaning
Dimensionality Reduction
Principal Component and Partial Mean squares Method
Implement Different Classification models like – Logistic Regression, Linear Discriminant Analysis, Quadratic Discriminant Analysis, k-Nearest Neighbors, CART, RandomForest, Bagging, Boosting, Support Vector Machine, Naïve Bayes Classifier
Use Cross validation on all the above models
Use ROC curves and AUC value to make a decison as to what model is the best.

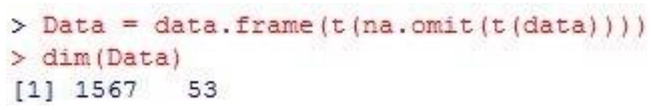
IMPLEMENTATION

The dataset is loaded onto R. The dataset has 1567 observations and 591 variables.



```
> data = read.csv("data")
> dim(data)
[1] 1567  591
> names(data)
  [1] "Y"    "V1"   "V2"   "V3"   "V4"   "V5"   "V6"   "V7"   "V8"   "V9"  
 [11] "V10"  "V11"  "V12"  "V13"  "V14"  "V15"  "V16"  "V17"  "V18"  "V19" 
 [21] "V20"  "V21"  "V22"  "V23"  "V24"  "V25"  "V26"  "V27"  "V28"  "V29" 
 [31] "V30"  "V31"  "V32"  "V33"  "V34"  "V35"  "V36"  "V37"  "V38"  "V39" 
 [41] "V40"  "V41"  "V42"  "V43"  "V44"  "V45"  "V46"  "V47"  "V48"  "V49" 
 [51] "V50"  "V51"  "V52"  "V53"  "V54"  "V55"  "V56"  "V57"  "V58"  "V59" 
 [61] "V60"  "V61"  "V62"  "V63"  "V64"  "V65"  "V66"  "V67"  "V68"  "V69" 
 [71] "V70"  "V71"  "V72"  "V73"  "V74"  "V75"  "V76"  "V77"  "V78"  "V79" 
 [81] "V80"  "V81"  "V82"  "V83"  "V84"  "V85"  "V86"  "V87"  "V88"  "V89" 
 [91] "V90"  "V91"  "V92"  "V93"  "V94"  "V95"  "V96"  "V97"  "V98"  "V99" 
[101] "V100" "V101" "V102" "V103" "V104" "V105" "V106" "V107" "V108" "V109"
[111] "V110" "V111" "V112" "V113" "V114" "V115" "V116" "V117" "V118" "V119"
[121] "V120" "V121" "V122" "V123" "V124" "V125" "V126" "V127" "V128" "V129"
[131] "V130" "V131" "V132" "V133" "V134" "V135" "V136" "V137" "V138" "V139"
[141] "V140" "V141" "V142" "V143" "V144" "V145" "V146" "V147" "V148" "V149"
[151] "V150" "V151" "V152" "V153" "V154" "V155" "V156" "V157" "V158" "V159"
[161] "V160" "V161" "V162" "V163" "V164" "V165" "V166" "V167" "V168" "V169"
[171] "V170" "V171" "V172" "V173" "V174" "V175" "V176" "V177" "V178" "V179"
[181] "V180" "V181" "V182" "V183" "V184" "V185" "V186" "V187" "V188" "V189"
[191] "V190" "V191" "V192" "V193" "V194" "V195" "V196" "V197" "V198" "V199"
[201] "V200" "V201" "V202" "V203" "V204" "V205" "V206" "V207" "V208" "V209"
[211] "V210" "V211" "V212" "V213" "V214" "V215" "V216" "V217" "V218" "V219"
[221] "V220" "V221" "V222" "V223" "V224" "V225" "V226" "V227" "V228" "V229"
[231] "V230" "V231" "V232" "V233" "V234" "V235" "V236" "V237" "V238" "V239"
[241] "V240" "V241" "V242" "V243" "V244" "V245" "V246" "V247" "V248" "V249"
[251] "V250" "V251" "V252" "V253" "V254" "V255" "V256" "V257" "V258" "V259"
[261] "V260" "V261" "V262" "V263" "V264" "V265" "V266" "V267" "V268" "V269"
[271] "V270" "V271" "V272" "V273" "V274" "V275" "V276" "V277" "V278" "V279"
[281] "V280" "V281" "V282" "V283" "V284" "V285" "V286" "V287" "V288" "V289"
[291] "V290" "V291" "V292" "V293" "V294" "V295" "V296" "V297" "V298" "V299"
[301] "V300" "V301" "V302" "V303" "V304" "V305" "V306" "V307" "V308" "V309"
[311] "V310" "V311" "V312" "V313" "V314" "V315" "V316" "V317" "V318" "V319"
```

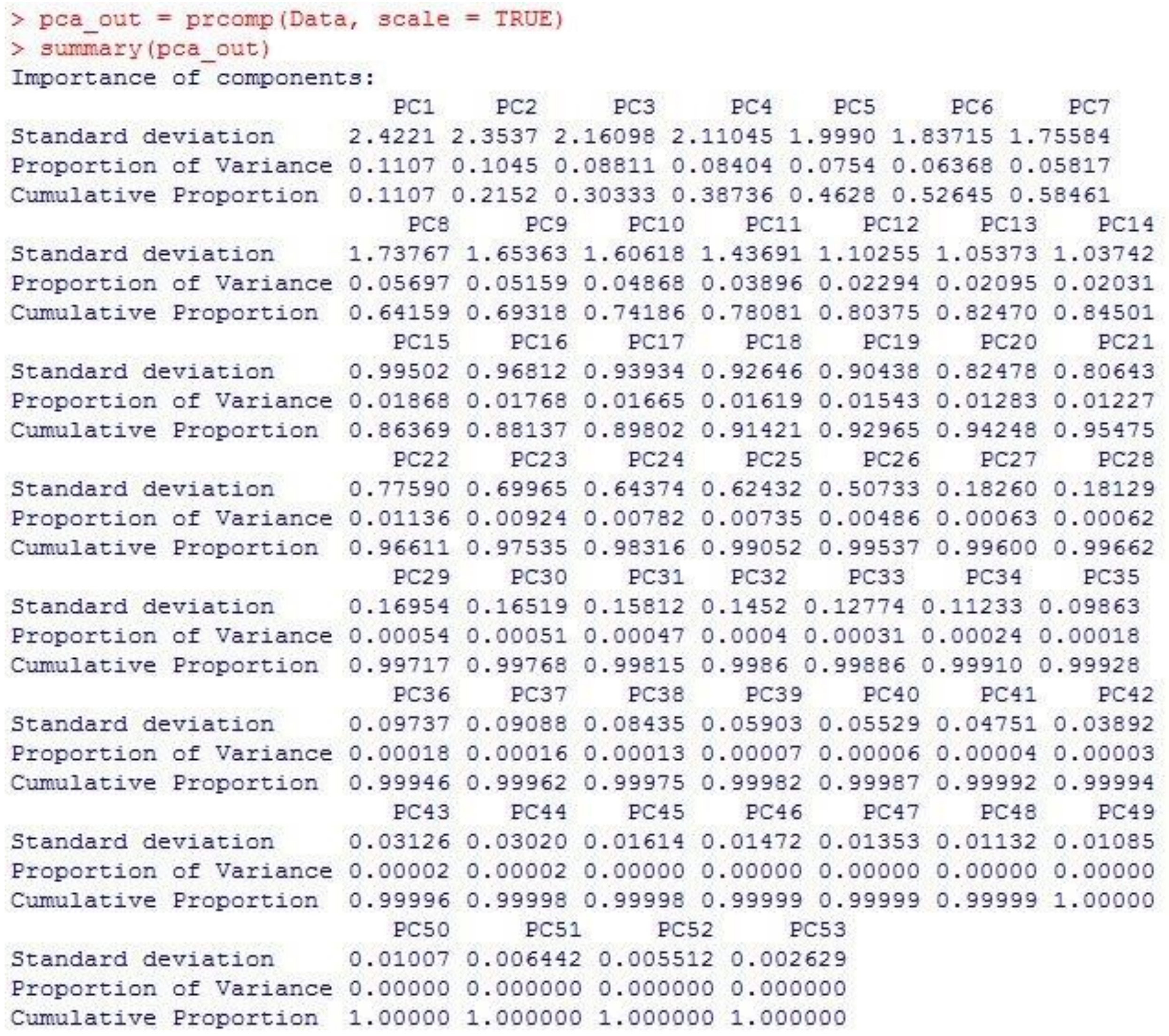
We could observe that there are missing values present in the dataset. Using the below code, we identified the names of the variables containing the missing values and omitted them. We transformed those values to a new set of constant values. The columns which contains these constants were omitted, as the variance of those constants is always zero. The new data obtained has a total of 53 variables as shown below.



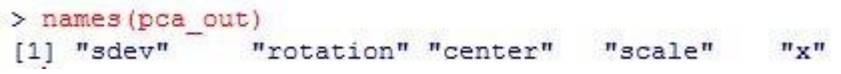
```
> Data = data.frame(t(na.omit(t(data))))
> dim(Data)
[1] 1567   53
```

Dimensionality Reduction:

We employed the process of Principal Component Analysis to reduce the dimensionality of the dataset. The summary of the code obtained is as shown below.

