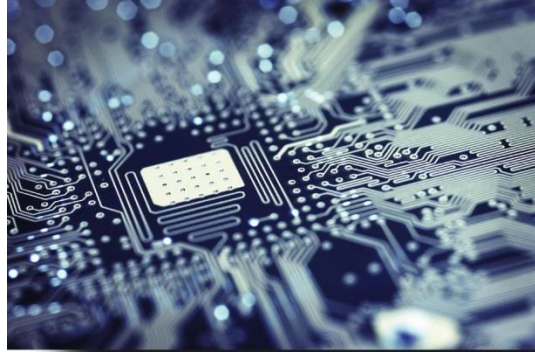


# Analysis of data from a semi-conductor manufacturing process using different classification models and Principal component analysis



**Submitted to:**

Dr.Satish Bukkapatnam

Professor in Industrial & Systems Engineering

Rockwell International Professor

**Submitted by:**

Jagadish Kumaran

Julien Edery

Praharsha Sunkara

Sai Jaswanth



## Table of Contents

1.	Project Proposal .....	5
2.	Literature Review .....	6
2.1	Project Introduction .....	6
2.2	Research Paper 1 .....	6
2.3	Research Paper 2 .....	9
2.4	Research Paper 3 .....	10
2.5	Research Paper 4 .....	11
2.6	Research Paper 5 .....	13
2.7	Research Paper 6 .....	14
2.8	Research Paper 7 .....	14
2.9	Research Paper 8 .....	16
2.10	Report Summary .....	18
2.11	Citations: .....	18
3.	Description of Methods.....	20
3.1	Naïve Bayes Classifier: - .....	20
3.2	Steps taken Block Diagram.....	21
4.	Implementation .....	22
4.1.	Install different libraries and read the data set .....	22
4.2.	Dimensions of the data and imputation of data .....	22
4.3.	Identifying the names and omitting the columns with constant values .....	23
4.4.	Applying Principal Component Analysis to the data set .....	23
4.5.	Applying Scree plot .....	25
4.6.	Applying Linear Regression Model on PCA .....	26
4.7.	Splitting the data into training and testing data sets.....	28
4.8.	Partial Least Squares .....	29
4.9.	CLASSIFICATION MODELS .....	32
4.9.1.	LOGISTIC REGRESSION MODEL .....	32
4.9.2.	CROSS VALIDATION for Logistic Regression .....	34
4.9.3.	ROC CURVE FOR BEST MODEL in Logistic Regression .....	36
4.9.4.	LINEAR DISCRIMINANT ANALYSIS .....	37
4.9.5.	CROSS VALIDATION for Linear Discriminant Analysis .....	38
4.9.6.	ROC CURVE FOR BEST MODEL in Linear Discriminant Analysis .....	40
4.9.7.	QUADRATIC DISCRIMINANT ANALYSIS .....	41
4.9.8.	CROSS VALIDATION for Quadratic Discriminant Analysis .....	42

4.9.9.	ROC CURVE FOR THE BEST MODEL in Quadratic Discriminant Analysis .....	44
4.9.10.	K-Nearest Neighbors .....	45
4.9.11.	CROSS VALIDATION for kNN .....	46
4.9.12.	ROC CURVE FOR kNN .....	47
4.9.13.	CART MODEL .....	48
4.9.14.	CROSS VALIDATION for CART .....	49
4.9.15.	ROC FOR CART MODEL.....	55
4.9.16.	BAGGING AND RANDOM FOREST .....	56
4.9.17.	BOOSTING .....	59
4.9.18.	NAÏVE BAYES CLASSIFIER.....	63
4.9.19.	CROSS VALIDATION for Naïve Bayes Classifier .....	64
4.9.20.	ROC Curve for Naïve Bayes Classifier .....	65
4.9.21.	SUPPORT VECTOR CLASSIFIER.....	66
4.9.22.	CROSS VALIDATION for Support Vector Machine.....	72
5.	Results & Conclusions:.....	73
	References.....	74

## List of Figures

Figure 1: Scree plot.....	25
Figure 2: PLS plot for model 1.....	29
Figure 3: PLS plot for model 2.....	30
Figure 4: PLS plot for model 3.....	31
Figure 5: ROC curve for logistic regression .....	36
Figure 6: ROC curve for Linear Discriminant Analysis.....	40
Figure 7: ROC Curve for Quadratic Discriminant Analysis .....	44
Figure 8: ROC Curve for kNN.....	47
Figure 9: Classification tree for model 1 .....	50
Figure 10: Classification tree for model2.....	52
Figure 11: Classification tree for model 3 .....	54
Figure 12: Plot of predicted vs tested for model 1 .....	56
Figure 13: Plot of predicted vs tested for model 2 .....	57
Figure 14: Plot of predicted vs tested for model 3 .....	58
Figure 15: Relative influence of predictors for model 1 .....	60
Figure 16: Relative influence of predictors for model 2 .....	61
Figure 17: Relative influence of predictors for model 3 .....	62

## List of Tables

Table 1: Error rates of different models for logistic regression.....	35
Table 2: Error rates of different models for Linear Discriminant Analysis.....	39
Table 3: Error rates of different models for Quadratic Discriminant Analysis.....	43
Table 4: Error rates of different models for kNN.....	47
Table 5: Error rates of different models for kNN.....	54
Table 6: Error rates of different models for bagging & random forest .....	59
Table 7: Error rates of different models for bagging & random forest .....	62
Table 8: Error rates of different models for Naïve Bayes Classifier .....	65
Table 9: Error rates of different models for support vector classifier .....	72

# 1. Project Proposal

**Title:** Detection/Monitoring of signals (pass/fail) from sensors for semiconductor manufacturing process using different analytical techniques

**Team:** Jagadish Kumaran, Praharsha Sunkara, Sai Jaswanth, Julien Edery

**Problem addressing:** Classification problem with response being binary (where 0 corresponds to a pass and 1 corresponds to a fail)

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

**Methods:** Logistic Regression, Linear Discriminant Analysis, Quadratic Discriminant Analysis, K Nearest neighbor approach, CART model, Bagging & Random Forest, Principal Component Analysis, Time Series Analysis

**Technical Challenges:** Learning new methods of solving the classification problem and dimensionally reducing a large data set

## **Project Plan:**

Tasks	Date of Completion
Brainstorming	11/19/2014
Dimensionality Reduction of data set	11/25/2014
Employing different methods	11/30/2014
Analysis and Interpretation	12/6/2014
Summarizing Results	12/15/2014

## **2.Literature Review**

### **2.1 Project Introduction**

Semiconductor manufacturing is a very complex process. It involves over 100 steps for the manufacturing of a single product. The manufacturing process involves series of steps to cover special material layers over the wafer surface. Defects occurring in this process can make the final products fail the test. Fault detection and classification techniques are applied to this critical manufacturing process which can help improve the quality of the product, reduce scrap, increase equipment uptime and also reliability.

The SECOM dataset is being used for this project. It consists of 1567 records taken from a wafer fabrication production line. Each record is a vector of 590 sensor measurements plus a label of pass/fail test. Among the 1567 records, there are 104 fail cases which are labeled as 1 and the rest are labeled as negative 0. The imbalance of pass and fail examples in addition to the large number of metrology data obtained from hundreds of sensors make this dataset a difficult one to accurately analyze.

We have been reading quite a few papers as a team. All the information we were able to gather from them are presented below one after another. Integration of all the research papers is not possible as each paper has its own set of data and approaches.

### **2.2 Research Paper 1**

In the research paper, "Feature Selection and Boosting Techniques to Improve Fault Detection Accuracy in the Semiconductor Manufacturing Process", by Kittisak Kerdprasop and Nittaya Kerdprasop, we inferred that a WEKA software was used to perform series of experiments. Selecting the vital few will show most differentiating fail cases from pass cases. In data preprocessing step 137 features were removed that contain a single value and lots of missing values.

From the remaining 454 features, 168 features were selected (to maintain around 95% of variances) by means of principal component analysis (PCA), Chi-square test, gain ratio computation, and our Mean diff method. The fault detection models were then derived from each feature selected data. They wanted the model that shows the highest values of TP rate, precision, and F-measure, but the lowest value in FP rate.

Actual class	Predicted class	
	Class=1 (fail)	Class= -1 (pass)
	Class= 1	Class= -1
Class= 1	TP	FN
Class= -1	FP	TN

$$\text{TP rate, or Recall} = \frac{TP}{TP + FN}$$

$$\text{Precision} = \frac{TP}{TP + FP}$$

$$\text{F - measure} = \frac{2TP}{2TP + FP + FN}$$

$$\text{FP rate} = \frac{FP}{FP + TN}$$

Fig. 2 classification matrix on predicting fail/pass cases and the computation of predicting performance

The symbols in the matrices can be explained as follows:  
TP = true positive or number of fail cases that are correctly identified as fail,  
FP = false positive or number of pass cases that are incorrectly identified as fail cases,  
FN = false negative or number of fail cases that are misclassified as pass cases,  
TN = true negative or number of pass cases that are correctly classified as pass cases

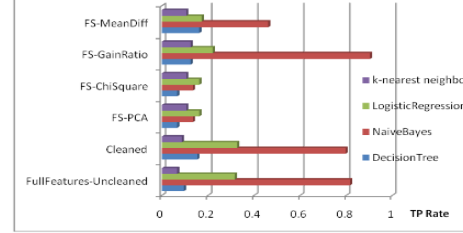


Fig. 3 TP rate or recall of fault detection models on different feature selection methods

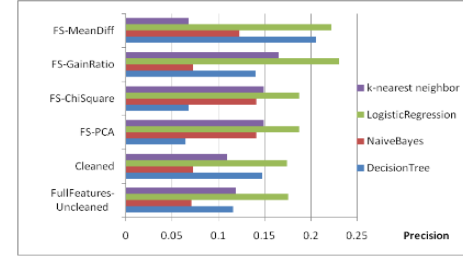


Fig. 4 Precision of fault detection models on different feature selection methods

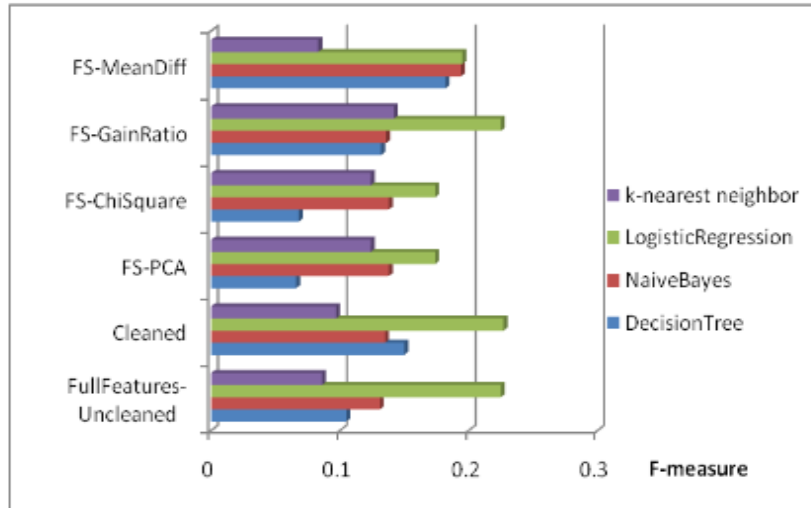


Fig. 5 F-measure of fault detection models on different feature selection methods

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.

TABLE I  
FAULT DETECTION MODEL ASSESSMENT

	k-Nearest Neighbor	Logistic Regression	Naïve Bayes	Decision Tree
TP rate	0.983	1.0	0.746	1.0
FP rate	0.24	0.335	0.352	0.161
Precision	0.372	0.301	0.234	0.472
F-measure	0.54	0.463	0.356	0.641

<b>k-Nearest Neighbor:</b>		Predicted class	
		Class=1 (fail)	Class= -1 (pass)
Actual class	Class= 1	58	1
	Class= -1	98	311

<b>Logistic regression:</b>		Predicted class	
		Class=1 (fail)	Class= -1 (pass)
Actual class	Class= 1	59	0
	Class= -1	137	272

<b>Naïve Bayes:</b>		Predicted class	
		Class=1 (fail)	Class= -1 (pass)
Actual class	Class= 1	44	15
	Class= -1	144	265

<b>Decision Tree:</b>		Predicted class	
		Class=1 (fail)	Class= -1 (pass)
Actual class	Class= 1	59	0
	Class= -1	66	343

Fig. 8 Performance matrices of each fault detection model

From a series of experiments, they discovered a feature selection technique based on the clustering analysis. Then, apply the over-sampling technique to duplicate fail cases.

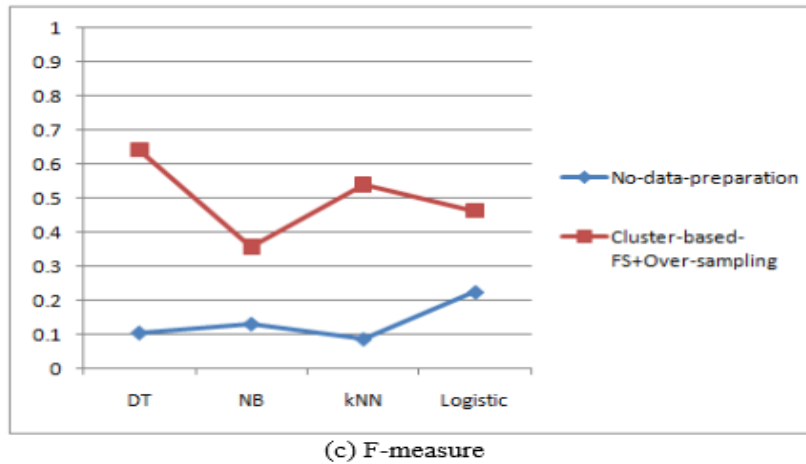


Fig. 9 Significant high increases in TP rate (or recall), Precision, and F-measure of fault detection models from applying the cluster-based feature selection and over-sampling techniques

It has been understood from the above figure that the data preparation techniques that combine the cluster-based feature selection and the over-sampling strategy to deal with class imbalance may appreciably improve the recall, precision, and f-measure of the four learning methods. It is found that Decision tree learning is the most accurate method. We thus can conclude that decision tree model is a good candidate for automatic generation of the fault detection model to be used in the semiconductor manufacturing process.



## 2.3 Research Paper 2

In the research paper, “Semiconductor manufacturing process monitoring based on adaptive substatistical PCA”, by Z.Ge and Z.Song proposes a new adaptive substatistical PCA-based method that can avoid future value estimation, new monitoring statistics scheme that has no Gaussian limitation of process data by employing support vector data description and also detailing the correlations among the new method, multimodel, and multi-way PCA. This paper focuses on process monitoring technologies for quality improvement. The main objective of this paper is to develop a new adaptive substatistical PCA-based method to account for the drawbacks of Multi-way PCA.

### Approach and results:

The methods used include Multi-way PCA, Multi-model PCA, Adaptive Substatistical PCA (ASSP) and Adaptive Substatistical PCA with Support vector data description (ASSP\_SVDD). The statistics used with these methods are Hotelling’s T square statistic and Q statistic. The data employed is from an Aluminum stack etch process performed on a commercial scale Lam9600 plasma etch tool at Texas Instruments. Original data set: 40 variables with 19 variables chosen for fault detection. Two variables remain zero during all of the batch time. No of monitoring variables reduced to 17. 107 batches and 20 faulty batches are used for case study. 95 normal batches were randomly chosen as training datasets. The rest 12 normal batches and 20 faulty batches are used for testing.