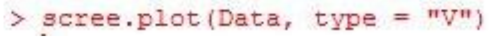


```
> pca_out = prcomp(Data, scale = TRUE)
> summary(pca_out)
Importance of components:
                          PC1    PC2     PC3     PC4    PC5     PC6     PC7
Standard deviation     2.4221 2.3537 2.16098 2.11045 1.9990 1.83715 1.75584
Proportion of Variance 0.1107 0.1045 0.08811 0.08404 0.0754 0.06368 0.05817
Cumulative Proportion  0.1107 0.2152 0.30333 0.38736 0.4628 0.52645 0.58461
                           PC8     PC9    PC10    PC11    PC12    PC13    PC14
Standard deviation     1.73767 1.65363 1.60618 1.43691 1.10255 1.05373 1.03742
Proportion of Variance 0.05697 0.05159 0.04868 0.03896 0.02294 0.02095 0.02031
Cumulative Proportion  0.64159 0.69318 0.74186 0.78081 0.80375 0.82470 0.84501
                          PC15    PC16    PC17    PC18    PC19    PC20    PC21
Standard deviation     0.99502 0.96812 0.93934 0.92646 0.90438 0.82478 0.80643
Proportion of Variance 0.01868 0.01768 0.01665 0.01619 0.01543 0.01283 0.01227
Cumulative Proportion  0.86369 0.88137 0.89802 0.91421 0.92965 0.94248 0.95475
                          PC22    PC23    PC24    PC25    PC26    PC27    PC28
Standard deviation     0.77590 0.69965 0.64374 0.62432 0.50733 0.18260 0.18129
Proportion of Variance 0.01136 0.00924 0.00782 0.00735 0.00486 0.00063 0.00062
Cumulative Proportion  0.96611 0.97535 0.98316 0.99052 0.99537 0.99600 0.99662
                          PC29    PC30    PC31   PC32    PC33    PC34    PC35
Standard deviation     0.16954 0.16519 0.15812 0.1452 0.12774 0.11233 0.09863
Proportion of Variance 0.00054 0.00051 0.00047 0.0004 0.00031 0.00024 0.00018
Cumulative Proportion  0.99717 0.99768 0.99815 0.9986 0.99886 0.99910 0.99928
                          PC36    PC37    PC38    PC39    PC40    PC41    PC42
Standard deviation     0.09737 0.09088 0.08435 0.05903 0.05529 0.04751 0.03892
Proportion of Variance 0.00018 0.00016 0.00013 0.00007 0.00006 0.00004 0.00003
Cumulative Proportion  0.99946 0.99962 0.99975 0.99982 0.99987 0.99992 0.99994
                          PC43    PC44    PC45    PC46    PC47    PC48    PC49
Standard deviation     0.03126 0.03020 0.01614 0.01472 0.01353 0.01132 0.01085
Proportion of Variance 0.00002 0.00002 0.00000 0.00000 0.00000 0.00000 0.00000
Cumulative Proportion  0.99996 0.99998 0.99998 0.99999 0.99999 0.99999 1.00000
                          PC50     PC51     PC52     PC53
Standard deviation     0.01007 0.006442 0.005512 0.002629
Proportion of Variance 0.00000 0.000000 0.000000 0.000000
Cumulative Proportion  1.00000 1.000000 1.000000 1.000000
```

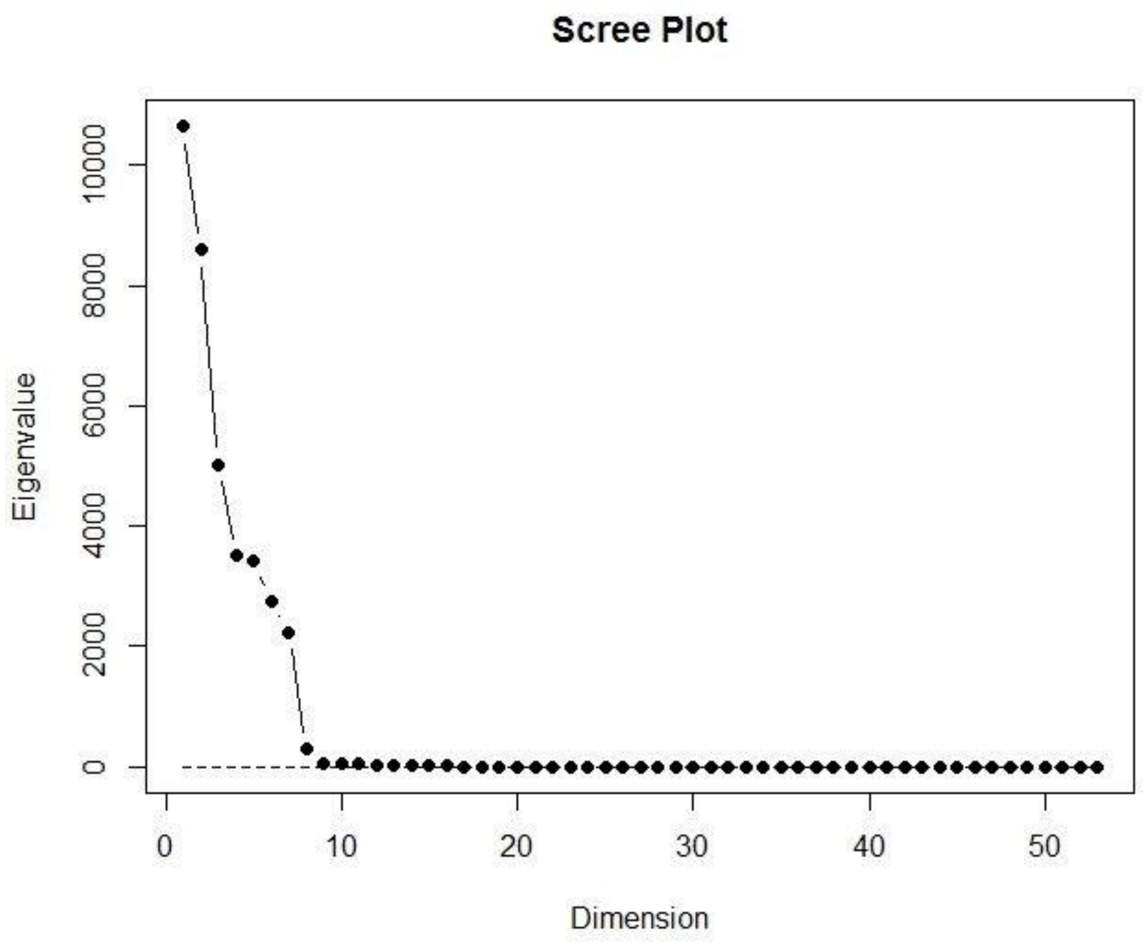


```
> names(pca_out)
[1] "sdev"     "rotation" "center"   "scale"    "x"
```

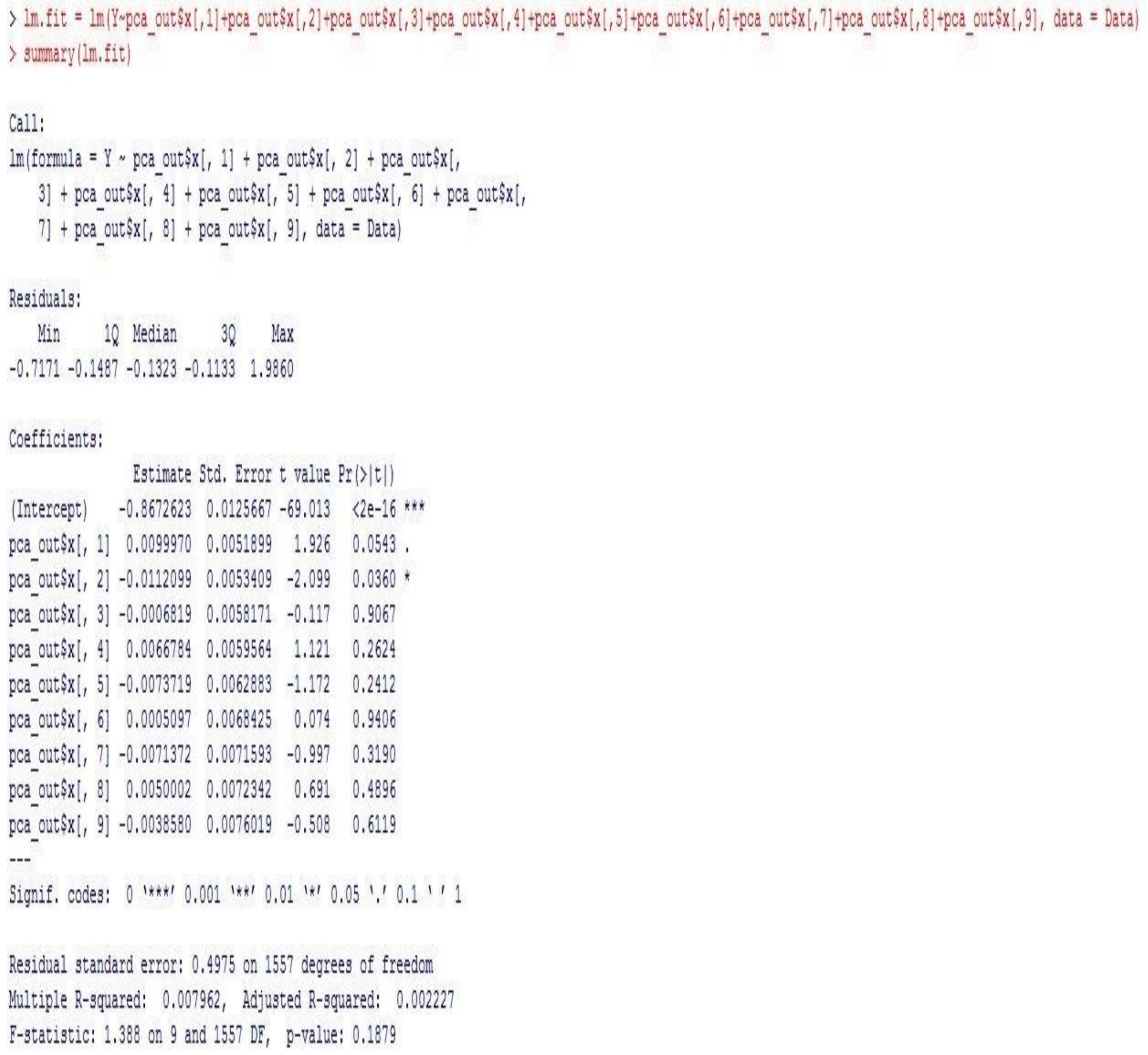
Center corresponds to the mean, sdev gives the standard deviation and rotation corresponds to the directional cosines of the respective principal components.
We then employ the method of “Scree Plot” to find out the optimal number of principal components.



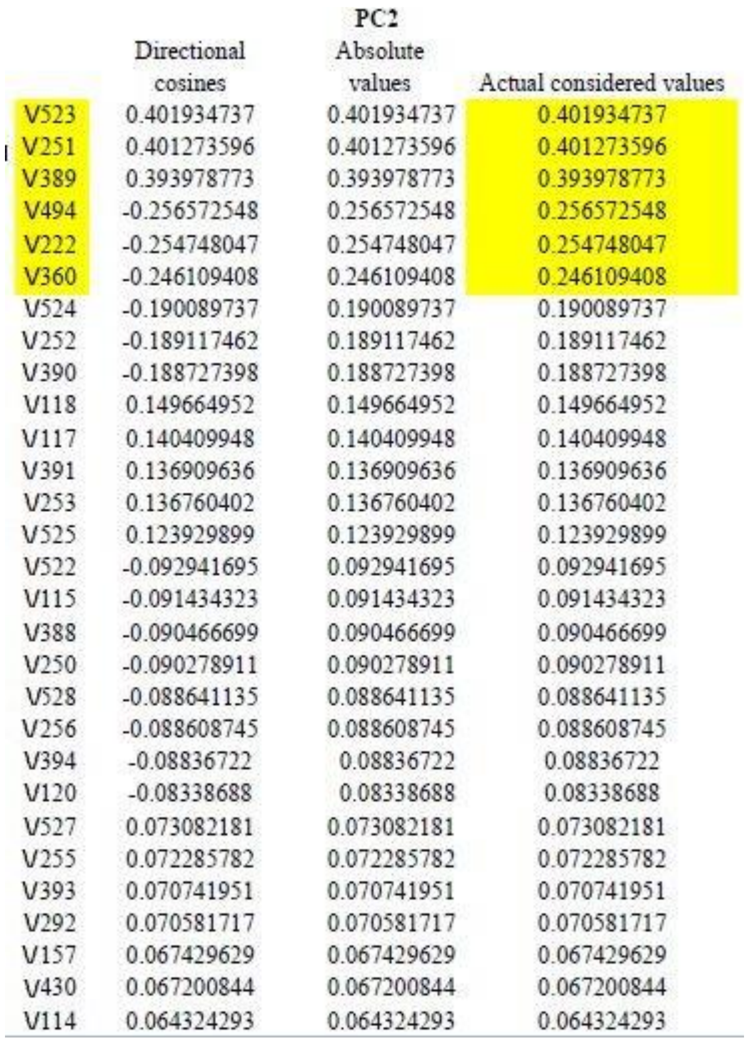
```
> scree.plot(Data, type = "V")
```



We can infer from the Scree plot that the optimal number of principal components required are 9.
When a linear regression model was fitted onto the above obtained PCA model, we get the following results:



```
> lm.fit = lm(Y~pca_out$x[,1]+pca_out$x[,2]+pca_out$x[,3]+pca_out$x[,4]+pca_out$x[,5]+pca_out$x[,6]+pca_out$x[,7]+pca_out$x[,8]+pca_out$x[,9], data = Data)
> summary(lm.fit)

Call:
lm(formula = Y ~ pca_out$x[, 1] + pca_out$x[, 2] + pca_out$x[,
    3] + pca_out$x[, 4] + pca_out$x[, 5] + pca_out$x[, 6] + pca_out$x[,
    7] + pca_out$x[, 8] + pca_out$x[, 9], data = Data)

Residuals:
    Min      1Q  Median      3Q     Max
-0.7171 -0.1487 -0.1323 -0.1133  1.9860

Coefficients:
                 Estimate Std. Error t value Pr(>|t|)
(Intercept)    -0.8672623  0.0125667 -69.013   <2e-16 ***
pca_out$x[, 1]  0.0099970  0.0051899   1.926   0.0543 .
pca_out$x[, 2] -0.0112099  0.0053409  -2.099   0.0360 *
pca_out$x[, 3] -0.0006819  0.0058171  -0.117   0.9067
pca_out$x[, 4]  0.0066784  0.0059564   1.121   0.2624
pca_out$x[, 5] -0.0073719  0.0062883  -1.172   0.2412
pca_out$x[, 6]  0.0005097  0.0068425   0.074   0.9406
pca_out$x[, 7] -0.0071372  0.0071593  -0.997   0.3190
pca_out$x[, 8]  0.0050002  0.0072342   0.691   0.4896
pca_out$x[, 9] -0.0038580  0.0076019  -0.508   0.6119
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.4975 on 1557 degrees of freedom
Multiple R-squared:  0.007962,	Adjusted R-squared:  0.002227
F-statistic: 1.388 on 9 and 1557 DF,  p-value: 0.1879
```



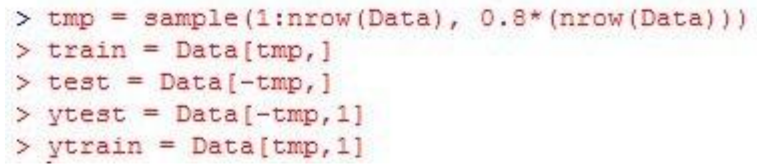
	PC2 Directional cosines	Absolute values	Actual considered values
V523	0.401934737	0.401934737	0.401934737
V251	0.401273596	0.401273596	0.401273596
V389	0.393978773	0.393978773	0.393978773
V494	-0.256572548	0.256572548	0.256572548
V222	-0.254748047	0.254748047	0.254748047
V360	-0.246109408	0.246109408	0.246109408
V524	-0.190089737	0.190089737	0.190089737
V252	-0.189117462	0.189117462	0.189117462
V390	-0.188727398	0.188727398	0.188727398
V118	0.149664952	0.149664952	0.149664952
V117	0.140409948	0.140409948	0.140409948
V391	0.136909636	0.136909636	0.136909636
V253	0.136760402	0.136760402	0.136760402
V525	0.123929899	0.123929899	0.123929899
V522	-0.092941695	0.092941695	0.092941695
V115	-0.091434323	0.091434323	0.091434323
V388	-0.090466699	0.090466699	0.090466699
V250	-0.090278911	0.090278911	0.090278911
V528	-0.088641135	0.088641135	0.088641135
V256	-0.088608745	0.088608745	0.088608745
V394	-0.08836722	0.08836722	0.08836722
V120	-0.08338688	0.08338688	0.08338688
V527	0.073082181	0.073082181	0.073082181
V255	0.072285782	0.072285782	0.072285782
V393	0.070741951	0.070741951	0.070741951
V292	0.070581717	0.070581717	0.070581717
V157	0.067429629	0.067429629	0.067429629
V430	0.067200844	0.067200844	0.067200844
V114	0.064324293	0.064324293	0.064324293

We can conclude that 2nd Principal component is the most significant of all the principal components.

With further analysis, we found out that the important variables to be considered are : V523, V251, V389, V494, V222, and V360. We will be building our further models by taking different combinations of these variables to obtain best prediction performance and to also study the features responsible for the failure of the semiconductor devices.

Splitting into Training and Testing datasets

The entire dataset is split into a separate training and test data. 80% of the observations are termed as the training data and the remaining 20% is the test data using the following code.

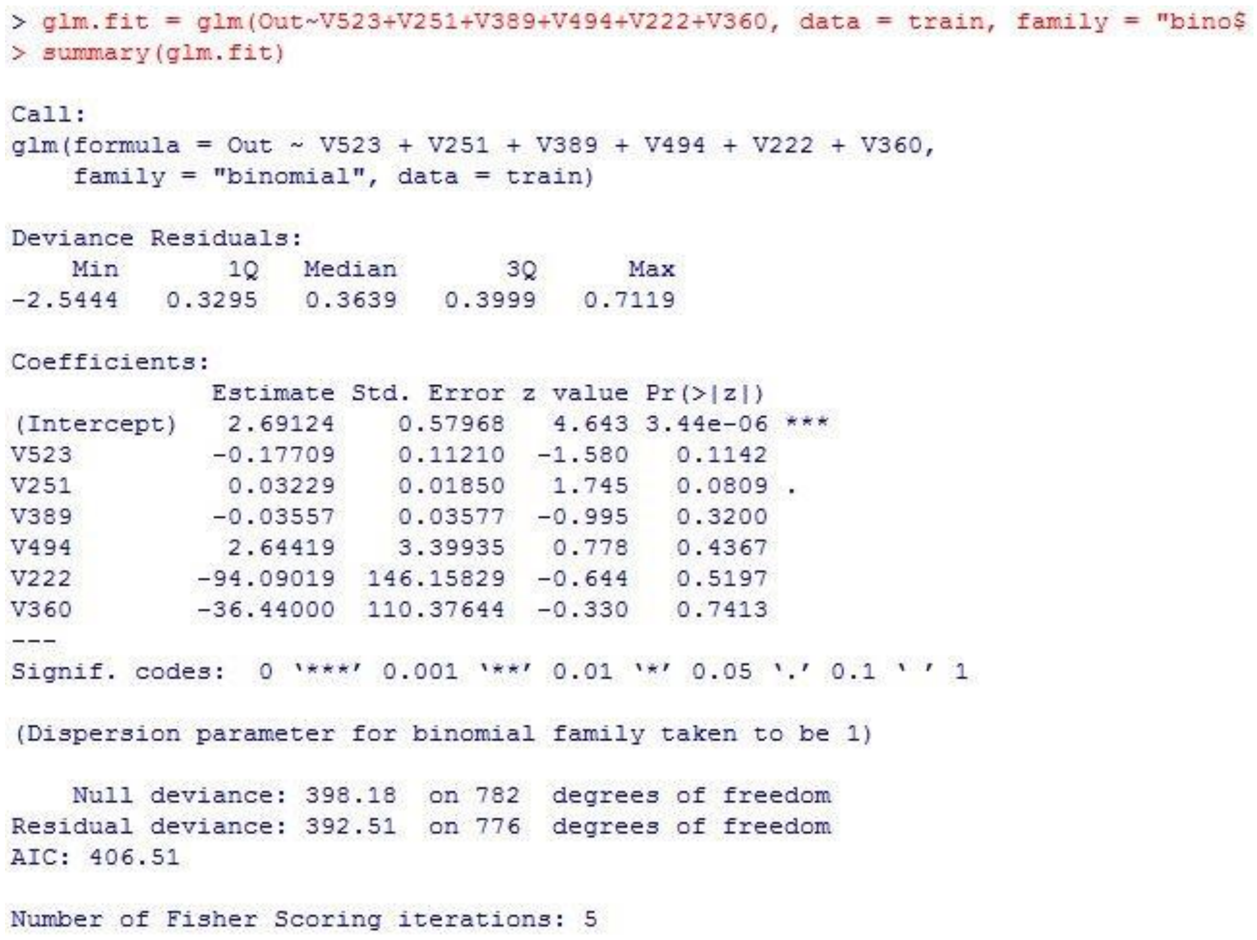


```
> tmp = sample(1:nrow(Data), 0.8*(nrow(Data)))
> train = Data[tmp,]
> test = Data[-tmp,]
> ytest = Data[-tmp,1]
> ytrain = Data[tmp,1]
```