TABLE IV  
FAULT DETECTION RESULTS OF DIFFERENT METHODS FOR FAULTY BATCHES

Fault	MPCA_T <sup>2</sup>	MPCA_SPE	MMPCA_T <sup>2</sup>	MMPCA_SPE	ASSP_T <sup>2</sup>	ASSP_SPE	ASSP_SVDD_T <sup>2</sup>
1	0.00	28.98	4.57	58.43	4.33	72.99	9.07
2	0.00	3.86	1.02	14.92	0.28	18.49	9.78
3	0.00	0.04	0.11	1.87	0.02	8.89	1.91
4	85.00	85.00	78.47	85.00	81.43	85.00	81.78
5	0.00	0.07	0.39	2.62	0.01	9.75	0.18
6	0.00	0.00	0.00	1.00	0.00	6.97	0.53
7	73.98	85.00	56.62	85.00	63.79	85.00	69.45
8	0.05	1.17	1.00	6.25	0.16	11.88	25.53
9	0.00	1.04	0.04	3.55	0.01	12.93	1.18
10	0.00	9.75	0.70	32.53	0.30	51.12	2.72
11	0.00	4.53	1.16	8.08	1.43	14.67	5.51
12	4.95	85.00	85.00	85.00	85.00	85.00	85.00
13	72.50	83.73	42.19	85.00	45.98	85.00	62.05
14	0.09	8.41	1.01	20.67	0.78	34.26	7.77
15	0.00	1.14	0.32	11.43	0.03	24.77	0.14
16	0.03	12.30	4.78	24.06	5.15	37.38	12.51
17	0.00	0.80	0.35	5.48	0.00	15.62	2.14
18	7.02	41.18	14.95	57.80	21.41	69.96	35.12
19	0.30	15.49	4.17	39.78	2.62	54.06	24.11
20	0.00	3.73	0.00	7.90	0.02	18.88	18.17

New proposed method more efficient monitoring in PCS has been improved by new SVDD-based statistic

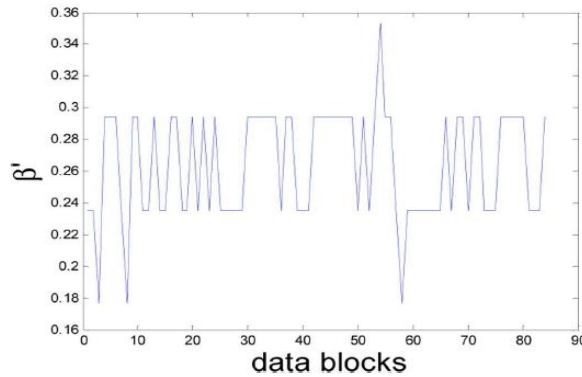


Fig. 5. The ratio of adapted PC number and variable number during the whole batch.

#### Inference:

It can be seen that there is a sudden change of Beta in the middle of the batch duration. It can be inferred that the stage of the process may be changed.

In the beginning there are two low beta values in the beginning of the batch. This is caused by some initial fluctuation

## 2.4 Research Paper 3

From the research paper by Q.P. He and J. Wang, “Fault detection using the k-nearest neighbor rule for semiconductor manufacturing processes,” we were able to find that the issue addressed in this paper is aimed at improving the process operations by reducing scrap, increase equipment uptime, and reduce the usage of test wafers. This paper proposes a new fault detection method “Fault detection KNN rule” that can avoid non linearity in most batch processes, multimodal batch trajectories due to product mix, and process steps with variable durations. This paper also focuses on comparing PCA method with FD-KNN proposed method. The main objective of this paper is to develop a new fault detection method to explicitly account for the unique characteristics of semiconductor process such as non –linearity of data, multi-modal batches.

#### Approach and results:

The methods used include Data preprocessing, Principal Component Analysis (PCA) and Fault Detection K Nearest Neighbors (FD-KNN). The statistics used with these methods are Hotelling’s T square statistic and SPE statistic. The data employed is from an Aluminum stack etch process performed on a commercial scale Lam9600 plasma etch tool at Texas Instruments. Original data set: 107 normal wafers from 3 experiments and 21 wafers with intentionally induced faults

TABLE I  
FAULT DETECTED BY DIFFERENT METHODS

Fault	PCA-SPE	PCA-T <sup>2</sup>	FD-kNN	PC-kNN
1	✓		✓	✓
2		✓	✓	✓
3				
4	✓	✓	✓	✓
5				✓
6				
7	✓	✓	✓	✓
8		✓	✓	✓
9				
10	✓		✓	✓
11			✓	
12	✓	✓	✓	✓
13	✓	✓	✓	✓
14	✓		✓	✓
15			✓	✓
16	✓	✓	✓	✓
17			✓	✓
18	✓	✓	✓	✓
19	✓	✓	✓	✓
20	✓	✓	✓	✓

Inference: We observe that FD-KNN has detected most number of faults. From the results we see that FD-KNN performs well than PCA with limited data processing and also capable of handling multimodal data.

#### Summary:

- Proposes a new fault detection method FD-KNN
- Provides simulation examples to illustrate fault detection capability of new proposed method
- Comparison of proposed method and PCA using industrial example

## 2.5 Research Paper 4

The research paper, ‘Semiconductor manufacturing process control and monitoring : A Fab-wide framework’, by S. Joe Qin, Gregory Cherry, Richard Good, Jin Wang, Christopher A. Harrison, proposes that the hierarchical fab-wide control framework for the semiconductor manufacturing with the integration of the 300 equipment and automated material handling system. Our focus on this research paper will be only on the data monitoring, fault detection and analysis techniques employed.

#### Approach and results:

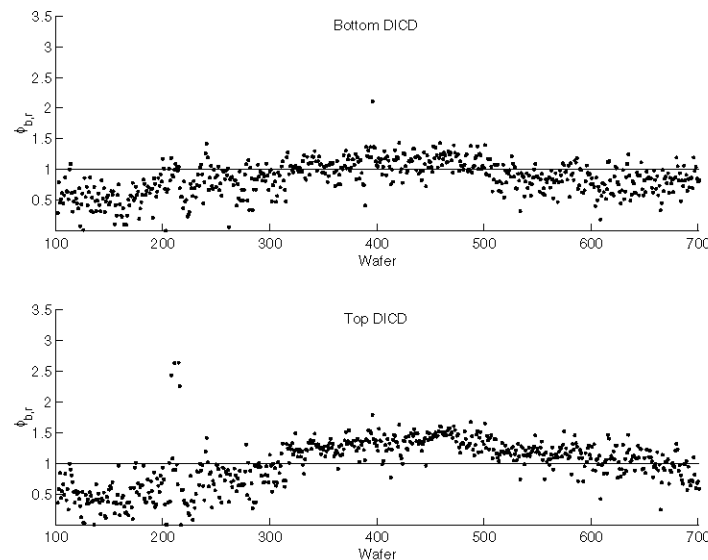
The data driven fault detection techniques are on multivariate statistical analysis such as principal component analysis (PCA) and partial least squares (PLS).

The following measurement data were considered for analysis and control:

- Real time data at the tool level
- Integrated metrology and in-line metrology data available for geometric dimensions

There are 700 wafers in this data set, where both the top and bottom of the resist are measured at nine sites on each wafer. An initial PCA model is built using the first 100 wafers and the VRE method which selects 10 components as the optimal number for reconstruction. The aim of VRE method is the reconstruction of variables. After formulation of PCA model, it is applied to the remaining 600 wafers, with the fault detection indices, SPE, T2, and theta that is the combined index.

The DICD (development inspection critical dimension) fault detection using SPEr, T2r, and Ur demonstrate that there is some behavior within those 600 wafers that is not consistent with the initial set of 100 wafers. In order to find the cause of the excursion, multiblock contributions to the combined index are calculated. For the case of site-level metrology, the most logical blocking of the 18 variables are by parameter (top and bottom) and by site (sites 1–9).



According to the paper, the inspection of DICD fault identification using parameter allows one to understand the general trend caused by drifts in both the bottom and top DICDs as time passes. The larger outliers are more pronounced in either the top or bottom, but not both. Due to the high correlation of the two CD's, the presence of such extreme outliers is likely to be caused by inaccurate values provided by the CD metrology tool, rather than problems with the physical structures on the wafers. The contribution plot showed that all faults indicated in the measurement contribution also propagated themselves to their corresponding multiblock contributions for both parameters and sites. This will help a well-designed fab-wide control framework improved competitiveness of the semiconductor manufacturers as they transition to 300 mm technology and fault detections.

## 2.6 Research Paper 5

The research paper, “Adaptive Mahalanobis Distance and k-Nearest Neighbor Rule for Fault Detection in Semiconductor Manufacturing”, by Ghislain Verdier and Ariane Ferreira states that, in semiconductor manufacturing, statistical control charts are widely used for fault detection. Among them, the most popular ones are the Hotelling T2 rule which makes sense when the variables are Gaussian and k-nn rule otherwise. In this paper, a new adaptive Mahalanobis distance, which takes into account the local structure of dependence of the variables, is proposed.

### Approach and Results:

A typical FDC system collects the on-line data from the process equipment sensors for every process run. The cumulative distance of an observation to its k nearest neighbors is too large, the observation is said to be out of control. The aim of this paper is to propose a new distance, more reliable than the Euclidean distance, for the k-nearest neighbor detection rule. In this paper, the k-NN rule is detailed and the new adaptive distance is introduced. The k-NN is compared to the Hotelling T2 and a nonparametric kernel detection rule on simulation trials.

According to the paper, simulation trials are performed in order to compare the new adaptive distance with the Euclidean distance for the k-NN rule. The number of detections is compared for the following rules: Hotelling T2 rule, k-NN rule with Euclidean distance (k=5, 10, and 15), k-NN rule with the new adaptive Mahalanobis distance, nonparametric kernel detection rule. The result of the comparison state that

TABLE I  
COMPARISON OF THE PERFORMANCE OF HOTELLING  $T^2$ , k-NN RULES, AND NONPARAMETRIC KERNEL DETECTION RULE, FOR  $N = 250$ ,  $N = 500$ , AND  $N = 1000$

N=250	Decision Rules													
	k-NN rule										Nonpar. kernel			
	Hot. T <sup>2</sup>	Euclidean Distance			Adaptive Distance (k=10)						K(x) = $\frac{e^{-\frac{\log^2}{2\sigma^2}}}{\sqrt{2\pi}}$ , $\delta = C \cdot N^{-\frac{1}{2}}$			
		k=5	k=10	k=15	K=20	K=50	K=80	K=110	K=140	K=170	C=0.75	C=0.95	C=1	C=1.25
Mean	34.00	45.66	42.06	39.21	44.07	51.26	54.58	54.99	54.40	49.04	83.66	67.45	62.17	31.37
Variance	25.6	801.8	708.6	623.7	621.0	709.3	723.4	732.0	702.5	618.7	44.4	84.1	105.0	85.8
FAR (%)	0.91	0.73	0.76	0.80	0.63	0.68	0.69	0.70	0.68	0.67	1.84	1.01	0.89	0.47
N=500	Decision Rules													
	k-NN rule										Nonpar. kernel			
	Hot. T <sup>2</sup>	Euclidean Distance			Adaptive Distance (k=10)						K(x) = $\frac{e^{-\frac{\log^2}{2\sigma^2}}}{\sqrt{2\pi}}$ , $\delta = C \cdot N^{-\frac{1}{2}}$			
		k=5	k=10	k=15	K=70	K=110	K=150	K=190	K=230	K=270	C=0.75	C=0.95	C=1	C=1.25
Mean	34.75	69.15	70.45	63.16	71.45	74.43	75.43	75.91	75.53	74.84	84.63	73.50	68.71	39.70
Variance	14.6	464.2	443.7	585.4	379.3	331.9	338.1	325.9	310.4	318.8	37.5	73.2	86.5	80.3
FAR (%)	0.89	0.98	0.98	1.00	0.98	0.96	0.96	0.96	0.95	0.98	1.57	0.98	0.85	0.51
N=1000	Decision Rules													
	k-NN rule										Nonpar. kernel			
	Hot. T <sup>2</sup>	Euclidean Distance			Adaptive Distance (k=10)						K(x) = $\frac{e^{-\frac{\log^2}{2\sigma^2}}}{\sqrt{2\pi}}$ , $\delta = C \cdot N^{-\frac{1}{2}}$			
		k=5	k=10	k=15	K=120	K=210	K=300	K=390	K=480	K=570	C=0.75	C=0.95	C=1	C=1.25
Mean	35.67	82.36	82.60	80.22	83.24	85.26	86.17	86.14	85.47	85.56	87.22	79.78	76.49	51.23
Variance	9.4	173.1	208.4	203.0	115.4	100.3	88.2	84.5	92.0	100.3	29.7	49.31	60.9	71.5
FAR (%)	0.92	1.00	0.99	0.98	0.98	0.99	0.99	0.99	0.98	1.00	1.48	0.99	0.88	0.53

Table I shows the mean and variance of the number of detections for 500 repetitions and different values for the learning sample size N. The Hotelling T2 rule gave the worst results since only one out of three out-of-control trajectories are detected. For the case  $N = 250$ , the best results are obtained with the nonparametric kernel detection rule. For  $N = 500$ , the k-NN rule with the adaptive distance gives better results than the kernel detection rule for a large number of values for K. And for  $N = 1000$ , the k-NN rule with the adaptive distance outperforms the kernel

detection rule. The results confirm that the new adaptive distance or Mahalanobis distance performs better than the Euclidean distance when the two parameter  $k$  and  $K$  are well chosen. More interesting, the adaptive distance seems to be more robust with respect to the choice of the parameter  $k$ .

## 2.7 Research Paper 6

According to the paper, “Fault Detection for a Via Etch process using Adaptive Multivariate Methods”, by Gerhard Spitzlsperger, Carsten Schmidt, Guenther Ernst, Hans Strasser, Michaela Speil, they have taken a semiconductor dataset and analyzed univariate and multivariate control charts in detecting the out of control conditions due to various parameters. In this dataset, there are many parameters so we may need as many univariate charts. Also changes due to correlation can't be detected by Univariate chart. So they wish to use multivariate charts to overcome the shortcomings of univariate charts. But these multivariate charts are poor in detecting slow changes in parameters over a long period. Then they find the Hotelling  $T^2$  statistic for each sample. When applying the UCL for  $T^2$  chart, they find that the theoretical UCL is not applicable because the data here is not normally distributed which is the assumption in  $T^2$  chart. So with domain knowledge they arbitrarily fix the UCL. Since the process drifts slowly away from the training distribution parameters, EWMA is employed along with PCA and the distribution parameters are updated. This is termed as full adaptive update. To decrease the false alarms and to detect the slow drifts, it is advised to get inputs from the engineers with domain knowledge. So they are updating the mean and variance of those parameters that are drifting. This is termed as Partial adaptive update. Next approach is finding a new model using moving window method. Here the covariance matrix is recalculated.

They compare the performance different versions of control charts such as Static  $T^2$ , partial adaptive  $T^2$ , and full adaptive  $T^2$  etc. on attributes such as Sensitivity, false alarms, detection of local faults, slow drifts and maintenance of each chart.

They have concluded that for the handled dataset, univariate chart would have performed better, number of charts is manageable. They felt univariate charts are more appropriate in this case as interaction between factors is not relevant here.

## 2.8 Research Paper 7

According to the paper by A.M. Ison, W. Li, and C.J. Spanos, “Fault diagnosis of plasma etch equipment,” the issue addressed is aimed at improving the accommodation of larger wafer sizes and meeting more stringent design demands. This necessitates accurate and robust characterization of the manufacturing process as well as reliable prediction and control of its effects on the final wafer product. This paper proposes to focus on plasma etching

because it is considered a critical manufacturing process and yield limiter. This paper compares tree-based models to generalized linear models. The main objective of this paper is to show the effectiveness of tree-based models in identifying sensor signals most sensitive to changes in the input settings of the machine. This method is compared with the performance of generalized linear models built to predict levels of the input settings based on sensor signals

### Approach

- Designed experiment was conducted on a Lam Rainbow 4400 plasma etcher in the U.C. Berkeley micro fabrication Laboratory
- Real time tool signals were collected while wafers were being processed at a rate of 1 Hz.

Response	Predictors
Pressure	DCBias, Power, Phase, Impedance, RFCoil
RF Power	DCBias, EndpointA,B
Gas Ratio	RFTune, RFCoil, MFC3, Impedance, DCBias, EndpointC
Total Flow	MFC3, MFC6, HeCFlow, Impedance, Pressure
Gap Spacing	RFTune, RFCoil, Phase, Impedance, Volt, DCBias, EndpointC, Pressure

- Predictor variables for input setting responses

Response	Predictors	Misc	Valid
Pressure	DCBias, Power	0.1667	0.6667
RF Power	Endpoint A	0.0416	0.0833
Gas Ratio	RFTune, RFCoil	0.25	0.4167
Total Flow	MFC3	0.125	0.5
Gap Spacing	Endpoint C	0.0833	0.1667

### Classification tree results

Response	High Misc	High Valid	Low Misc	Low Valid
Pressure	0.0000	0.0000	0.0000	0.0000
RF Power	0.0000	0.0000	0.0000	0.0000
Gas Ratio	0.2500	0.2500	0.1250	0.2500
Total Flow	0.0000	0.0000	0.0000	0.6667
Gap Spacing	0.0000	0.3333	0.0000	0.0000

- Generalized linear models results
- Classification trees are shown to be effective in predicting changes in the input settings using only a small subset of the real-time tool signals.