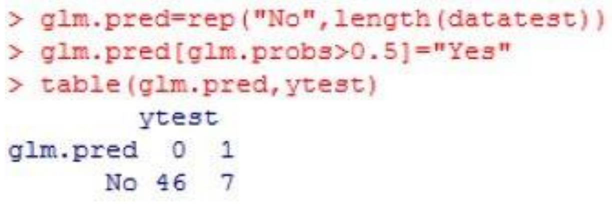
CLASSIFICATION MODELS

1. Logistic Regression

The method of logistic regression is done to all the important set of predictors. After several iterations, we get the best model with a combination of predictors: V523,V251,V389,V494,V222,V360. The summary of the output of the model is as shown below.

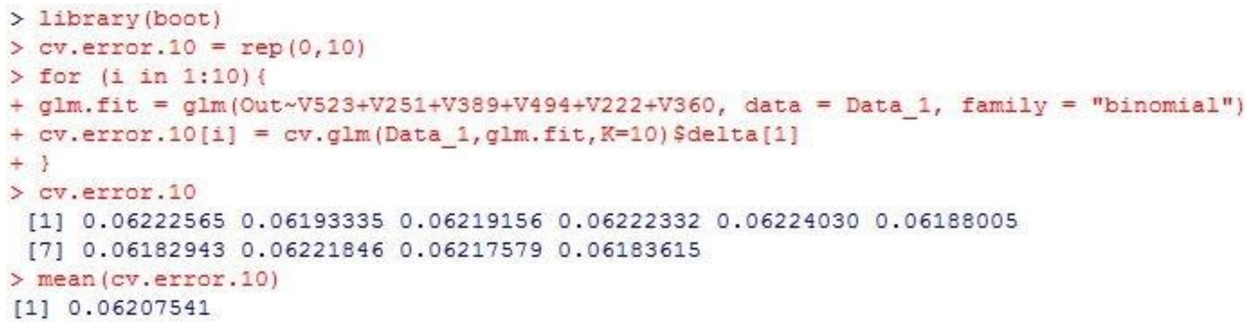


```
> glm.fit = glm(Out~V523+V251+V389+V494+V222+V360, data = train, family = "bino$
> summary(glm.fit)

Call:
glm(formula = Out ~ V523 + V251 + V389 + V494 + V222 + V360,
    family = "binomial", data = train)

Deviance Residuals:
    Min       1Q   Median       3Q      Max
-2.5444   0.3295   0.3639   0.3999   0.7119

Coefficients:
             Estimate Std. Error z value Pr(>|z|)
(Intercept)   2.69124    0.57968   4.643 3.44e-06 ***
V523         -0.17709    0.11210  -1.580   0.1142
V251          0.03229    0.01850   1.745   0.0809 .
V389         -0.03557    0.03577  -0.995   0.3200
V494          2.64419    3.39935   0.778   0.4367
V222        -94.09019  146.15829  -0.644   0.5197
V360        -36.44000  110.37644  -0.330   0.7413
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 398.18  on 782  degrees of freedom
Residual deviance: 392.51  on 776  degrees of freedom
AIC: 406.51

Number of Fisher Scoring iterations: 5
```



```
> glm.pred=rep("No",length(datatest))
> glm.pred[glm.probs>0.5]="Yes"
> table(glm.pred,ytest)
        ytest
glm.pred  0  1
      No 46  7
```

Cross Validation:



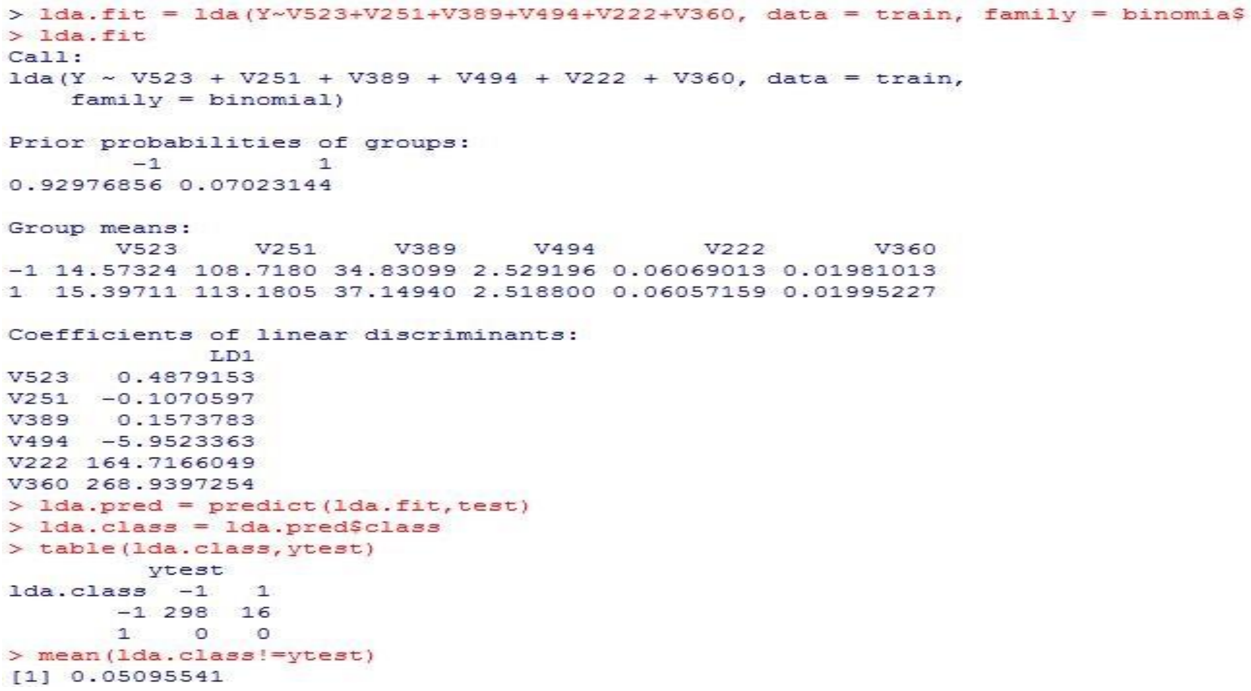
```
> library(boot)
> cv.error.10 = rep(0,10)
> for (i in 1:10){
+ glm.fit = glm(Out~V523+V251+V389+V494+V222+V360, data = Data_1, family = "binomial")
+ cv.error.10[i] = cv.glm(Data_1,glm.fit,K=10)$delta[1]
+ }
> cv.error.10
 [1] 0.06222565 0.06193335 0.06219156 0.06222332 0.06224030 0.06188005
 [7] 0.06182943 0.06221846 0.06217579 0.06183615
> mean(cv.error.10)
[1] 0.06207541
```

The error rate after cross validation, is 6.2075%

Predictors used	Error Rate
V523,V251,V389,V494,V222,V360	6.2075%

2. Linear Discriminant Analysis(LDA)

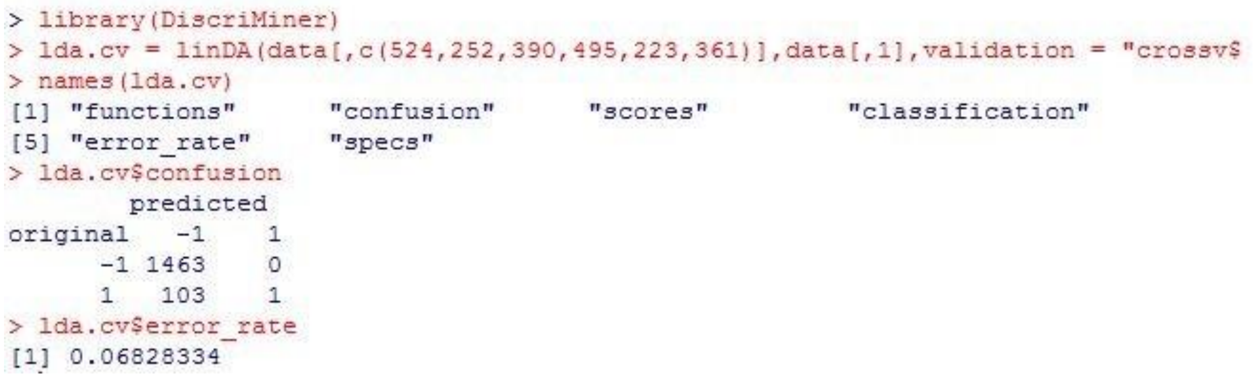
From the main predictors chosen, we found that the best model possible from the method of LDA is when we use the predictors: V523,V251,V389,V494,V222,V360.
The summary of the model obtained is as shown below.



```
> lda.fit = lda(Y~V523+V251+V389+V494+V222+V360, data = train, family = binomia$
> lda.fit
Call:
lda(Y ~ V523 + V251 + V389 + V494 + V222 + V360, data = train,
    family = binomial)

Prior probabilities of groups:
        -1          1
0.92976856 0.07023144

Group means:
       V523     V251     V389     V494       V222       V360
-1 14.57324 108.7180 34.83099 2.529196 0.06069013 0.01981013
1  15.39711 113.1805 37.14940 2.518800 0.06057159 0.01995227

Coefficients of linear discriminants:
             LD1
V523   0.4879153
V251  -0.1070597
V389   0.1573783
V494  -5.9523363
V222 164.7166049
V360 268.9397254
> lda.pred = predict(lda.fit,test)
> lda.class = lda.pred$class
> table(lda.class,ytest)
         ytest
lda.class  -1   1
       -1 298  16
       1    0   0
> mean(lda.class!=ytest)
[1] 0.05095541
```

The classification error rate is 5.0955%.

Cross Validation:



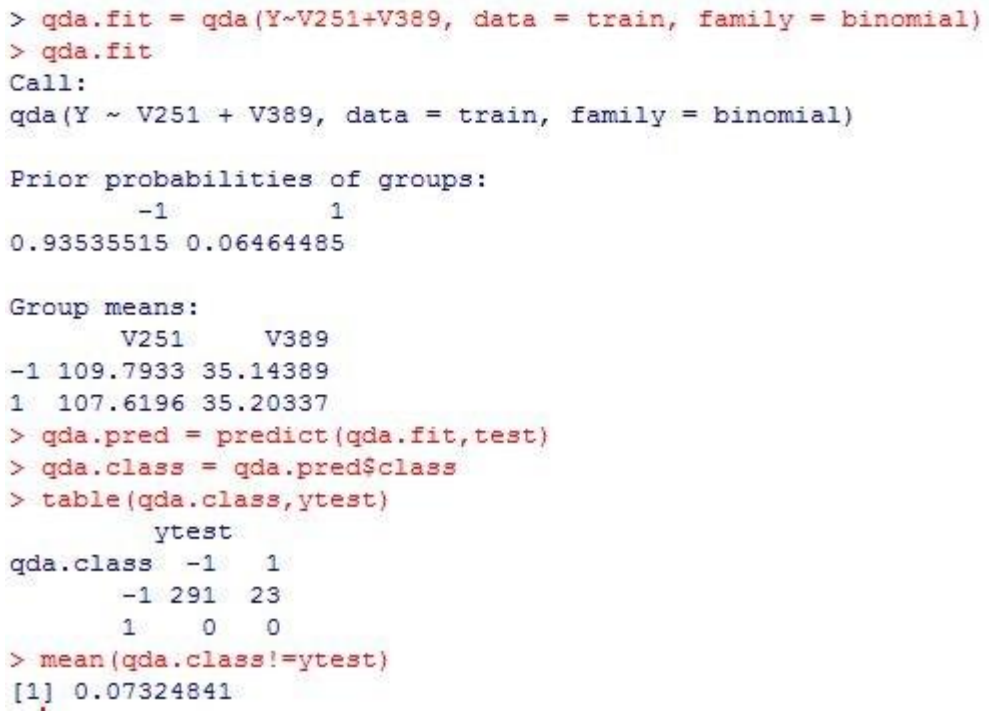
```
> library(DiscriMiner)
> lda.cv = linDA(data[,c(524,252,390,495,223,361)],data[,1],validation = "crossv$
> names(lda.cv)
[1] "functions"      "confusion"      "scores"         "classification"
[5] "error_rate"     "specs"
> lda.cv$confusion
        predicted
original   -1    1
      -1 1463    0
      1   103    1
> lda.cv$error_rate
[1] 0.06828334
```

After cross validation, the classification error rate is 6.8283%.

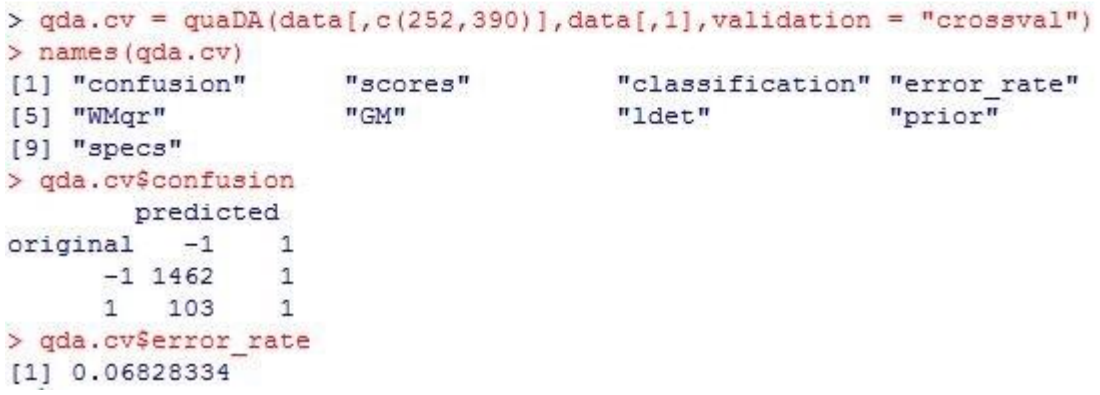
Predictors used	Error Rate	CV Error Rate
V523,V251,V389,V494,V222,V360	5.0955%	6.8283%

3. Quadratic Discriminant Analysis

From the main predictors chosen, we found that the best model possible from the method of LDA is when we use the predictors: V251,V389.
The summary of the model obtained is as shown below.


```
> qda.fit = qda(Y~V251+V389, data = train, family = binomial)
> qda.fit
Call:
qda(Y ~ V251 + V389, data = train, family = binomial)

Prior probabilities of groups:
        -1          1
0.93535515 0.06464485

Group means:
       V251     V389
-1 109.7933 35.14389
1  107.6196 35.20337
> qda.pred = predict(qda.fit,test)
> qda.class = qda.pred$class
> table(qda.class,ytest)
         ytest
qda.class  -1   1
       -1 291  23
       1    0   0
> mean(qda.class!=ytest)
[1] 0.07324841
```
The classification error rate is 7.3248%.

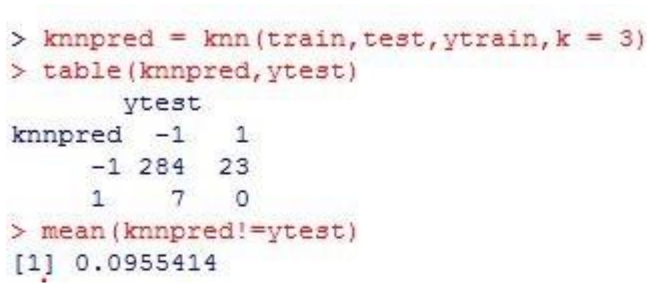
Cross validation:


```
> qda.cv = quaDA(data[,c(252,390)],data[,1],validation = "crossval")
> names(qda.cv)
[1] "confusion"      "scores"         "classification" "error_rate"
[5] "WMqr"           "GM"             "ldet"           "prior"
[9] "specs"
> qda.cv$confusion
        predicted
original   -1    1
      -1 1462    1
      1   103    1
> qda.cv$error_rate
[1] 0.06828334
```
The cross validated classification error rate is 6.8283%.

Predictors used	**Error Rate**	**CV Error Rate**
V251,V389	7.3248%	6.8283%

4. K-Nearest-Neighbour (KNN) Method

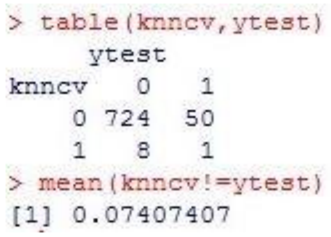
An iterative procedure is done for the following cases: k=1, k=2 and k=3. We find that the best model is obtained for the case when k=3.



```
> knnpred = knn(train,test,ytrain,k = 3)
> table(knnpred,ytest)
        ytest
knnpred  -1   1
     -1 284  23
      1   7   0
> mean(knnpred!=ytest)
[1] 0.0955414
```

The classification error rate obtained is 9.5541%.

Cross validation:



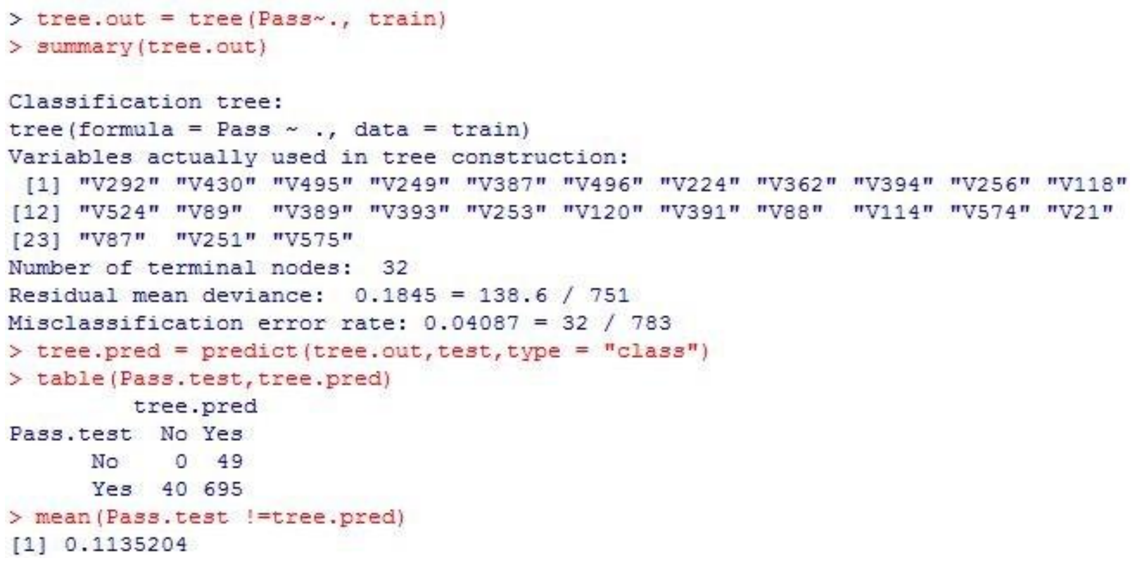
```
> table(knncv,ytest)
     ytest
knncv   0   1
    0 724  50
    1   8   1
> mean(knncv!=ytest)
[1] 0.07407407
```

The cross validated classification error rate is 7.4074%.