### Summary

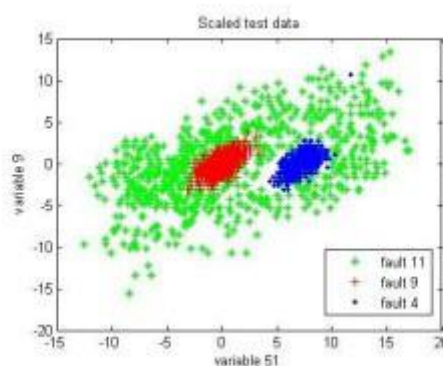
- Examination of the diagnostic performance of two probabilistic modeling techniques in using sensor signals to classify faults.
- Discussion of how the strength of these models may be combined in a hierarchical architecture giving rise to a more powerful diagnostic tool.

## 2.9 Research Paper 8

According to the paper by E. Tafazzoli and M. Saif, “Application of combined support vector machines in process fault diagnosis,” the issue here is to model the Tennessee Eastman chemical process and the Three Tank system which are a classic example used in research. This paper compares the Combined Support Vector Machines with K- nearest and simple SVM. The main objective of this paper is to examine the performance of, C-SVM, by comparing its classification results with k-nearest neighbor and simple SVM classifier

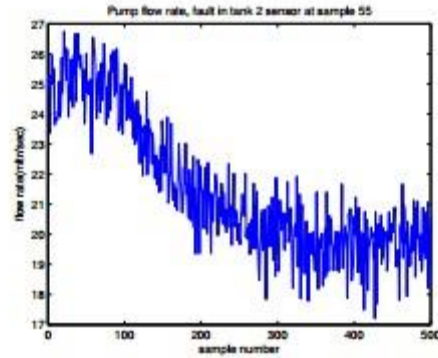
### Approach

- Experiments are the Tennessee Eastman process and the Three Tank system.
- For the TE process: each set of training and testing data contain  $480 \times 52$  and  $960 \times 52$  points respectively, observed every three min of simulation and faults occur after 1 hour and 8 hour of simulation respectively.
- For the Three Tank system the training and testing data size are  $500 \times 5$  for each case of fault with water levels and flow rates as variables. Faults are instigated at sample 55 in each case.



- Data plot for the TE process





- Flow Rate for three tank system

TABLE I  
CLASSIFICATION ERROR FOR DIFFERENT CLASSIFIERS APPLIED TO TE  
PROCESS DATA

Classifier	Classification error %
SVM(linear kernel)	26.7
SVM(RBF kernel)	8.3
SVM(Polynomial kernel)	7.3
C-SVM	6.7
KNN classifier	8.4

- The CSVM gives the best result for classification with 6.7% classification error.

TABLE II  
CLASSIFICATION ERROR FOR DIFFERENT CLASSIFIERS APPLIED TO  
THREE TANK SYSTEM DATA

Classifier	Classification error %
SVM(linear kernel)	14.03
SVM(RBF kernel)	13.74
SVM(Polynomial kernel)	30.53
C-SVM	12.17
KNN classifier	14.57

- The CSVM gives the best result for classification with 12.17% classification error. Summary:
  - The performance of C-SVM, is examined by comparing it's classification results with k-nearest neighbor and simple SVM classifier
  - Results show that the C-SVM classifier gives the lowest classification error compared to other methods.
  - Appropriate selection of number of variables using contribution charts for classification enhances the performance of the classifiers on Tennessee Eastman.

## 2.10 Report Summary

The semiconductors manufacturing processes consists of humongous amount of data for analysis and

achieve optimization. Because of the large data in the dataset studies have shown that fault detection is the best way to approach this situation. According to the research reports, there have been different

methods used such as KNN, logistic regression, Bayes classifier, Adaptive Statistical PCA and so on for

fault detection. From the experiments conducted, the Bayes classification was able to detect fault cases at nearly 90% but at the same time, the false positive rate is also at 80%. CART method was able to generate a fault detection model of false positive rate of only 4.5%, but the true positive rate is at 16%.

In the end a boosting model is devised to improve the precision for fault detection, the TPR was found to be 100% while the false positive rate was 16%.

## 2.11 Citations:

- [1] T. F. Edgar, S. W. Butler, W. J. Campbell, C. Pfeiffer, C. Bode, S. B.Hwang, K. S. Balakrishnan, and J. Hahn, "Automatic control in microelectronics manufacturing: Practices, challenges, and possibilities," *Automatica*, vol. 36, pp. 1567–1603, 2000.
- [2] B. M. Wise, N. B. Gallagher, S. W. Butler, D. D. White, Jr, and G.G. Barna, "A comparison of principal component analysis, multiway principal component analysis, trilinear decomposition and parallel factor analysis for fault detection in a semiconductor etch process," *J.Chemometr.*, vol. 13, pp.379–396, 1999.
- [3] G. A. Cherry and S. J. Qin, "Multiblock principal component analysis based on a combined index for semiconductor fault detection and diagnosis," *IEEE Trans. Semicond. Manuf.*, vol. 19, pp. 159–172, 2006.
- [4] J. J. Gertler, *Fault Diagnosis: Models, Artificial Intelligence, Applicati-ons*. Marcel Dekker, 1998.
- [5] P. Mhaskar, J. Liu, and P. D. Christofides, *Fault-Tolerant Process Control: Methods and Applications*.Bristol,UK,Springer,2013

- [6] A. Zolghadri, D. Henry, J. Cieslak, D. Efimov, and P. Goupil, Fault Diagnosis and Fault-Tolerant Control and Guidance for Aerospace Vehicles: From Theory to Application, 2nd ed. London: Springer, 2013.
- [7] J. M. Candelaria, Fault detection and isolation in low-voltage DC-bus micro grid systems. ProQuest, 2012.
- [8] Z. Ge and Z. Song, "Semiconductor manufacturing process monitoring based on adaptive substatistical PCA," IEEE Trans. Semiconductor Manufacturing, vol.23, no.1, pp.99-108, Feb. 2010.
- [9] Q.P. He and J. Wang, "Fault detection using the k-nearest neighbor rule for semiconductor manufacturing processes," IEEE Trans. Semiconductor Manufacturing, vol.20, no.4, pp.345-354, Nov. 2007.
- [10] S.J. Qin, G. Cherry, R. Good, J. Wang, and C.A. Harrison, "Semiconductor manufacturing process control and monitoring: a fab-wide framework," Journal of Process Control, vol.16, pp.179-191, 2006.
- [11] G. Verdier and A. Ferreira. (2010). Adaptive Mahalanobis distance and k-nearest neighbor rule for fault detection in semiconductor manufacturing. IEEE Trans. Semiconductor Manufacturing. Available: doi:10.1109/TSM.2010.2065531
- [12] G. Spitzlsperger, C. Schmidt, G. Ernst, H. Strasser, and M. Speil, "Fault detection for a via etch process using adaptive multivariate methods," IEEE Trans. Semiconductor Manufacturing, vol.18, no.4, pp.528-533, Nov. 2005

### 3. Description of Methods

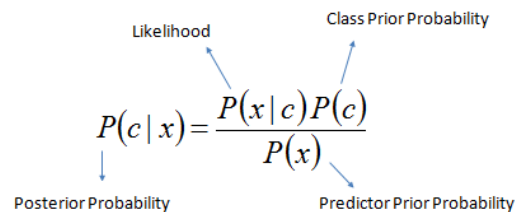
#### 3.1 Naïve Bayes Classifier: -

One of the main new methods we used for this Project on Semiconductor Manufacturing Data is the Naïve Bayes Classification. According to the Research Paper by Kittisak Kerdprasop and Nittaya Kerdprasop, “ Feature Selection and Boosting Techniques to Improve Fault Detection Accuracy in the Semiconductor Manufacturing Process”, one of the major methods of classification used were Logistics Regression, k-Nearest Neighbors, CART and Naïve Bayes Classification. We picked up the idea of using Naïve Bayes Analysis after reading through this research paper and understanding the working of it.

Naïve Bayes classifiers are based on general application of Bayes Theorem. It is called naïve because the features are independent. Naïve Bayes method may reference the use of Bayes theorem in the classifier decision rules but the naïve Bayes is not necessarily the Bayesian Classification. It is well suited when the dimensionality of input data is high. It is simple and is said to outperform many other sophisticated classification methods. It can handle any number of independent variables irrespective of whether they are continuous or categorical.

#### 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  $P(x|c)$ .



The diagram shows the formula for Bayes' Theorem: 
$$P(c | x) = \frac{P(x | c)P(c)}{P(x)}$$
 Arrows point from the terms in the formula to their respective labels: 

- $P(c | x)$  is labeled "Posterior Probability" (with a downward arrow).
- $P(x | c)$  is labeled "Likelihood" (with an upward arrow).
- $P(c)$  is labeled "Class Prior Probability" (with an upward arrow).
- $P(x)$  is labeled "Predictor Prior Probability" (with a downward arrow).

$$P(c | X) = P(x_1 | c) \times P(x_2 | c) \times \dots \times P(x_n | c) \times P(c)$$

Source: [http://www.saedsayad.com/naive\\_bayesian.htm](http://www.saedsayad.com/naive_bayesian.htm)

### 3.2 Steps taken Block Diagram

Major Steps taken for the completion of this project



## 4. Implementation

### 4.1. Install different libraries and read the data set

```
> library(ISLR)
> library(MASS)
> library(boot)
> library(class)
> library(psy)
> library(randomForest)
> library(tree)
> library(hmeasure)
> library(Deducer)
> library(ROCR)
> library(e1071)
> library(Discriminer)
> library(gbm)
> library(party)
> library(pls)
> library(caret)
> library(ipred)
```

### 4.2. Dimensions of the data and imputation of data

```
> dim(data)
[1] 1567 591
```

The data set has 1567 observations and 591 variables

```
> is.na(data)|
[77,] FALSE TRUE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
[78,] FALSE TRUE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
[79,] FALSE TRUE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
[80,] FALSE TRUE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
[81,] FALSE TRUE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
[82,] FALSE TRUE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
[83,] FALSE TRUE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
[84,] FALSE TRUE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
[85,] FALSE TRUE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
[86,] FALSE TRUE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
[87,] FALSE TRUE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
[88,] FALSE TRUE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
[89,] FALSE TRUE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
[90,] FALSE TRUE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
[91,] FALSE TRUE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
[92,] FALSE TRUE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
[93,] FALSE TRUE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
[94,] FALSE TRUE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
[95,] FALSE TRUE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
[96,] FALSE TRUE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
[97,] FALSE TRUE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
[98,] FALSE TRUE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
[99,] FALSE TRUE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
[100,] FALSE TRUE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
[101,] FALSE TRUE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
```

You can observe that there are missing values

### 4.3. Identifying the names and omitting the columns with constant values

```
> names(data)
[1] "y" "x1" "x2" "x3" "x4" "x5" "x6" "x7" "x8" "x9"
[11] "x10" "x11" "x12" "x13" "x14" "x15" "x16" "x17" "x18" "x19"
[21] "x20" "x21" "x22" "x23" "x24" "x25" "x26" "x27" "x28" "x29"
[31] "x30" "x31" "x32" "x33" "x34" "x35" "x36" "x37" "x38" "x39"
[41] "x40" "x41" "x42" "x43" "x44" "x45" "x46" "x47" "x48" "x49"
[51] "x50" "x51" "x52" "x53" "x54" "x55" "x56" "x57" "x58" "x59"
[61] "x60" "x61" "x62" "x63" "x64" "x65" "x66" "x67" "x68" "x69"
[71] "x70" "x71" "x72" "x73" "x74" "x75" "x76" "x77" "x78" "x79"
[81] "x80" "x81" "x82" "x83" "x84" "x85" "x86" "x87" "x88" "x89"
[91] "x90" "x91" "x92" "x93" "x94" "x95" "x96" "x97" "x98" "x99"
[101] "x100" "x101" "x102" "x103" "x104" "x105" "x106" "x107" "x108" "x109"
[111] "x110" "x111" "x112" "x113" "x114" "x115" "x116" "x117" "x118" "x119"
[121] "x120" "x121" "x122" "x123" "x124" "x125" "x126" "x127" "x128" "x129"
[131] "x130" "x131" "x132" "x133" "x134" "x135" "x136" "x137" "x138" "x139"
[141] "x140" "x141" "x142" "x143" "x144" "x145" "x146" "x147" "x148" "x149"
[151] "x150" "x151" "x152" "x153" "x154" "x155" "x156" "x157" "x158" "x159"
[161] "x160" "x161" "x162" "x163" "x164" "x165" "x166" "x167" "x168" "x169"
[171] "x170" "x171" "x172" "x173" "x174" "x175" "x176" "x177" "x178" "x179"
[181] "x180" "x181" "x182" "x183" "x184" "x185" "x186" "x187" "x188" "x189"
[191] "x190" "x191" "x192" "x193" "x194" "x195" "x196" "x197" "x198" "x199"
[201] "x200" "x201" "x202" "x203" "x204" "x205" "x206" "x207" "x208" "x209"

> data.2 <- data.frame(t(na.omit(t(data))))
> dim(data.2)
[1] 1567 53
```

You can observe that the dimensions of the data set have been reduced to 53 variables. We have omitted those columns which have constants since the variance of those constants is always zero

### 4.4. Applying Principal Component Analysis to the data set

```
> pr.out=prcomp(data.2, scale=TRUE)
> summary(pr.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(pr.out)
[1] "sdev"      "rotation" "center"    "scale"     "x"
> pr.out$center
      y      x21      x87      x88      x89      x114
6.636886e-02 1.405054e+00 2.401872e+00 9.824203e-01 1.807815e+03 9.454242e-01
      x115      x116      x117      x118      x120      x121
1.225271e-04 7.473838e+02 9.871299e-01 5.862591e+01 9.707771e-01 6.310863e+00
      x157      x222      x223      x224      x249      x250
5.808858e-02 6.071755e-02 8.821187e-03 1.228466e+02 2.517122e-02 1.065220e-03
      x251      x252      x253      x255      x256      x292
1.096510e+02 4.284812e-03 4.645115e+00 1.394320e-02 4.038481e-01 1.992623e-02
      x360      x361      x362      x387      x388      x389
1.983995e-02 2.944863e-03 3.993641e+01 8.280791e-03 3.392470e-04 3.515509e+01
      x390      x391      x393      x394      x430      x494
1.338354e-03 1.431868e+00 4.532802e-03 1.339895e-01 4.171844e+00 2.530046e+00
      x495      x496      x521      x522      x523      x524
9.564420e-01 6.807826e+00 2.695999e+00 1.161008e+01 1.472887e+01 4.538964e-01
      x525      x527      x528      x571      x572      x573
5.687782e+00 1.443457e+00 6.395717e+00 5.305236e+02 2.101836e+00 2.845017e+01
      x574      x575      x576      x577      x578
3.456364e-01 9.162315e+00 1.047292e-01 5.563747e+00 1.664236e+01
> pr.out$sdev
      y      x21      x87      x88      x89      x114
2.490052e-01 1.673693e-02 3.733186e-02 1.284784e-02 5.353726e+01 1.213287e-02
      x115      x116      x117      x118      x120      x121
1.668094e-03 4.894925e+01 9.496504e-03 6.485174e+00 8.948562e-03 1.243044e-01
      x157      x222      x223      x224      x249      x250
7.917449e-02 2.330523e-02 5.593677e-02 5.515600e+01 4.923507e-02 1.577079e-02
      x251      x252      x253      x255      x256      x292
5.459727e+01 3.747249e-02 6.435476e+01 9.132139e-03 1.203342e-01 2.554882e-02
      x360      x361      x362      x387      x388      x389
7.136358e-03 2.000257e-02 1.705630e+01 1.548813e-02 4.989334e-03 1.722700e+01
      x390      x391      x393      x394      x430      x494
1.181600e-02 2.032642e+01 2.955553e-03 3.840849e-02 6.435390e+00 9.739483e-01
      x495      x496      x521      x522      x523      x524
6.615200e+00 3.260019e+00 5.702366e+00 1.031230e+02 7.104435e+00 4.147581e+00
      x525      x527      x528      x571      x572      x573
2.066341e+01 9.584275e-01 1.888698e+00 1.749974e+01 2.751118e-01 8.630468e+01
      x574      x575      x576      x577      x578
2.484781e-01 2.692015e+01 6.779112e-02 1.692137e+01 1.248527e+01
> pr.out$rotation
      PC1      PC2      PC3      PC4      PC5
y      0.020073843 -2.250935e-02 -0.001369181 0.0134101720 -0.0148027588
x21     -0.018409455 -1.968703e-02 0.006201561 0.0001639471 -0.0053922733
x87     -0.014757798 1.844903e-02 0.003348590 0.0013958675 -0.0167820202
x88     -0.031784193 1.615791e-02 -0.045651750 -0.2914111750 -0.0630609211
x89     0.006667959 -1.326253e-02 0.008041251 -0.0211935329 -0.0237358882
x114    -0.195835724 -5.753400e-03 0.003613796 -0.0161274392 -0.0135119667
x115     0.362426260 -1.112610e-02 0.002618015 -0.0313557294 -0.0232258373
x116    -0.006070338 1.657623e-02 0.008796017 0.0192324084 -0.0167588677
x117    -0.017141358 1.680479e-02 -0.188532287 0.0412353457 -0.0210555054
x118    -0.004452361 -2.337885e-02 0.366855343 -0.0680596215 0.1282094943
x120    -0.045622341 -2.617020e-02 0.070478792 0.0353652211 -0.3603458199
x121     0.028380262 1.594580e-02 -0.046275097 -0.0238037144 0.2110651625
x157     0.001327385 2.635604e-02 0.004093587 -0.0097261231 -0.0011308620
x222     0.033377168 4.288100e-02 0.060401207 -0.0095613266 -0.0250951957
x223     0.029648714 -1.010775e-02 0.053061179 0.4081403370 0.0732331500
x224     0.012303975 -3.849187e-03 0.059721769 0.3511835436 0.0286876221
x249     0.381037349 -5.821472e-03 0.007478576 -0.0171771883 -0.0440575449
x250     0.369434254 -1.129780e-02 0.003846947 -0.0323508368 -0.0276100214
x251     0.102934418 3.570535e-03 0.030463609 0.0279403238 -0.1052482951
x252    -0.010226713 -2.702622e-02 0.304544031 -0.0550432234 0.0316364000
x253    -0.002360139 -2.541997e-02 0.379655631 -0.0726259408 0.1302323393
x255     0.056700038 3.692966e-02 -0.099158763 -0.0429380924 0.4337244184
x256     0.011033905 2.163842e-02 -0.105940393 -0.0282505365 0.1975384405
x292     0.001019736 2.634492e-02 0.002016765 -0.0105976044 0.0006752364
x360     0.034469007 4.472498e-02 0.061972502 -0.0095756357 -0.0312763403
x361     0.029269333 -1.111620e-02 0.051942061 0.4027888608 0.0727128650
x362     0.017188133 -5.393829e-05 0.059744040 0.3492289327 0.0313847351
x387     0.379930697 -6.122699e-03 0.008006874 -0.0166723693 -0.0464646045
x388     0.369553840 -1.130462e-02 0.003817888 -0.0323689962 -0.0275519659
x389     0.106938386 2.703920e-03 0.033640551 0.0268682738 -0.1055603210
x390    -0.010152941 -2.686535e-02 0.304192403 -0.0549047882 0.0315910563

```



**pr.out\$center** gives us the mean and **pr.out\$scale** gives us the standard deviation and **pr.out\$rotation** gives the directional cosines

```
> dim(pr.out$rotation)
[1] 53 53
> dim(pr.out$x)
[1] 1567 53
```

#### 4.5. Applying Scree plot

```
> scree.plot(data.2,type="V")
```

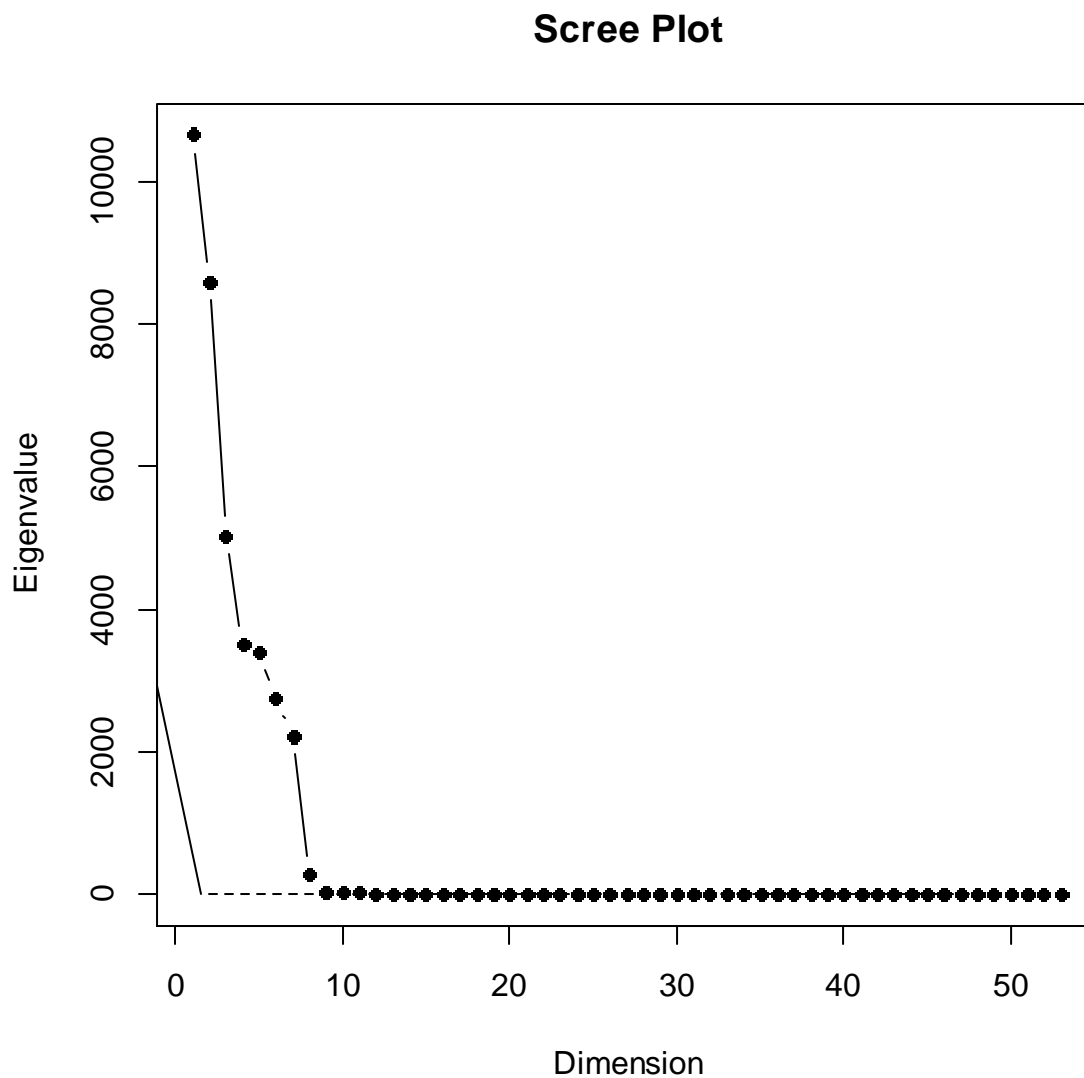


Figure 1: Scree plot

**Inference:** You can observe that no of optimal principal components is **equal to 9**

## 4.6. Applying Linear Regression Model on PCA

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

Call:
lm(formula = y ~ pr.out$x[, 1] + pr.out$x[, 2] + pr.out$x[, 3] +
    pr.out$x[, 4] + pr.out$x[, 5] + pr.out$x[, 6] + pr.out$x[,
    7] + pr.out$x[, 8] + pr.out$x[, 9], data = data.2)

Residuals:
    Min       1Q   Median       3Q      Max
-0.35853 -0.07437 -0.06616 -0.05667  0.99301

Coefficients:
              Estimate Std. Error t value Pr(>|t|)
(Intercept)   0.0663689   0.0062833   10.563  <2e-16 ***
pr.out$x[, 1]  0.0049985   0.0025950    1.926   0.0543 .
pr.out$x[, 2] -0.0056049   0.0026704   -2.099   0.0360 *
pr.out$x[, 3] -0.0003409   0.0029086   -0.117   0.9067
pr.out$x[, 4]  0.0033392   0.0029782    1.121   0.2624
pr.out$x[, 5] -0.0036860   0.0031442   -1.172   0.2412
pr.out$x[, 6]  0.0002549   0.0034212    0.074   0.9406
pr.out$x[, 7] -0.0035686   0.0035797   -0.997   0.3190
pr.out$x[, 8]  0.0025001   0.0036171    0.691   0.4896
pr.out$x[, 9] -0.0019290   0.0038009   -0.508   0.6119
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.2487 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
```

**Inference:** You can observe that **PC2** has turned out to be most significant

		<b>PC2</b>	
	Directional	Absolute	Actual considered values
	cosines	values	
x523	0.401934737	0.401934737	0.401934737
x251	0.401273596	0.401273596	0.401273596
x389	0.393978773	0.393978773	0.393978773
x494	-0.256572548	0.256572548	0.256572548
x222	-0.254748047	0.254748047	0.254748047
x360	-0.246109408	0.246109408	0.246109408
x524	-0.190089737	0.190089737	0.190089737
x252	-0.189117462	0.189117462	0.189117462
x390	-0.188727398	0.188727398	0.188727398
x118	0.149664952	0.149664952	0.149664952
x117	0.140409948	0.140409948	0.140409948
x391	0.136909636	0.136909636	0.136909636
x253	0.136760402	0.136760402	0.136760402
x525	0.123929899	0.123929899	0.123929899
x522	-0.092941695	0.092941695	0.092941695
x115	-0.091434323	0.091434323	0.091434323
x388	-0.090466699	0.090466699	0.090466699
x250	-0.090278911	0.090278911	0.090278911
x528	-0.088641135	0.088641135	0.088641135
x256	-0.088608745	0.088608745	0.088608745
x394	-0.08836722	0.08836722	0.08836722
x120	-0.08338688	0.08338688	0.08338688
x527	0.073082181	0.073082181	0.073082181
x255	0.072285782	0.072285782	0.072285782
x393	0.070741951	0.070741951	0.070741951
x292	0.070581717	0.070581717	0.070581717
x157	0.067429629	0.067429629	0.067429629
x430	0.067200844	0.067200844	0.067200844
x114	0.064324293	0.064324293	0.064324293
x87	0.060869015	0.060869015	0.060869015
x572	-0.06081353	0.06081353	0.06081353
x89	0.06067121	0.06067121	0.06067121
x361	-0.052898619	0.052898619	0.052898619
x223	-0.052735214	0.052735214	0.052735214
x495	-0.052337903	0.052337903	0.052337903
x88	0.04605091	0.04605091	0.04605091
x224	0.038430228	0.038430228	0.038430228

x362	0.037369573	0.037369573	0.037369573
x21	-0.035605442	0.035605442	0.035605442
x116	-0.033901977	0.033901977	0.033901977
x496	0.033275657	0.033275657	0.033275657
x387	0.029281062	0.029281062	0.029281062
x521	0.026958945	0.026958945	0.026958945
x249	0.026341608	0.026341608	0.026341608
y	-0.014331458	0.014331458	0.014331458
x121	-0.013023859	0.013023859	0.013023859
x578	0.0099536	0.0099536	0.0099536
x573	0.008868599	0.008868599	0.008868599
x577	0.007345318	0.007345318	0.007345318
x575	0.006921465	0.006921465	0.006921465
x576	-0.006663906	0.006663906	0.006663906
x574	-0.004229699	0.004229699	0.004229699
x571	-0.001293587	0.001293587	0.001293587

**So the variables that are considered are x523, x251, x389, x494, x222, and x360**

#### 4.7. Splitting the data into training and testing data sets

```
> train=sample(1:nrow(data.2), (nrow(data.2))/2)
> datatest=data.2[-train,]
> datatrain=data.2[train,]
> ytest=data.2[-train,1]
> ytrain=data.2[train,1]
~ |
```

## 4.8. Partial Least Squares

```
> library(pls)
> pls.fit = plsr(y~x523+x251+x389+x494+x222+x360,data = data.2,scale = TRUE,validation = "CV")
> summary(pls.fit)
Data:  X dimension: 1567 6
      Y dimension: 1567 1
Fit method: kernelpls
Number of components considered: 6

VALIDATION: RMSEP
Cross-validated using 10 random segments.
      (Intercept) 1 comps 2 comps 3 comps 4 comps 5 comps 6 comps
CV      0.2491    0.2493   0.2492   0.2489   0.2488   0.2488   0.2488
adjCV    0.2491    0.2492   0.2491   0.2489   0.2487   0.2487   0.2487

TRAINING: % variance explained
      1 comps 2 comps 3 comps 4 comps 5 comps 6 comps
X 50.28403 54.9898 99.1756 99.5690 99.9788 100.000
y  0.09145  0.7329  0.7947  0.9143  0.9394  1.013
> validationplot(pls.fit, val.type = "MSEP")
> pls.pred <- predict(pls.fit, datatest, ncomp= 2)
> mean((pls.pred-ytest)^2)
[1] 0.06137963
```

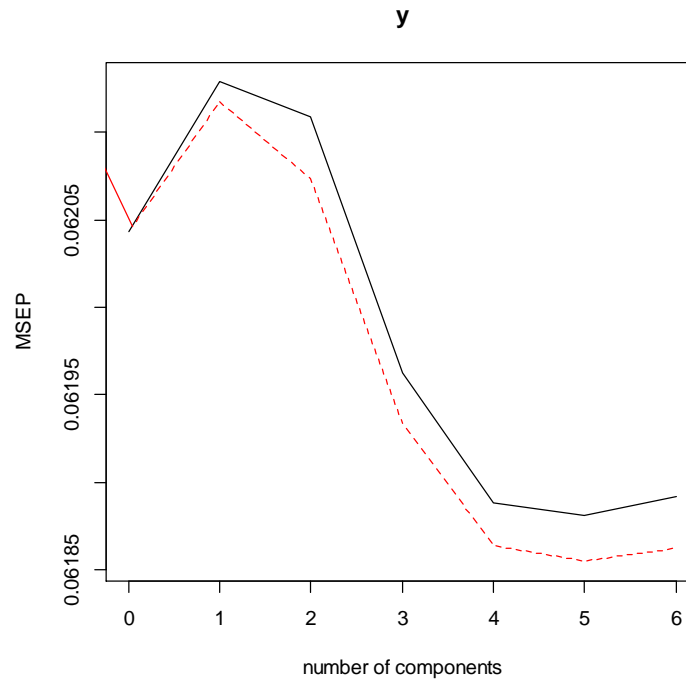


Figure 2: PLS plot for model 1

```

> library(pls)
> pls.fit = pls(y~x251+x389+x360,data = data.2,scale = TRUE,validation = "CV")
> summary(pls.fit)
Data:   X dimension: 1567 3
        Y dimension: 1567 1
Fit method: kernelpls
Number of components considered: 3

VALIDATION: RMSEP
Cross-validated using 10 random segments.
      (Intercept)  1 comps  2 comps  3 comps
CV           0.2491  0.2493  0.249  0.2489
adjCV        0.2491  0.2493  0.249  0.2489