Error Rate	**CV Error Rate**
9.5541%	**7.4074%**

5. Classification and Regression Tree (CART)

Tree - based models are popular for their interpretability and hence we are not just considering the variables which are the highly significant from the principal component analysis but are going to consider all the variables present in the data set. The summary of the model obtained is as shown below and the variables considered for the tree are shown in the summary. We can interpret the model as if the value of the variable V292, which is the most significant one is less than 0.03515 and V430 is less than 4.87605, the semiconductor device is more likely to fail. Feature selection for process improvement activities could be obtained from tree - based models.

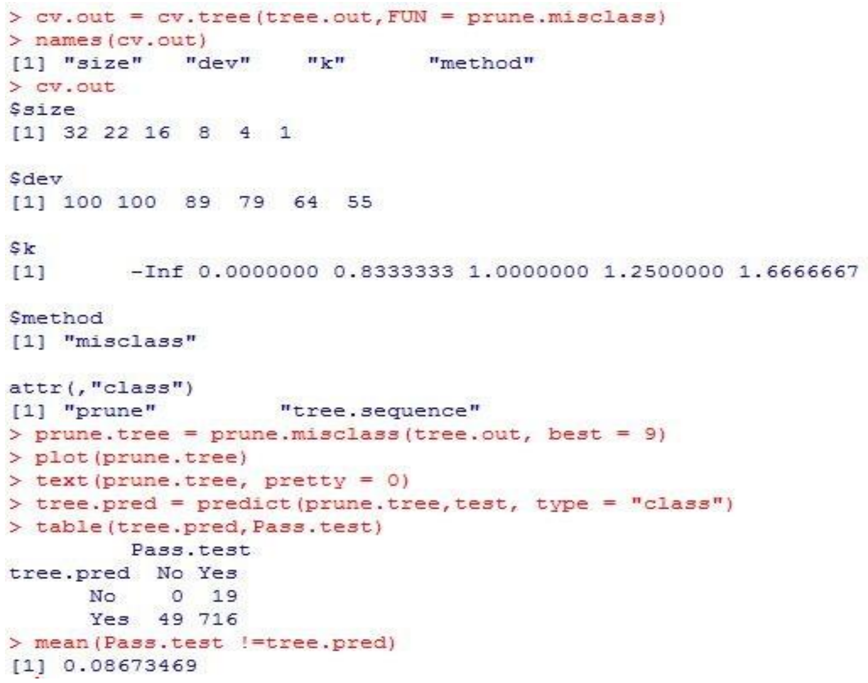


```
> tree.out = tree(Pass~., train)
> summary(tree.out)

Classification tree:
tree(formula = Pass ~ ., data = train)
Variables actually used in tree construction:
 [1] "V292" "V430" "V495" "V249" "V387" "V496" "V224" "V362" "V394" "V256" "V118"
[12] "V524" "V89"  "V389" "V393" "V253" "V120" "V391" "V88"  "V114" "V574" "V21"
[23] "V87"  "V251" "V575"
Number of terminal nodes:  32
Residual mean deviance:  0.1845 = 138.6 / 751
Misclassification error rate: 0.04087 = 32 / 783
> tree.pred = predict(tree.out,test,type = "class")
> table(Pass.test,tree.pred)
         tree.pred
Pass.test  No Yes
      No    0  49
      Yes  40 695
> mean(Pass.test !=tree.pred)
[1] 0.1135204
```

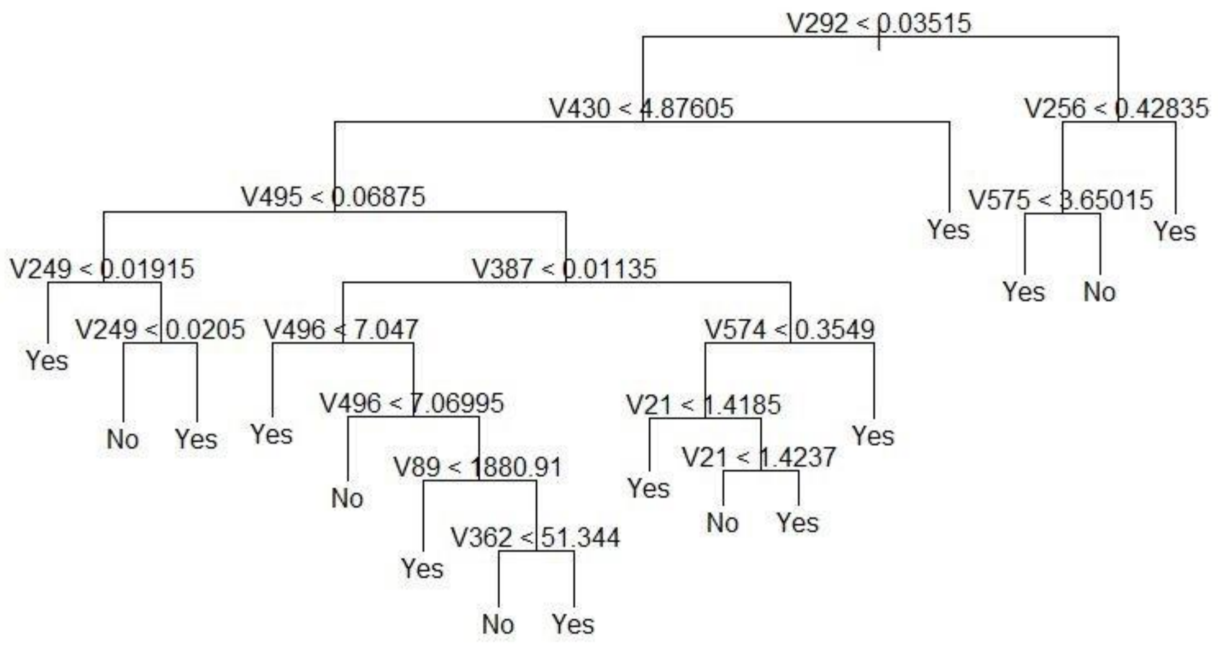
The classification error rate obtained is 11.352%.

Pruning:

Cross - Validation with pruning could be considered for better tree selection.



```
> cv.out = cv.tree(tree.out,FUN = prune.misclass)
> names(cv.out)
[1] "size"   "dev"    "k"      "method"
> cv.out
$size
[1] 32 22 16  8  4  1

$dev
[1] 100 100  89  79  64  55

$k
[1]      -Inf 0.0000000 0.8333333 1.0000000 1.2500000 1.6666667

$method
[1] "misclass"

attr(,"class")
[1] "prune"         "tree.sequence"
> prune.tree = prune.misclass(tree.out, best = 9)
> plot(prune.tree)
> text(prune.tree, pretty = 0)
> tree.pred = predict(prune.tree,test, type = "class")
> table(tree.pred,Pass.test)
         Pass.test
tree.pred  No Yes
      No    0  19
      Yes  49 716
> mean(Pass.test !=tree.pred)
[1] 0.08673469
```

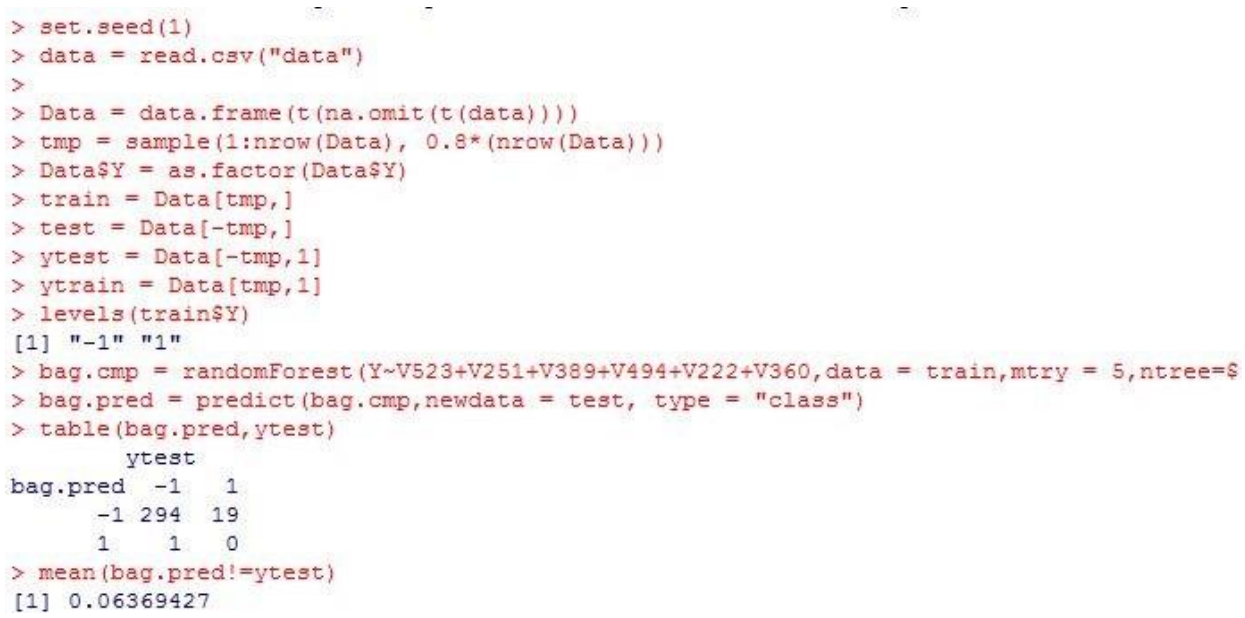


The plot of the pruned tree is as shown in the figure above. The classification error rate is 8.6734%.

Variables used for tree construction	**Error Rate**	**Pruned Tree - CV Error Rate**
V292,V430,V495,V249,V387,V496,V224,V362,V394,V256,V118,V524,V89,V389,V393,V253,V120,V391,V88,V114,V574,V21,V87,V251,V575	11.352%	8.6734%

6. Bagging and Random Forest

After several iterations, the best model with the lowest classification error rate is obtained which is as shown below. The predictors used for this model are V523,V251,V389,V494,V222 and V360.



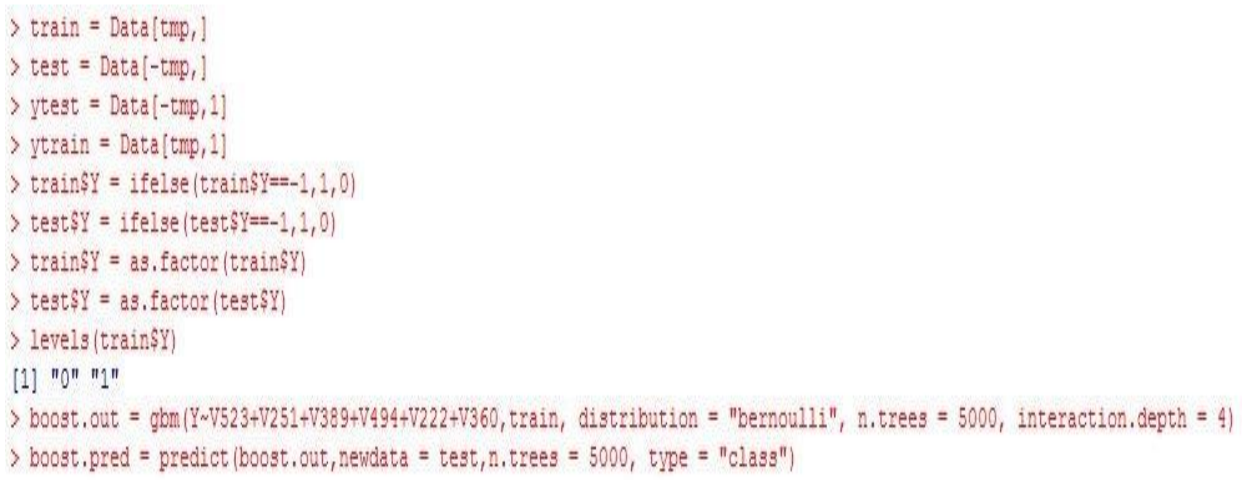
```
> set.seed(1)
> data = read.csv("data")
>
> Data = data.frame(t(na.omit(t(data))))
> tmp = sample(1:nrow(Data), 0.8*(nrow(Data)))
> Data$Y = as.factor(Data$Y)
> train = Data[tmp,]
> test = Data[-tmp,]
> ytest = Data[-tmp,1]
> ytrain = Data[tmp,1]
> levels(train$Y)
[1] "-1" "1"
> bag.cmp = randomForest(Y~V523+V251+V389+V494+V222+V360,data = train,mtry = 5,ntree=$
> bag.pred = predict(bag.cmp,newdata = test, type = "class")
> table(bag.pred,ytest)
        ytest
bag.pred  -1   1
      -1 294  19
      1    1   0
> mean(bag.pred!=ytest)
[1] 0.06369427
```

The classification error rate obtained is 6.3693%.

Predictors used	Error Rate
V523,V251,V389,V494,V222,V360	6.3694%

7. Boosting

Boosting model was studied on the dataset but the model failed to provide good prediction performance which can be seen from the below results. The predictors used for this model are V523,V251,V389,V494,V222,V360.



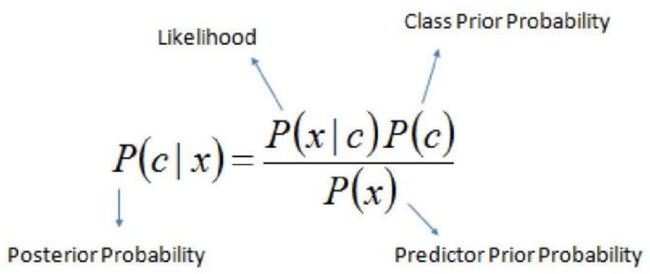
```
> train = Data[tmp,]
> test = Data[-tmp,]
> ytest = Data[-tmp,1]
> ytrain = Data[tmp,1]
> train$Y = ifelse(train$Y==-1,1,0)
> test$Y = ifelse(test$Y==-1,1,0)
> train$Y = as.factor(train$Y)
> test$Y = as.factor(test$Y)
> levels(train$Y)
[1] "0" "1"
> boost.out = gbm(Y~V523+V251+V389+V494+V222+V360,train, distribution = "bernoulli", n.trees = 5000, interaction.depth = 4)
> boost.pred = predict(boost.out,newdata = test,n.trees = 5000, type = "class")
```

The classification error rate obtained is 15.0593%.

Predictors used	Error Rate
V523,V251,V389,V494,V222,V360	15.0593%

8. Naïve Bayes Classifier

The Bayesian Classification represents a supervised learning method as well as a statistical method for classification. Assumes an underlying probabilistic model and it allows us to capture uncertainty about the model in a principled way by determining probabilities of the outcomes. It can solve diagnostic and predictive problems. This Classification is named after Thomas Bayes (1702-1761), who proposed the Bayes Theorem. **Algorithm:**
Naive Bayes classifier understands that the effect of the value of a predictor 'x' on a given class 'c' is independent of the values of other predictors. Bayes theorem provides a way of calculating the posterior probability, P(c|x), from P(c), P(x), and P(x|c).

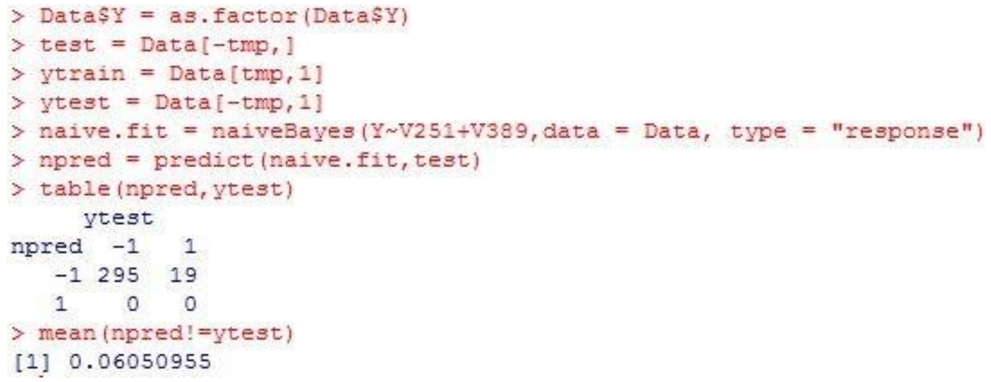

Likelihood
Class Prior Probability
$P(c \mid x) = \frac{P(x \mid c) P(c)}{P(x)}$
Posterior Probability
Predictor Prior Probability

$$P(c \mid X) = P(x_1 \mid c) \times P(x_2 \mid c) \times \cdots \times P(x_n \mid c) \times P(c)$$

Where,

- *P*(*c*|*x*) is the posterior probability of *class* (*target*) given *predictor* (*attribute*).
- *P*(*c*) is the prior probability of *class*.
- *P*(*x*|*c*) is the likelihood which is the probability of *predictor* given *class* .
- *P*(*x*) is the prior probability of *predictor*.

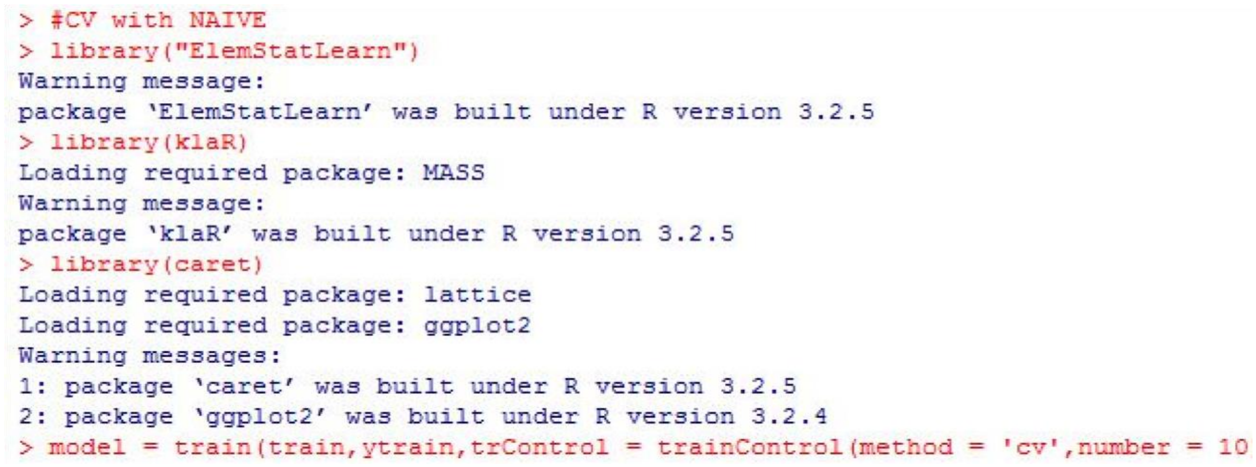
We have tried to incorporate this concept into our model and applied Naive Bayes Classification on our dataset. We studied the classifier using different sets of significant predictors and the best below is selected after the study. We also applied 10 fold cross validation technique with our classifier and studied the prediction performance. The output for the best model with the least misclassification rate is as shown below.