TRAINING: % variance explained
      1 comps  2 comps  3 comps
X  47.5457  98.7812 100.0000
y   0.1261   0.1499   0.4205
> validationplot(pls.fit, val.type = "MSEP")
> pls.pred <- predict(pls.fit, datatest, ncomp= 2)
> mean((pls.pred-ytest)^2)
[1] 0.06190335

```

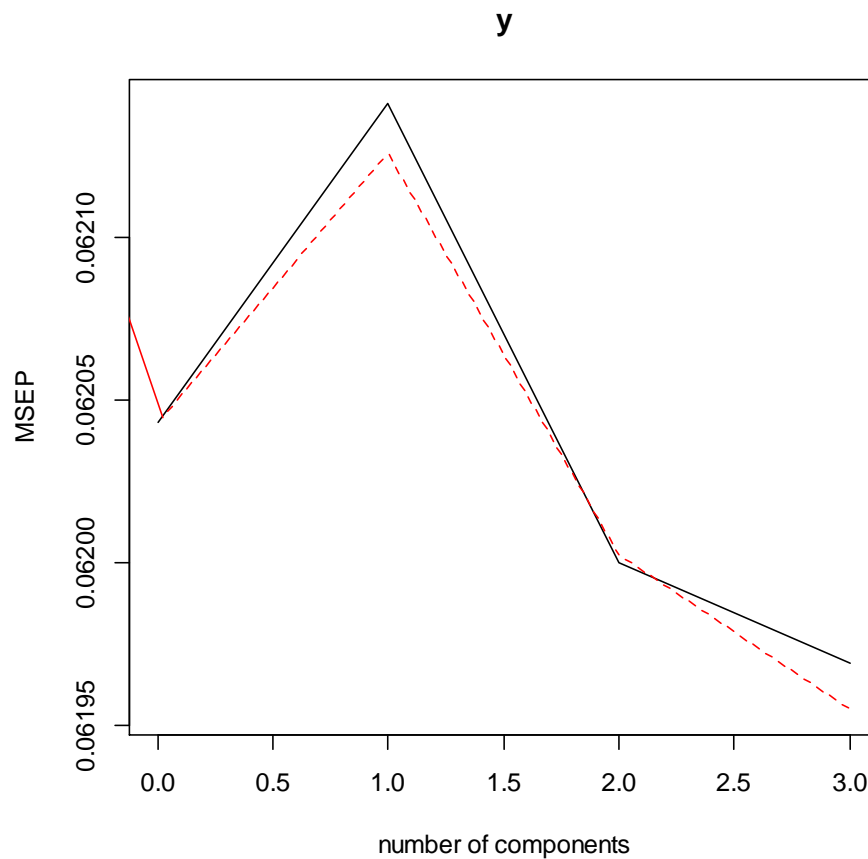


Figure 3: PLS plot for model 2

```

> library(pls)
> pls.fit = plsr(y~x251+x389,data = data.2,scale = TRUE,validation = "CV")
> summary(pls.fit)
Data:   X dimension: 1567 2
        Y dimension: 1567 1
Fit method: kernelpls
Number of components considered: 2

VALIDATION: RMSEP
Cross-validated using 10 random segments.
      (Intercept)  1 comps  2 comps
CV           0.2491  0.2491  0.2489
adjCV        0.2491  0.2491  0.2489

TRAINING: % variance explained
      1 comps  2 comps
X  98.44522 100.0000
y   0.02714  0.3386
> validationplot(pls.fit, val.type = "MSEP")
> pls.pred <- predict(pls.fit, datatest, ncomp= 2)
> mean((pls.pred-ytest)^2)
[1] 0.06161501

```

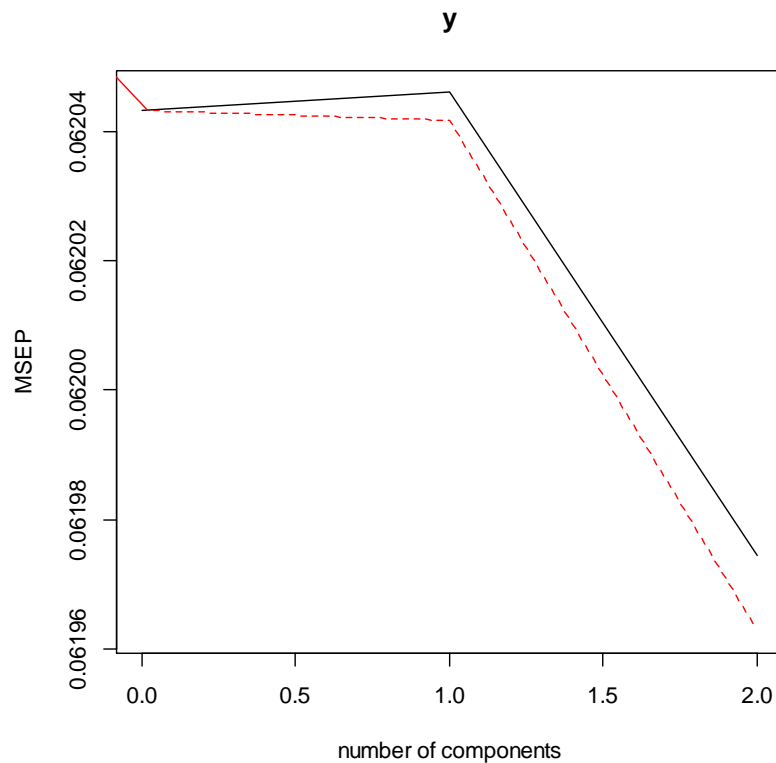


Figure 4: PLS plot for model 3

## 4.9. CLASSIFICATION MODELS

### 4.9.1. LOGISTIC REGRESSION MODEL

#### MODEL 1:

```
> #logistic regression model
> train=sample(1:nrow(data.2), (nrow(data.2))/2)
> datatest=data.2[-train,]
> ytest=data.2[-train,1]
> glm.fit=glm(y~x523+x251+x389+x494+x222+x360,data=data.2,family=binomial)
> summary(glm.fit)

Call:
glm(formula = y ~ x523 + x251 + x389 + x494 + x222 + x360, family = binomial,
    data = data.2)

Deviance Residuals:
    Min       1Q   Median       3Q      Max
-0.7923  -0.3906  -0.3519  -0.3172   2.5850

Coefficients:
              Estimate Std. Error z value Pr(>|z|)
(Intercept)  -3.08662    0.35451  -8.707  <2e-16 ***
x523           0.12141    0.08422   1.442   0.1494
x251          -0.03053    0.01317  -2.318   0.0205 *
x389           0.04746    0.02502   1.897   0.0578 .
x494          -3.19985    2.28052  -1.403   0.1606
x222          96.75651   99.12172   0.976   0.3290
x360         126.01427   63.03429   1.999   0.0456 *
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 765.15  on 1566  degrees of freedom
Residual deviance: 751.16  on 1560  degrees of freedom
AIC: 765.16

Number of Fisher Scoring iterations: 5
```



```

> glm.probs=predict(glm.fit,datatest,type="response")
> glm.probs[1:10]
      4      5      6      9     10     11     13
0.08149219 0.14769677 0.09616417 0.04690543 0.06636887 0.15094280 0.08713491
      14     15     16
0.03609757 0.11793219 0.13682556
> 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

```

## MODEL 2:

```

> glm.fit=glm(y~x251+x389+x360,data=data.2,family=binomial)
> summary(glm.fit)

Call:
glm(formula = y ~ x251 + x389 + x360, family = binomial, data = data.2)

Deviance Residuals:
      Min       1Q   Median       3Q      Max
-0.6205  -0.3880  -0.3578  -0.3309   2.4761

Coefficients:
              Estimate Std. Error z value Pr(>|z|)
(Intercept) -3.076113    0.348933  -8.816  <2e-16 ***
x251         -0.016384    0.008014  -2.044   0.0409 *
x389          0.054075    0.024921   2.170   0.0300 *
x360         15.189699   13.503401   1.125   0.2606
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 765.15  on 1566  degrees of freedom
Residual deviance: 758.87  on 1563  degrees of freedom
AIC: 766.87

Number of Fisher Scoring iterations: 5

> glm.probs=predict(glm.fit,datatest,type="response")
> glm.probs[1:10]
      1      3      4      7     11     15     17
0.04736722 0.05873270 0.07714591 0.10333410 0.10212534 0.07985020 0.07530311
      19     20     21
0.08051504 0.11232660 0.07052045
> glm.pred=rep("No",length(datatest))
> glm.pred[glm.probs>0.5]="Yes"
> table(glm.pred,ytest)
      ytest
glm.pred 0  1
      No 45  8

```

### MODEL 3:

```
> glm.fit=glm(y~x251+x389,data=data.2,family=binomial)
> summary(glm.fit)

Call:
glm(formula = y ~ x251 + x389, family = binomial, data = data.2)

Deviance Residuals:
    Min       1Q   Median       3Q      Max
-0.6015  -0.3860  -0.3589  -0.3364   2.4865

Coefficients:
            Estimate Std. Error z value Pr(>|z|)
(Intercept) -2.780109   0.224867  -12.363  <2e-16 ***
x251         -0.017026   0.008004   -2.127   0.0334 *
x389          0.056374   0.024881    2.266   0.0235 *
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 765.15  on 1566  degrees of freedom
Residual deviance: 760.09  on 1564  degrees of freedom
AIC: 766.09

Number of Fisher Scoring iterations: 5

> glm.probs=predict(glm.fit,datatest,type="response")
> glm.probs[1:10]
      1      3      4      7     11     15     17
0.04510178 0.05031163 0.07213193 0.08528115 0.09028781 0.06998835 0.06814977
     19     20     21
0.07243556 0.10599314 0.06356210
> glm.pred=rep("No",length(datatest))
> glm.pred[glm.probs>0.5]="Yes"
> table(glm.pred,ytest)
      ytest
glm.pred 0  1
      No 45  8
```

## 4.9.2. CROSS VALIDATION for Logistic Regression

### Cross validation for the model 1:

```
> library(boot)
> set.seed(12)
> cv.error.10= rep(0 ,10)
> for (i in 1:10) {
+   glm.fit=glm(y~x523+x251+x389+x494+x222+x360,data=data.2,family=binomial)
+   cv.error.10[i]=cv.glm(data.2,glm.fit,K=10)$delta[1]
+ }
> cv.error.10
[1] 0.06202428 0.06198660 0.06187209 0.06196317 0.06192981 0.06214584
[7] 0.06175791 0.06193640 0.06196005 0.06175740
> min(cv.error.10)
[1] 0.0617574
```

**Inference: The classification error rate is 6.17%**

### Cross validation for the model 2:

```
> #CROSS VALIDATION FOR LOGISTIC MODEL
> library(boot)
> set.seed(12)
> cv.error.10= rep(0 ,10)
> for (i in 1:10) {
+ glm.fit=glm(y~x251+x389+x360,data=data.2,family=binomial)
+ cv.error.10[i]=cv.glm(data.2,glm.fit,K=10)$delta[1]
+ }
> cv.error.10
[1] 0.06220045 0.06218119 0.06213588 0.06213454 0.06200871 0.06217668
[7] 0.06207442 0.06214757 0.06214692 0.06203628
> min(cv.error.10)
[1] 0.06200871
```

**Inference: The classification error rate is 6.2%**

### Cross validation for the model 3:

```
>
> #CROSS VALIDATION FOR LOGISTIC MODEL
> library(boot)
> set.seed(12)
> cv.error.10= rep(0 ,10)
> for (i in 1:10) {
+ glm.fit=glm(y~x251+x389,data=data.2,family=binomial)
+ cv.error.10[i]=cv.glm(data.2,glm.fit,K=10)$delta[1]
+ }
> cv.error.10
[1] 0.06210278 0.06201635 0.06198448 0.06213756 0.06195369 0.06199276
[7] 0.06192631 0.06206881 0.06199603 0.06192079
> min(cv.error.10)
[1] 0.06192079
```

**Inference: The classification error rate is 6.19%**

## CONCLUSIONS FROM LOGISTIC REGRESSION

MODEL	CROSS VALIDATION ERROR RATE
MODEL 1	6.17%
MODEL 2	6.2%
MODEL 3	6.19%

Table 1: Error rates of different models for logistic regression

### 4.9.3. ROC CURVE FOR BEST MODEL in Logistic Regression

```
> glm.fit=glm(y~x523+x251+x389+x494+x222+x360,data=data.2,family=binomial)  
> rocplot(glm.fit)
```

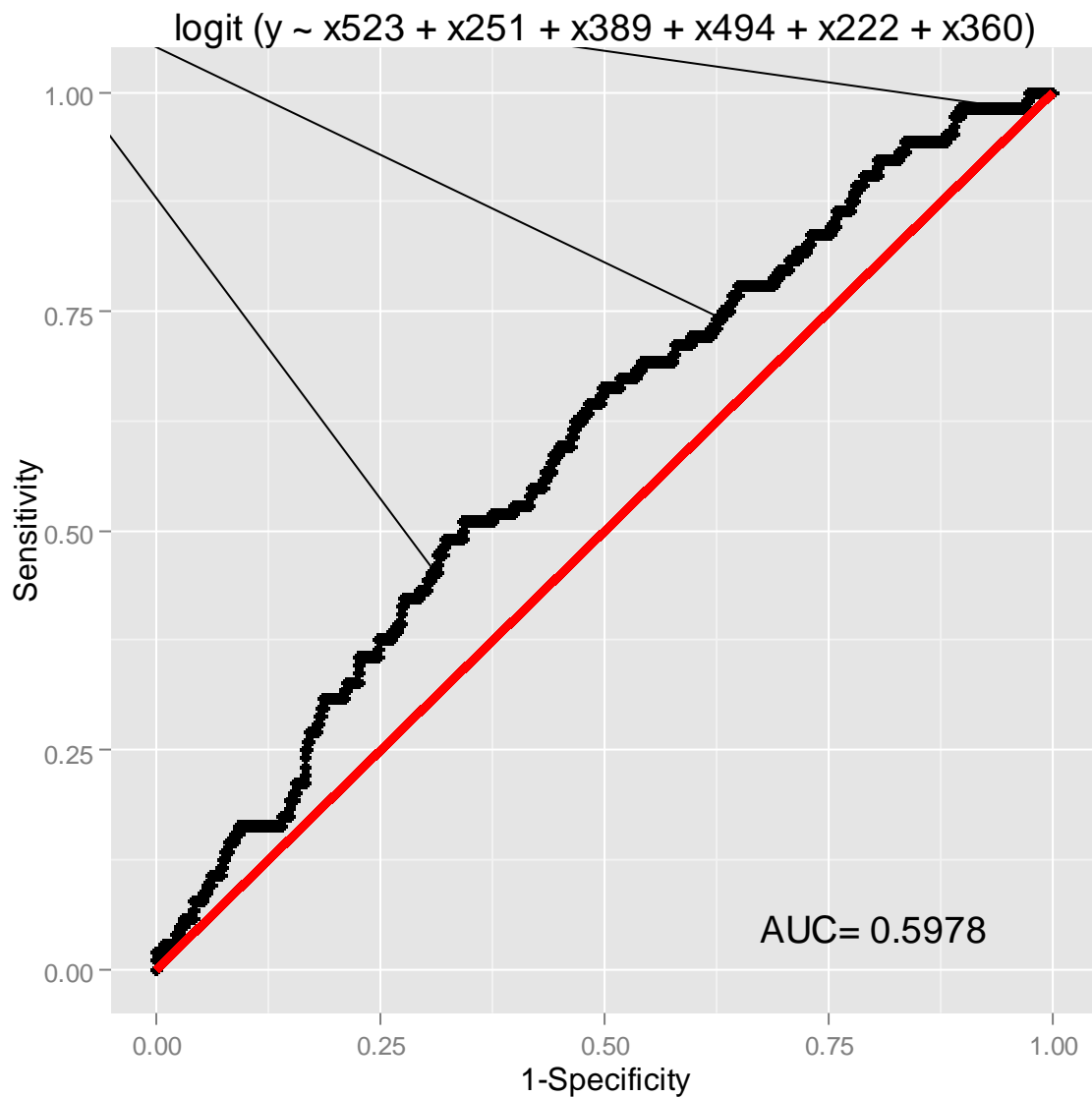


Figure 5: ROC curve for logistic regression

**Area Under the curve is 60%**

## 4.9.4. LINEAR DISCRIMINANT ANALYSIS

### MODEL 1:

```
> library(ISLR)
> library(MASS)
> lda.fit=lda(y~x523+x251+x389+x494+x222+x360,data=data.2,family=binomial)
> lda.fit
Call:
lda(y ~ x523 + x251 + x389 + x494 + x222 + x360, data = data.2,
    family = binomial)

Prior probabilities of groups:
      0      1
0.93363114 0.06636886

Group means:
      x523      x251      x389      x494      x222      x360
0 14.70297 109.5496 35.06453 2.525005 0.06058332 0.01977703
1 15.09318 111.0766 36.42900 2.600964 0.06260577 0.02072500

Coefficients of linear discriminants:
      LD1
x523  0.32418184
x251 -0.08052313
x389  0.12815252
x494 -8.74382227
x222 250.13819874
x360 407.21197694
> lda_pred=predict(lda.fit,datatest)
> names(lda_pred)
[1] "class"      "posterior"  "x"
> lda.class=lda_pred$class
> table(lda.class,ytest)
      ytest
lda.class  0      1
      0 733    50
      1   0     1
> mean(lda.class!=ytest)
[1] 0.06377551
```

**Inference: The classification error rate is 6.37%**

### MODEL 2:

```
> lda.fit=lda(y~x251+x389+x360,data=data.2,family=binomial)
> lda.fit
Call:
lda(y ~ x251 + x389 + x360, data = data.2, family = binomial)

Prior probabilities of groups:
      0      1
0.93363114 0.06636886

Group means:
      x251      x389      x360
0 109.5496 35.06453 0.01977703
1 111.0766 36.42900 0.02072500

Coefficients of linear discriminants:
      LD1
x251 -0.06668174
x389  0.22265645
x360 62.11560457
> lda_pred=predict(lda.fit,datatest)
> names(lda_pred)
[1] "class"      "posterior"  "x"
> lda.class=lda_pred$class
> table(lda.class,ytest)
      ytest
lda.class  0      1
      0 733    51
      1   0     0
> mean(lda.class!=ytest)
[1] 0.06505102
```

**Inference: The classification error rate is 6.5%**

### MODEL 3:

```
> lda.fit=lda(y~x251+x389,data=data.2,family=binomial)
> lda.fit
Call:
lda(y ~ x251 + x389, data = data.2, family = binomial)

Prior probabilities of groups:
          0          1
0.93363114 0.06636886

Group means:
      x251      x389
0 109.5496 35.06453
1 111.0766 36.42900

Coefficients of linear discriminants:
      LD1
x251 -0.07707309
x389  0.25776628
> lda_pred=predict(lda.fit,datatest)
> names(lda_pred)
[1] "class"      "posterior" "x"
> lda.class=lda_pred$class
> table(lda.class,ytest)
      ytest
lda.class  0    1
      0 733   51
      1   0    0
> mean(lda.class!=ytest)
[1] 0.06505102
```

**Inference:** The classification error rate is 6.5%

#### 4.9.5. CROSS VALIDATION for Linear Discriminant Analysis

##### Cross validation for the model 1:

```
> #CROSS VALIDATION FOR LDA
> library(Discriminer)
> lda.cv=linDA(data[,c(524,252,390,495,223,361)],y,validation="crossval")
> lda.cv$confusion
      predicted
original    0    1
      0 1463    0
      1  103    1
> lda.cv$error_rate
[1] 0.06764518
```

**Inference:** The classification error rate is 6.7%

### Cross validation for the model 2:

```
> library(Discriminer)
> lda.cv=linDA(data[,c(252,390,361)],y,validation="crossval")
> lda.cv$confusion
      predicted
original    0    1
      0 1463    0
      1  104    0
> lda.cv$error_rate
[1] 0.06828334
```

**Inference:** The classification error rate is 6.8%

### Cross validation for the model 3:

```
> #CROSS VALIDATION FOR LDA
> library(Discriminer)
> lda.cv=linDA(data[,c(252,390)],y,validation="crossval")
> lda.cv$confusion
      predicted
original    0    1
      0 1463    0
      1  104    0
> lda.cv$error_rate
[1] 0.06828334
```

**Inference:** The classification error rate is 6.8%

### CONCLUSIONS FROM LDA:

MODEL	PREDICTORS	ERROR RATE	CV ERROR RATE
MODEL 1	x523,x251,x389,x494,x222,x360	6.37%	6.7%
MODEL 2	x251,x389,x360	6.5%	6.8%
MODEL 3	x251,x389	6.5%	6.8%

Table 2: Error rates of different models for Linear Discriminant Analysis

**Inference :** Model 1 is the best model

## 4.9.6. ROC CURVE FOR BEST MODEL in Linear Discriminant Analysis

```
> true.class<-datatest[,1]
> cmp.lda<-lda(y~x523+x251+x389+x494+x222+x360,data=data.2,family=binomial)
> out.lda<-predict(cmp.lda,datatest)
> class.lda<-out.lda$class
> scores.lda<-out.lda$posterior[,1]
> lda.counts<-misclassCounts(class.lda,true.class)
> lda.counts$conf.matrix
      pred.1 pred.0
actual.1    1    51
actual.0    0   732
> lda.counts.T03<-misclassCounts(scores.lda>0.5,true.class)
> lda.counts.T03$conf.matrix
      pred.1 pred.0
actual.1    51    1
actual.0   732    0
> lda.counts.T03$metrics[c('Sens','Spec')]
      Sens Spec
1 0.9807692    0
> scores<-data.frame(LDA=scores.lda)
> results<-Hmeasure(true.class,scores)
Error: could not find function "Hmeasure"
> results<-HMeasure(true.class,scores)
Warning message:
In HMeasure.single(y = true.class, s = s, classifier.name = name.now, :
  ROC curve of LDA mostly lying under the diagonal. Switching scores.
> summary(results,show.all=TRUE)
      H      Gini      AUC      AUCH      KS      MER      MWL
LDA 0.05622695 0.1169084 0.5584542 0.600331 0.1240017 0.06377551 0.1084965
Spec.Sens95 Sens.Spec95      ER      Sens Spec Precision Recall
LDA 0.1338798 0.05769231 0.06505102 0.01923077 1 1 0.01923077
      TPR FPR      F      Youden TP FP TN FN
LDA 0.01923077 0 0.03773585 0.01923077 1 0 732 51
> plotROC(results,which=1)
```

ROC (continuous) and ROCH (dotted)

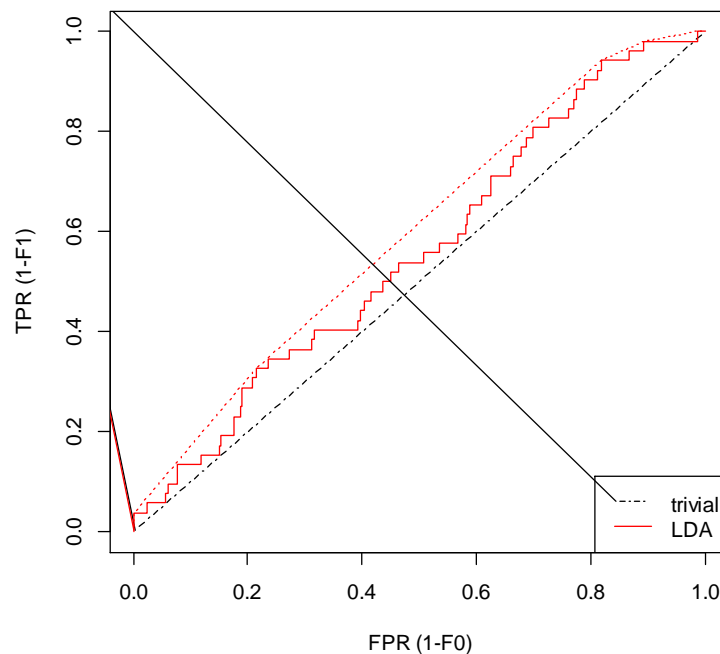


Figure 6: ROC curve for Linear Discriminant Analysis

**AREA UNDER THE CURVE IS 55.8%**



## 4.9.7. QUADRATIC DISCRIMINANT ANALYSIS

### MODEL 1:

```
> library(MASS)
> qda.fit=qda(y~x523+x251+x389+x494+x222+x360,data=data.2,family=binomial)
> qda.fit
Call:
qda(y ~ x523 + x251 + x389 + x494 + x222 + x360, data = data.2,
    family = binomial)

Prior probabilities of groups:
      0      1
0.93363114 0.06636886

Group means:
      x523      x251      x389      x494      x222      x360
0 14.70297 109.5496 35.06453 2.525005 0.06058332 0.01977703
1 15.09318 111.0766 36.42900 2.600964 0.06260577 0.02072500
> qda.pred=predict(qda.fit,datatest)
> qdaclass=qda.pred$class
> table(qdaclass,ytest)
      ytest
qdaclass 0    1
      0 728  49
      1   5   2
> mean(qdaclass!=ytest)
[1] 0.06887755
```

**Inference: The classification error rate is 6.88%**

### MODEL 2:

```
> library(MASS)
> qda.fit=qda(y~x251+x389+x360,data=data.2,family=binomial)
> qda.fit
Call:
qda(y ~ x251 + x389 + x360, data = data.2, family = binomial)

Prior probabilities of groups:
      0      1
0.93363114 0.06636886

Group means:
      x251      x389      x360
0 109.5496 35.06453 0.01977703
1 111.0766 36.42900 0.02072500
> qda.pred=predict(qda.fit,datatest)
> qdaclass=qda.pred$class
> table(qdaclass,ytest)
      ytest
qdaclass 0    1
      0 730  49
      1   3   2
> mean(qdaclass!=ytest)
[1] 0.06632653
```

**Inference: The classification error rate is 6.66%**

### MODEL 3:

```
> library(MASS)
> qda.fit=qda(y~x251+x389,data=data.2,family=binomial)
> qda.fit
Call:
qda(y ~ x251 + x389, data = data.2, family = binomial)

Prior probabilities of groups:
      0      1
0.93363114 0.06636886

Group means:
      x251      x389
0 109.5496 35.06453
1 111.0766 36.42900
> qda.pred=predict(qda.fit,datatest)
> qda.class=qda.pred$class
> table(qda.class,ytest)
      ytest
qda.class 0  1
0    733  51
1     0   0
> mean(qda.class!=ytest)
[1] 0.06505102
.
```

**Inference: The classification error rate is 6.5%**

## 4.9.8. CROSS VALIDATION for Quadratic Discriminant Analysis

### Cross Validation for Model 1

```
> library(Discriminer)
> qda.cv=quaDA(data[,c(524,252,390,495,223,361)],y,validation="crossval")
> qda.cv$confusion
      predicted
original  0    1
0    1444    19
1     101     3
> qda.cv$error_rate
[1] 0.08168475
.
```

**Inference: The classification error rate is 8.16%**

### Cross Validation for Model 2

```
> library(Discriminer)
> qda.cv=quaDA(data[,c(252,390,361)],y,validation="crossval")
> qda.cv$confusion
      predicted
original  0    1
0    1453    10
1     101     3
> qda.cv$error_rate
[1] 0.06955967
.
```

**Inference: The classification error rate is 6.95%**

### Cross Validation for Model 3

```
> #CROSS VALIDATION
> library(Discriminer)
> qda.cv=quaDA(data[,c(252,390)],y,validation="crossval")
> qda.cv$confusion
      predicted
original  0    1
      0 1462    1
      1  103    1
> qda.cv$error_rate
[1] 0.06828334
```

**Inference: The classification error rate is 6.82%**

### CONCLUSIONS FROM QDA:

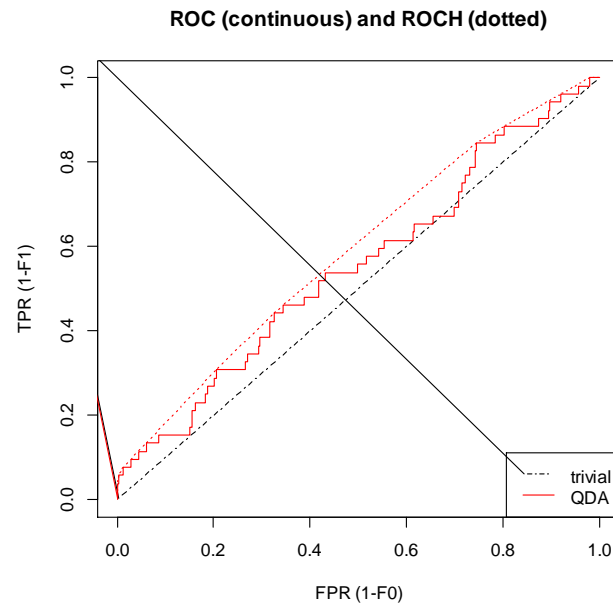
MODEL	PREDICTORS	ERROR RATE	CV ERROR RATE
MODEL 1	x523,x251,x389,x494,x222,x360	6.88 %	8.16%
MODEL 2	x251,x389,x360	6.66 %	6.95%
MODEL 3	x251,x389	6.5 %	6.8%

Table 3: Error rates of different models for Quadratic Discriminant Analysis

**Inference : Model 3 is the Best**

#### 4.9.9. ROC CURVE FOR THE BEST MODEL in Quadratic Discriminant Analysis

```
> true.class<-datatest[,1]
> cmp.qda<-qda(y~x523+x251+x389+x494+x222+x360,data=data.2,family=binomial)
> out.qda<-predict(cmp.qda,datatest)
> class.qda<-out.qda$class
> scores.qda<-out.qda$posterior[,1]
> qda.counts<-misclassCounts(class.qda,true.class)
> qda.counts$conf.matrix
      pred.1 pred.0
actual.1     2    50
actual.0     8   724
> qda.counts.T03<-misclassCounts(scores.qda>0.5,true.class)
> qda.counts.T03$conf.matrix
      pred.1 pred.0
actual.1    50     2
actual.0   724     8
> qda.counts.T03$metrics[c('Sens','Spec')]
      Sens      Spec
1 0.9615385 0.01092896
> scores<-data.frame(QDA=scores.qda)
> results<-HMeasure(true.class,scores)
Warning message:
In HMeasure.single(y = true.class, s = s, classifier.name = name.now, :
  ROC curve of QDA mostly lying under the diagonal. Switching scores.
> summary(results,show.all=TRUE)
      H      Gini      AUC      AUCH      KS      MER      MWL
QDA 0.09417874 0.2448508 0.6224254 0.6643679 0.2079655 0.06505102 0.09809715
      Spec.Sens95 Sens.Spec95      ER      Sens      Spec Precision
QDA 0.192623 0.1346154 0.07397959 0.03846154 0.989071 0.2
      Recall      TPR      FPR      F      Youden TP FP TN FN
QDA 0.03846154 0.03846154 0.01092896 0.06451613 0.02753258 2 8 724 50
. |
```



**Figure 7: ROC Curve for Quadratic Discriminant Analysis**

**AREA UNDER THE CURVE IS 62.2%**

## 4.9.10. K-Nearest Neighbors

### MODEL 1: (K=1)

```
> library(class)
> library(ISLR)
> library(MASS)
> datatest=data.2[-train,]
> datatrain=data.2[train,]
> ytest=data.2[-train,1]
> ytrain=data.2[train,1]
> set.seed(1)
> knnpred=knn(datatrain,datatest,y[train],k=1)
> table(knnpred,ytest)
      ytest
knnpred  0   1
      0 677  44
      1  55   7
> mean(knnpred!=ytest)
[1] 0.1264368
```

**Inference: The classification error rate is 12.6%**

### MODEL 2: (K=2)

```
> datatest=data.2[-train,]
> datatrain=data.2[train,]
> ytest=data.2[-train,1]
> ytrain=data.2[train,1]
> set.seed(1)
> knnpred=knn(datatrain,datatest,y[train],k=2)
> table(knnpred,ytest)
      ytest
knnpred  0   1
      0 670  46
      1  62   5
> mean(knnpred!=ytest)
[1] 0.137931
```

**Inference: The classification error rate is 13.79%**

### MODEL 3: (K=3)

```
> set.seed(1)
> knnpred=knn(datatrain,datatest,y[train],k=3)
> table(knnpred,ytest)
      ytest
knnpred  0   1
      0 721  51
      1  11   0
> mean(knnpred!=ytest)
[1] 0.07918263
```

**Inference: The classification error rate is 7.9%**

#### 4.9.11. CROSS VALIDATION for kNN

##### Cross Validation for Model 1 (K=1)

```
> #CROSS VALIDATION KNN
> knncv=knn.cv(datatrain,ytrain,k=1,l=0, prob = FALSE, use.all = TRUE)
> table(knncv,ytest)
      ytest
knncv  0   1
      0 685  49
      1  47   2
> mean(knncv!=ytest)
[1] 0.1226054
```

**Inference: The classification error rate is 12.2%**

##### Cross Validation for Model 2(K=2)

```
> knncv=knn.cv(datatrain,ytrain,k=2,l=0, prob = FALSE, use.all = TRUE)
> table(knncv,ytest)
      ytest
knncv  0   1
      0 695  46
      1  37   5
> mean(knncv!=ytest)
[1] 0.1060026
```

**Inference: The classification error rate is 10.6%**

##### Cross Validation for Model 3(K=3)