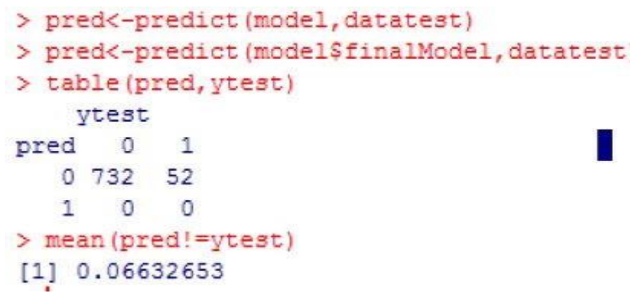
```
> Data$Y = as.factor(Data$Y)
> test = Data[-tmp,]
> ytrain = Data[tmp,1]
> ytest = Data[-tmp,1]
> naive.fit = naiveBayes(Y~V251+V389,data = Data, type = "response")
> npred = predict(naive.fit,test)
> table(npred,ytest)
     ytest
npred  -1   1
   -1 295  19
   1    0   0
> mean(npred!=ytest)
[1] 0.06050955
```

The classification error rate obtained is 6.0509%.

Cross validation:



```
> #CV with NAIVE
> library("ElemStatLearn")
Warning message:
package 'ElemStatLearn' was built under R version 3.2.5
> library(klaR)
Loading required package: MASS
Warning message:
package 'klaR' was built under R version 3.2.5
> library(caret)
Loading required package: lattice
Loading required package: ggplot2
Warning messages:
1: package 'caret' was built under R version 3.2.5
2: package 'ggplot2' was built under R version 3.2.4
> model = train(train,ytrain,trControl = trainControl(method = 'cv',number = 10
```



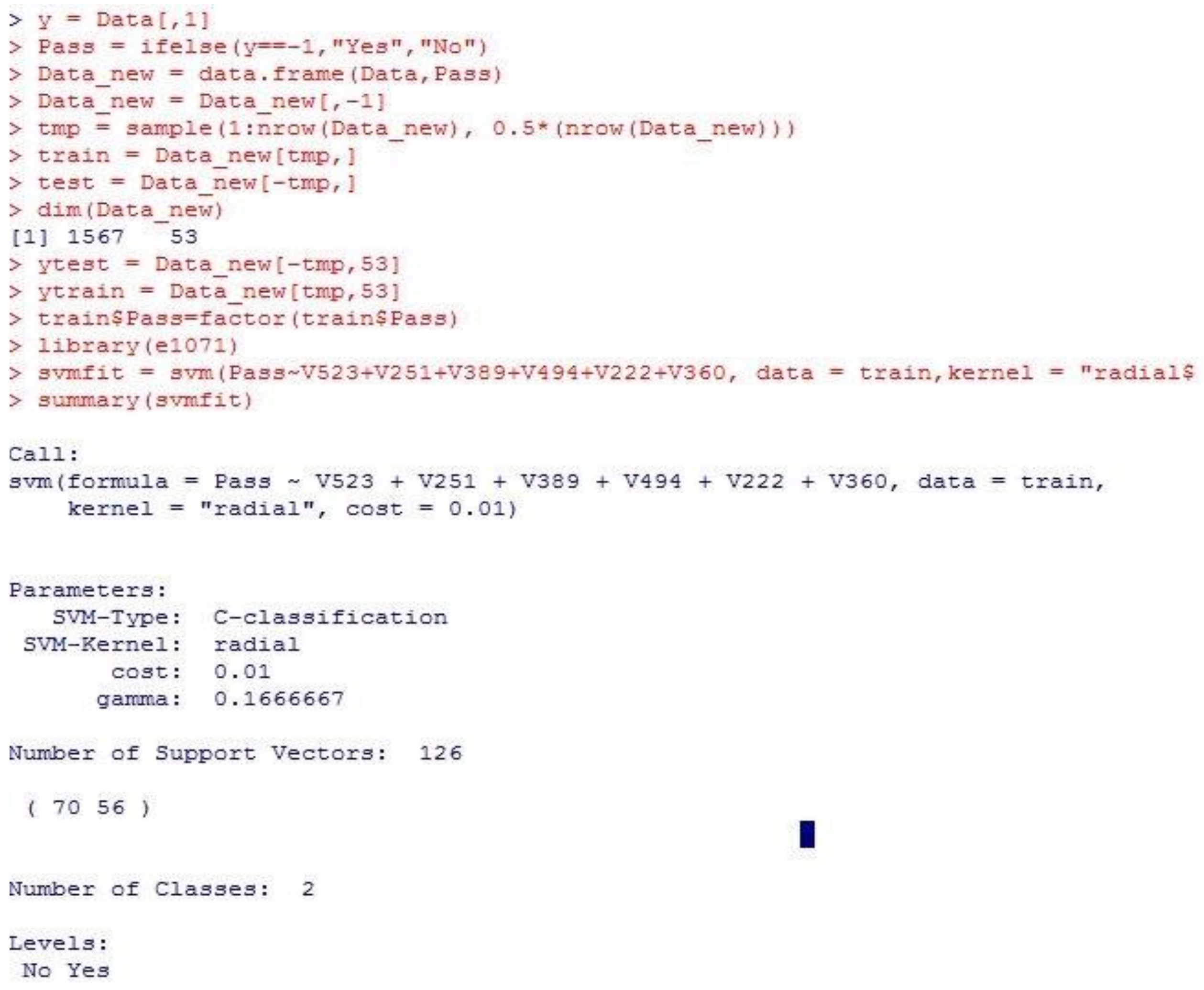
```
> pred<-predict(model,datatest)
> pred<-predict(model$finalModel,datatest)
> table(pred,ytest)
    ytest
pred   0   1
   0 732  52
   1   0   0
> mean(pred!=ytest)
[1] 0.06632653
```

The classification error rate obtained is 6.6326%.

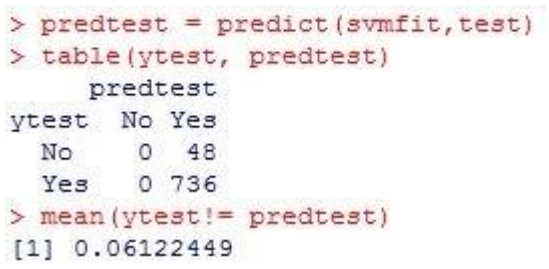
Predictors used	Error Rate	CV Error Rate
V251, V389	6.0509%	6.6326%

9. Support Vector Machine (SVM)

The model was incorporated iteratively with a combination of predictors and also used different kernels such as “linear” and “Radial”at a time. The best cost parameter and the gamma value are chosen by studying the summary of each of the SVM models using different kernel functions. The best model found was a model with radial kernel and the predictors: V523,V251,V389,V494,V222 and V360.



```
> y = Data[,1]
> Pass = ifelse(y==-1,"Yes","No")
> Data_new = data.frame(Data,Pass)
> Data_new = Data_new[,-1]
> tmp = sample(1:nrow(Data_new), 0.5*(nrow(Data_new)))
> train = Data_new[tmp,]
> test = Data_new[-tmp,]
> dim(Data_new)
[1] 1567   53
> ytest = Data_new[-tmp,53]
> ytrain = Data_new[tmp,53]
> train$Pass=factor(train$Pass)
> library(e1071)
> svmfit = svm(Pass~V523+V251+V389+V494+V222+V360, data = train,kernel = "radial$
> summary(svmfit)

Call:
svm(formula = Pass ~ V523 + V251 + V389 + V494 + V222 + V360, data = train,
    kernel = "radial", cost = 0.01)


Parameters:
   SVM-Type:  C-classification
 SVM-Kernel:  radial
       cost:  0.01
      gamma:  0.1666667

Number of Support Vectors:  126

 ( 70 56 )


Number of Classes:  2

Levels:
 No Yes
```



```
> predtest = predict(svmfit,test)
> table(ytest, predtest)
     predtest
ytest  No Yes
  No    0  48
  Yes   0 736
> mean(ytest!= predtest)
[1] 0.06122449
```

The classification error rate obtained is 6.1224%. We employed the method of cross validation, though we didn't get any distinguishing parameters and hence, we conclude our best model with radial kernel only based on the misclassification rate from the model.

RESULTS AND CONCLUSIONS

A summary of the various methods used and their respective error rates is as shown below.

METHOD	**MISCLASSIFICATION RATE**
Naive Baye's Classifier	**6.6326%**
Logistic Regression	**6.2075%**
Linear Discriminant Analysis	**6.8283%**
Quadratic Discriminant Analysis	**6.8283%**
k-Nearest Neighbours (kNN)	**7.4074%**
CART	**8.6734%**
Bagging and Random Forest	**6.3775%**
Boosting	**15.0593%**
Support Vector Machine(Radial Kernel)	**6.1224%**

After studying the sensitivity, specificity, the classification error rate and the interpretability of each model, we can conclude that the best method is the **Support Vector Machine method with Radial kernel for forecasting purposes**. We observe the lowest error rate to be 6.1224% according to this model. The error rates vary from a factor of +/- 1% when the model is run a multiple number of times. Hence the SVM(Radial Kernel) could be used for forecasting whether the semiconducting device would fail and the Tree-Based models could be used for feature selection which give us more interpretability of significant variables causing the failure of the semiconductor device. In the studied dataset we observe that the variable and/or signals that mainly influence the device failure are V292, V430 and V495. These signals may be used to determine key factors contributing to yield excursions downstream in the process which will enable an increase in process throughput, decreased time to learning and reduce the per unit production costs.