```
> #CROSS VALIDATION KNN
> knncv=knn.cv(datatrain,ytrain,k=3,l=0, prob = FALSE, use.all = TRUE)
> table(knncv,ytest)
      ytest
knncv  0   1
      0 724  50
      1   8   1
> mean(knncv!=ytest)
[1] 0.07407407
```

**Inference: The classification error rate is 7.4%**

## CONCLUSIONS FROM KNN

MODEL	K VALUE	ERROR RATE	CV ERROR RATE
MODEL 1	K=1	12.6 %	12.2%
MODEL 2	K=2	13.79 %	10.6%
MODEL 3	K=3	7.9 %	7.4%

Table 4: Error rates of different models for kNN

**Inference : Model 3 is the Best**

### 4.9.12. ROC CURVE FOR kNN

```
> class.knn<-knn(datatrain,datatest,ytrain,k=3,prob=TRUE,use.all=TRUE)
> scores.knn<-attr(class.knn,"prob")
> scores.knn[class.knn=="No"]<-1-scores.knn[class.knn=="No"]
> results<-HMeasure(true.class,scores.knn)
> summary(results,show.all=TRUE)
```

	H	Gini	AUC	AUCH	KS	MER
scores	0.00602082	0.05821774	0.5291089	0.5291089	0.05821774	0.06632653

	MWL	Spec.Sens95	Sens.Spec95	ER	Sens	Spec	Precision	Recall
scores	0.1166441	0.07409836	0.8504039	0.9336735	1	0	0.06632653	1

	TPR	FPR	F	Youden	TP	FP	TN	FN
scores	1	1	0.1244019	0	52	732	0	0

```
> plotROC(results,which=1)
. |
```

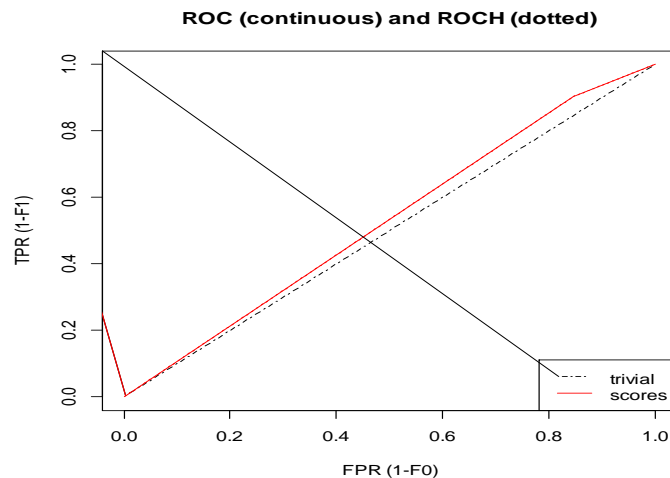


Figure 8: ROC Curve for kNN

**AREA UNDER THE CURVE IS 52.9 %**

### 4.9.13. CART MODEL

#### MODEL 1:

```
> library(ISLR)
> Pass=ifelse(y==0,"Yes","No")
> data.3=data.frame(data.2,Pass)
> train=sample(1:nrow(data.3),(nrow(data.3))/2)
> datatrain=data.3[train,]
> datatest=data.3[-train,]
> Pass.test= datatest$Pass
> tree.Response = tree(Pass~x523+x251+x389+x494+x222+x360,datatrain)
> summary(tree.Response)

Classification tree:
tree(formula = Pass ~ x523 + x251 + x389 + x494 + x222 + x360,
      data = datatrain)
Number of terminal nodes: 33
Residual mean deviance: 0.2199 = 165 / 750
Misclassification error rate: 0.04853 = 38 / 783
> tree.pred = predict(tree.Response,datatest,type="class")
> table(Pass.test,tree.pred)
      tree.pred
Pass.test No Yes
      No    2  54
      Yes   30 698
> mean(Pass.test!= tree.pred)
[1] 0.1071429
```

#### MODEL 2:

```
> Pass=ifelse(y==0,"Yes","No")
> data.3=data.frame(data.2,Pass)
> train=sample(1:nrow(data.3),(nrow(data.3))/2)
> datatrain=data.3[train,]
> datatest=data.3[-train,]
> Pass.test= datatest$Pass
> tree.Response = tree(Pass~x251+x389+x360,datatrain)
> summary(tree.Response)

Classification tree:
tree(formula = Pass ~ x251 + x389 + x360, data = datatrain)
Number of terminal nodes: 26
Residual mean deviance: 0.2966 = 224.5 / 757
Misclassification error rate: 0.05875 = 46 / 783
> tree.pred = predict(tree.Response,datatest,type="class")
> table(Pass.test,tree.pred)
      tree.pred
Pass.test No Yes
      No    1  55
      Yes   27 701
> mean(Pass.test!= tree.pred)
[1] 0.1045918
```



### MODEL 3:

```
> Pass=ifelse(y==0,"Yes","No")
> data.3=data.frame(data.2,Pass)
> train=sample(1:nrow(data.3),(nrow(data.3))/2)
> datatrain=data.3[train,]
> datatest=data.3[-train,]
> Pass.test= datatest$Pass
> tree.Response = tree(Pass~x251+x389,datatrain)
> summary(tree.Response)

Classification tree:
tree(formula = Pass ~ x251 + x389, data = datatrain)
Number of terminal nodes: 12
Residual mean deviance: 0.4244 = 327.2 / 771
Misclassification error rate: 0.06641 = 52 / 783
> tree.pred = predict(tree.Response,datatest,type="class")
> table(Pass.test,tree.pred)
      tree.pred
Pass.test No Yes
      No    2  46
      Yes   14 722
> mean(Pass.test!= tree.pred)
[1] 0.07653061
```

### 4.9.14. CROSS VALIDATION for CART

#### CROSS VALIDATION FOR MODEL 1:

```
> cv.cmp=cv.tree(tree.Response ,FUN=prune.misclass)
> par(mfrow=c(1,2))
> plot(cv.cmp$size, cv.cmp$dev, type="b")
> plot(cv.cmp$k, cv.cmp$dev, type="b")
> prune.cmp=prune.misclass(tree.Response,best=9)
> plot(prune.cmp)
> text(prune.cmp,pretty=0)
> #Using Cross Validation
> set.seed(3)
> cv.cmp=cv.tree(tree.Response ,FUN=prune.misclass)
> names(cv.cmp)
[1] "size" "dev" "k" "method"
> cv.cmp
$size
[1] 33 17 1

$dev
[1] 60 60 49

$k
[1] -Inf 0.000 0.625

$method
[1] "misclass"

attr(,"class")
[1] "prune" "tree.sequence"
> par(mfrow=c(1,2))
> plot(cv.cmp$size, cv.cmp$dev, type="b")
> plot(cv.cmp$k, cv.cmp$dev, type="b")
> prune.cmp=prune.misclass(tree.Response,best=9)
> plot(prune.cmp)
> text(prune.cmp,pretty=0)
> tree.pred=predict(prune.cmp,datatest,type="class")
> table(tree.pred,Pass.test)
      Pass.test
tree.pred No Yes
      No    1  26
      Yes   55 702
> mean(tree.pred!=Pass.test)
[1] 0.1033163
```



## CROSS VALIDATION FOR MODEL 2:

```
> #Using Cross Validation
> set.seed(3)
> cv.cmp=cv.tree(tree.Response ,FUN=prune.misclass)
> names(cv.cmp)
[1] "size"    "dev"      "k"        "method"
> cv.cmp
$size
[1] 26  9  1

$dev
[1] 53 53 51

$k
[1] -Inf 0.00 0.25

$method
[1] "misclass"

attr(,"class")
[1] "prune"      "tree.sequence"
> par(mfrow=c(1,2))
> plot(cv.cmp$size,cv.cmp$dev,type="b")
> plot(cv.cmp$k,cv.cmp$dev,type="b")
> prune.cmp=prune.misclass(tree.Response,best=9)
> plot(prune.cmp)
> text(prune.cmp,pretty=0)
> tree.pred=predict(prune.cmp,datatest,type="class")
> table(tree.pred,Pass.test)
      Pass.test
tree.pred  No  Yes
      No    1  18
      Yes   55 710
> mean(tree.pred!=Pass.test)
```

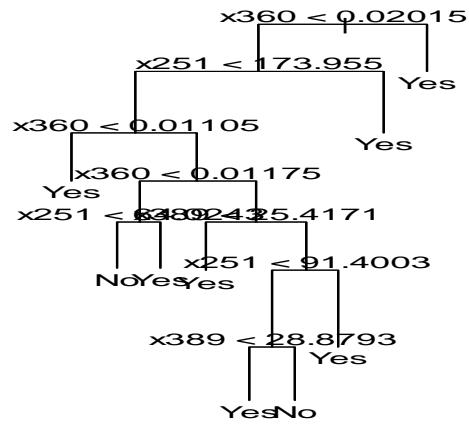
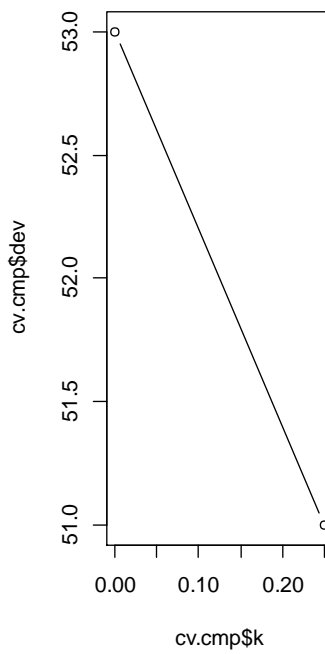
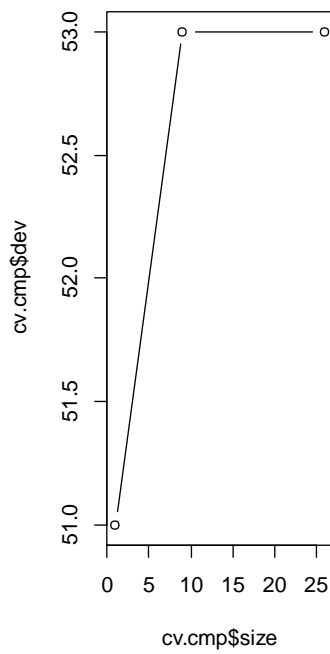


Figure 10: Classification tree for model2



## CROSS VALIDATION FOR MODEL 3

```
> #Using Cross Validation
> set.seed(3)
> cv.cmp=cv.tree(tree.Response ,FUN=prune.misclass)
> names(cv.cmp)
[1] "size"    "dev"      "k"        "method"
> cv.cmp
$size
[1] 12 11  1

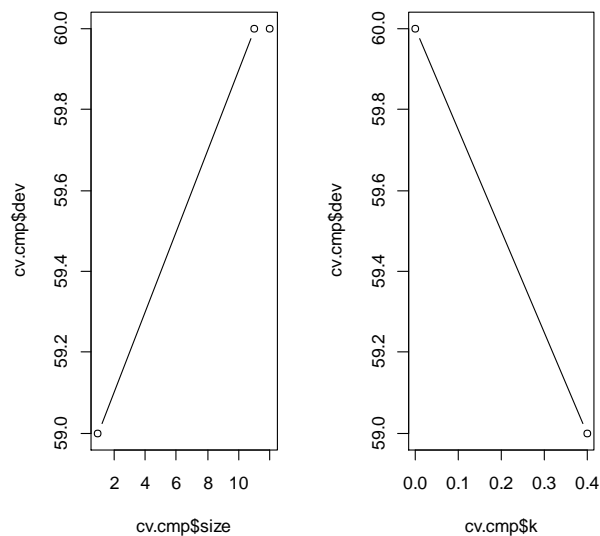
$dev
[1] 60 60 59

$sk
[1] -Inf  0.0  0.4

$method
[1] "misclass"

attr(,"class")
[1] "prune"      "tree.sequence"
> par(mfrow=c(1,2))
> plot(cv.cmp$size,cv.cmp$dev,type="b")
> plot(cv.cmp$k,cv.cmp$dev,type="b")
> prune.cmp=prune.misclass(tree.Response,best=9)
> plot(prune.cmp)
> text(prune.cmp,pretty=0)
> tree.pred=predict(prune.cmp,datatest,type="class")
> table(tree.pred,Pass.test)
      Pass.test
tree.pred  No  Yes
      No     2  14
      Yes   46 722
> mean(tree.pred!=Pass.test)
[1] 0.07653061
> summary(prune.cmp)

Classification tree:
snip.tree(tree = tree.Response, nodes = 228L)
Number of terminal nodes:  11
Residual mean deviance:  0.4293 = 331.4 / 772
Misclassification error rate: 0.06641 = 52 / 783
```



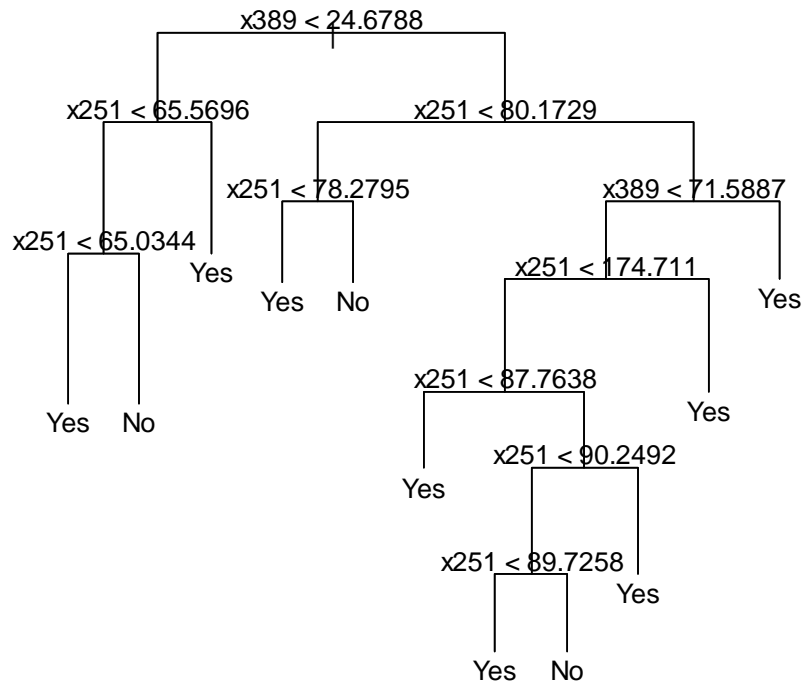


Figure 11: Classification tree for model 3

## CONCLUSIONS FROM CART MODEL:

MODEL	PREDICTORS	ERROR RATE	CV ERROR RATE
MODEL 1	x523,x251,x389,x494,x222,x360	10.7	10.3
MODEL 2	x251,x389,x360	10.4	10.2
MODEL 3	x251,x389	7.65	7.65

Table 5: Error rates of different models for kNN

**Inference : Model 3 is the Best**

#### 4.9.15. ROC FOR CART MODEL

```
> roc.curve=function(threshold,print=FALSE)
+ { Ps=(S>threshold)*1
+ FP=sum((Ps==1)*(Pass.test==0))/sum(Pass.test==0)
+ TP=sum((Ps==1)*(Pass.test==1))/sum(Pass.test==1)
+ if(print==TRUE){
+ print(table(Observed=Pass.test,Predicted=tree.pred))
+ }
+ vect=c(FP,TP)
+ names(vect)=c("FPR","TPR")
+ return(vect)
+ }
> threshold=0.5
> S=predict(tree.Response,datatest,type="vector")
> roc.curve(threshold,print=TRUE)
      Predicted
Observed  No  Yes
      No    0  55
      Yes   0 729
FPR TPR
NaN NaN
Warning messages:
1: In (Ps == 1) * (Pass.test == 0) :
  longer object length is not a multiple of shorter object length
2: In (Ps == 1) * (Pass.test == 1) :
  longer object length is not a multiple of shorter object length
> ROC.curve=Vectorize(roc.curve)
> M.ROC1cart=ROC.curve(seq(0,1,by=.01))
There were 50 or more warnings (use warnings() to see the first 50)
> plot(M.ROC1cart[1,],M.ROC1cart[2,],type="l",col="yellow",lwd=2)
Error in plot.window(...) : need finite 'xlim' values
In addition: Warning messages:
1: In min(x) : no non-missing arguments to min; returning Inf
2: In max(x) : no non-missing arguments to max; returning -Inf
3: In min(x) : no non-missing arguments to min; returning Inf
4: In max(x) : no non-missing arguments to max; returning -Inf
```

**Note:** We tried executing the code but we encountered some errors.

## 4.9.16. BAGGING AND RANDOM FOREST

### BAGGING ( MODEL 1:)

```
> library(randomForest)
> set.seed(1)
> bag.cmp=randomForest(y~x523+x251+x389+x494+x222+x360,datatrain,mtry=5,importance=TRUE)
> bag.cmp

Call:
randomForest(formula = y ~ x523 + x251 + x389 + x494 + x222 +      x360, data = datatrain, mtry = 5, importance = TRUE)
  Type of random forest: classification
        Number of trees: 500
No. of variables tried at each split: 5

      OOB estimate of  error rate: 6.77%
Confusion matrix:
  0 1 class.error
0 729 1 0.001369863
1  52 1 0.981132075
> yhat.bag=predict(bag.cmp,datatest)
> yhat.bag=as.numeric(yhat.bag)
> plot(yhat.bag,ytest)
> abline(0,1)
> mean((yhat.bag-ytest)^2)
[1] 0.06377551
> bag.cmp$importance
```

	0	1	MeanDecreaseAccuracy	MeanDecreaseGini
x523	0.08248146	-0.07687750	0.07165921	21.97006
x251	0.06529562	-0.05648732	0.05717559	20.32543
x389	0.05358311	-0.05364453	0.04630104	20.81507
x494	0.06929761	-0.06535887	0.06022883	11.75123
x222	0.06367484	-0.06064852	0.05533121	11.19617
x360	0.04554854	-0.04533312	0.03938110	12.11285

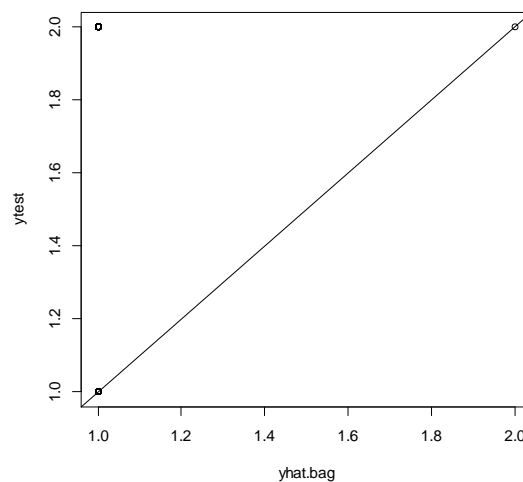


Figure 12: Plot of predicted vs tested for model 1

**Inference: The classification error rate is 6.3%**



## MODEL 2:

```
> library(randomForest)
> set.seed(1)
> bag.cmp=randomForest(y~x251+x389+x360,datatrain,mtry=3,importance=TRUE)
> bag.cmp

Call:
randomForest(formula = y ~ x251 + x389 + x360, data = datatrain,      mtry = 3, importance = TRUE)
      Type of random forest: classification
      Number of trees: 500
No. of variables tried at each split: 3

      OOB estimate of  error rate: 7.28%
Confusion matrix:
      0 1 class.error
0 725 5 0.006849315
1  52 1 0.981132075
> yhat.bag=predict(bag.cmp,datatest)
> yhat.bag=as.numeric(yhat.bag)
> plot(yhat.bag,ytest)
> abline(0,1)
> mean((yhat.bag-ytest)^2)
[1] 0.07015306
> bag.cmp$importance
      0      1 MeanDecreaseAccuracy MeanDecreaseGini
x251 0.041755911 -0.028591910      0.036985545      37.46488
x389 0.038054398 -0.043221111      0.032436542      34.27638
x360 0.003737721  0.001083625      0.003551905      25.67537
```

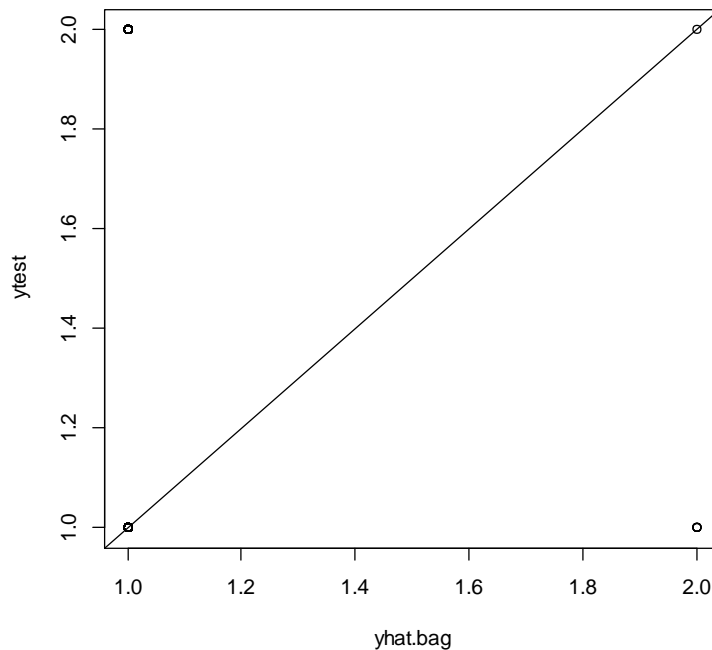


Figure 13: Plot of predicted vs tested for model 2

**Inference: The classification error rate is 7.01%**

### MODEL 3:

```
> library(randomForest)
> set.seed(1)
> bag.cmp=randomForest(y~x251+x389,datatrain,mtry=2,importance=TRUE)
> bag.cmp

Call:
randomForest(formula = y ~ x251 + x389, data = datatrain, mtry = 2, importance = TRUE)
  Type of random forest: classification
    Number of trees: 500
No. of variables tried at each split: 2

  OOB estimate of  error rate: 7.79%
Confusion matrix:
      0 1 class.error
0 722 8   0.0109589
1  53 0   1.0000000
> yhat.bag=predict(bag.cmp,datatest)
> yhat.bag=as.numeric(yhat.bag)
> plot(yhat.bag,ytest)
> abline(0,1)
> mean((yhat.bag-ytest)^2)
[1] 0.07015306
> bag.cmp$importance
      0      1 MeanDecreaseAccuracy MeanDecreaseGini
x251 0.03392005 -0.02715729      0.02975852      50.24353
x389 0.03370880 -0.03316184      0.02927345      48.56057
```

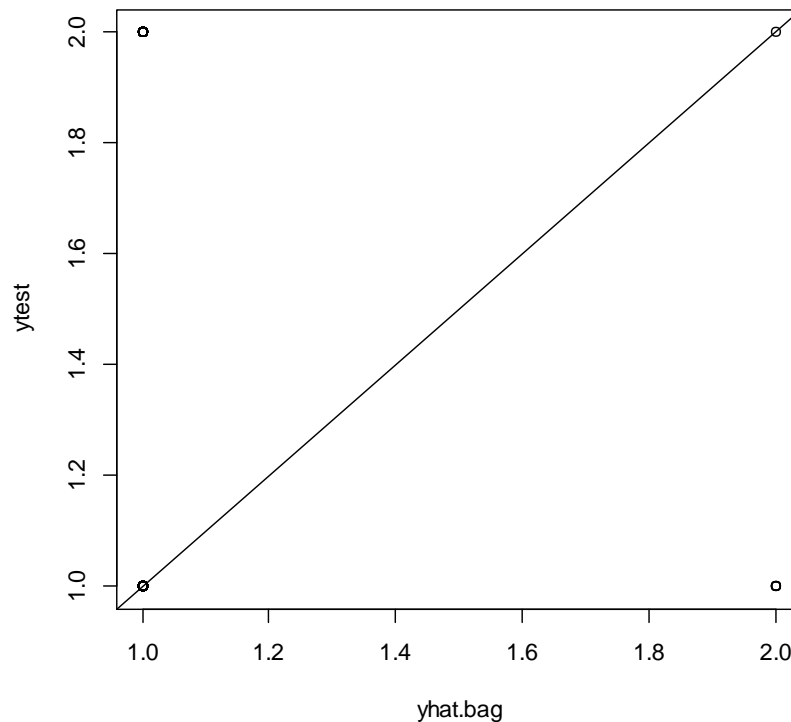


Figure 14: Plot of predicted vs tested for model 3

**Inference: The classification error rate is 7.01%**

### CONCLUSIONS FROM BAGGING AND RANDOM FOREST

MODEL	PREDICTORS	ERROR RATE
<b>MODEL 1</b>	<b>x523,x251,x389,x494,x222,x360</b>	<b>6.3%</b>
<b>MODEL 2</b>	<b>x251,x389,x360</b>	<b>7.01%</b>
<b>MODEL 3</b>	<b>x251,x389</b>	<b>7.01%</b>

Table 6: Error rates of different models for bagging & random forest

**Inference : Model 1 is the Best**

#### 4.9.17. BOOSTING

```
> library(gbm)
> ytest<-as.numeric(ytest)
> set.seed(1)
> boost.data <- gbm(y~x523+x251+x389+x494+x222+x360,data.2, distribution = "bernoulli", n.trees = 5000, interaction.depth = 4)
> summary(boost.data)
      var rel.inf
x523 x523 21.61637
x389 x389 21.30449
x251 x251 16.51787
x222 x222 13.70461
x494 x494 13.57435
x360 x360 13.28231
> yhat.boost = predict(boost.data, newdata = datatest,n.trees = 5000, shrinkage = 0.2, verbose = F)
> mean((yhat.boost-ytest)^2)
[1] 15.20375
|
```

**TEST MEAN SQUARE ERROR IS 15.2**

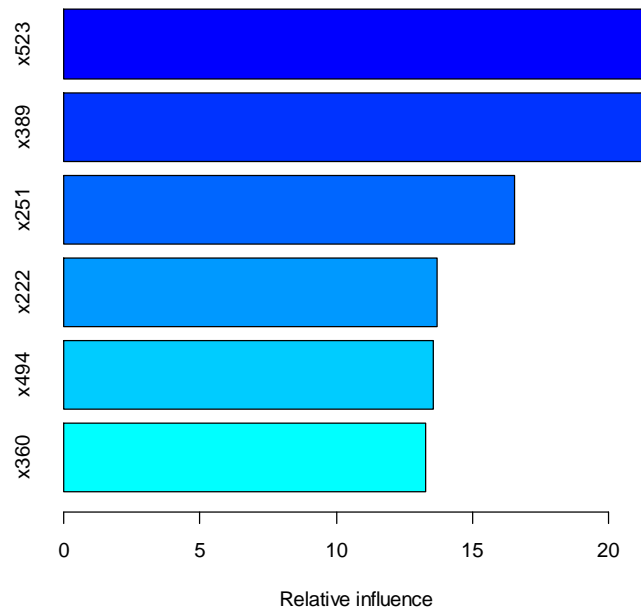


Figure 15: Relative influence of predictors for model 1

## MODEL 2:

```
> library(gbm)
> ytest<-as.numeric(ytest)
> set.seed(1)
> boost.data <- gbm(y~x251+x389+x360,data.2, distribution = "bernoulli", n.trees = 5000, interaction.depth = 4)
> summary(boost.data)
      var rel.inf
x389 x389 35.54315
x251 x251 33.63137
x360 x360 30.82547
> yhat.boost = predict(boost.data, newdata = datatest,n.trees = 5000, shrinkage = 0.2, verbose = F)
> mean((yhat.boost-ytest)^2)
[1] 15.05938
```

**TEST MEAN SQUARE ERROR IS 15.05**

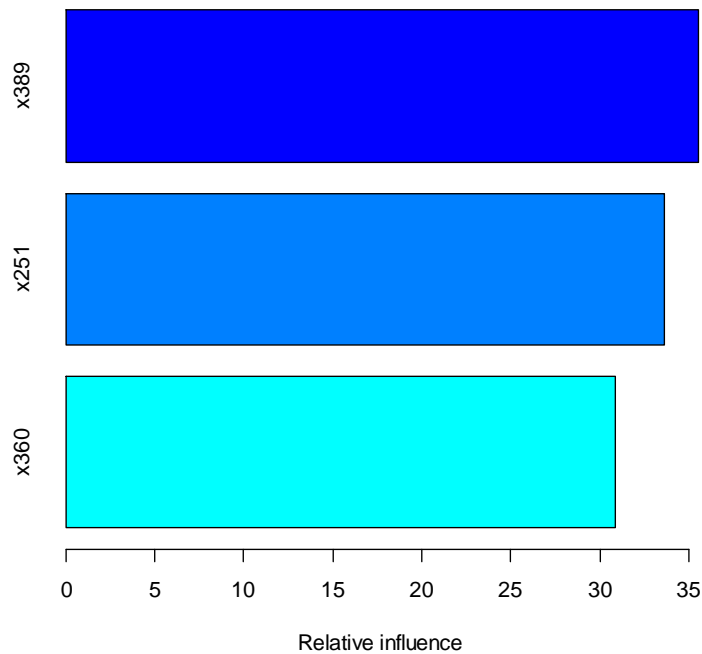


Figure 16: Relative influence of predictors for model 2

### MODEL 3:

```
> library(gbm)
> ytest<-as.numeric(ytest)
> set.seed(1)
> boost.data <- gbm(y~x251+x389,data.2, distribution = "bernoulli", n.trees = 5000, interaction.depth = 4)
> summary(boost.data)
      var rel.inf
x389 x389 52.15182
x251 x251 47.84818
> yhat.boost = predict(boost.data, newdata = datatest,n.trees = 5000, shrinkage = 0.2, verbose = F)
> mean((yhat.boost-ytest)^2)
[1] 14.95638
```

**TEST MEAN SQUARE ERROR IS 14.95**

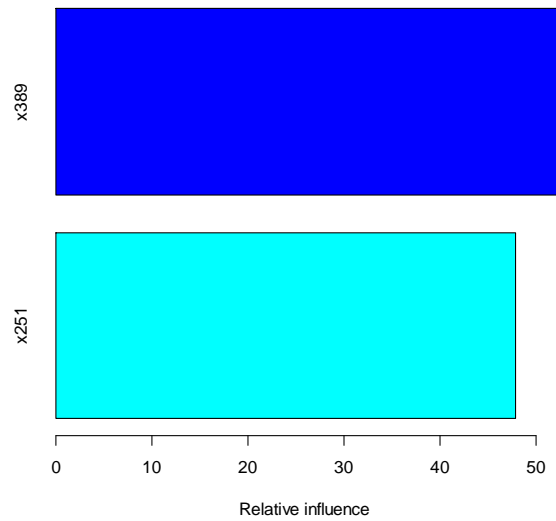


Figure 17: Relative influence of predictors for model 3

### CONCLUSIONS FROM BOOSTING:

MODEL	PREDICTORS	TEST MEAN SQUARE ERROR
MODEL 1	x523,x251,x389,x494,x222,x360	15.2
MODEL 2	x251,x389,x360	15.05
MODEL 3	x251,x389	14.95

Table 7: Error rates of different models for bagging & random forest

**Inference : Model 3 is the Best**

#### 4.9.18. NAÏVE BAYES CLASSIFIER

##### MODEL 1:

```
library(e1071)
set.seed(1)
datatest=data.2[-train,]
data.2 <- data.frame(t(na.omit(t(data))))
data.2$y<-as.factor(data.2$y)
datatest=data.2[-train,]
ytest=data.2[-train,1]
naivebayes.fit <- naiveBayes(y~x523+x251+x389+x494+x222+x360, data=data.2, type = "response")
nbpred <- predict(naivebayes.fit, datatest)
table(nbpred,ytest)
      ytest
nbpred  0   1
      0 726 48
      1   7  3
> mean(nbpred!=ytest)
[1] 0.07015306
```

**Inference: The classification error rate is 7.01%**

##### MODEL 2:

```
library(e1071)
set.seed(1)
datatest=data.2[-train,]
data.2 <- data.frame(t(na.omit(t(data))))
data.2$y<-as.factor(data.2$y)
datatest=data.2[-train,]
ytest=data.2[-train,1]
naivebayes.fit <- naiveBayes(y~x251+x389+x360, data=data.2, type = "response")
nbpred <- predict(naivebayes.fit, datatest)
table(nbpred,ytest)
      ytest
nbpred  0   1
      0 730 49
      1   3  2
> mean(nbpred!=ytest)
[1] 0.06632653
```

**Inference: The classification error rate is 6.66%**

### MODEL 3:

```
> set.seed(1)
> library(e1071)
> set.seed(1)
> datatest=data.2[-train,]
> data.2 <- data.frame(t(na.omit(t(data))))
> data.2$y<-as.factor(data.2$y)
> datatest=data.2[-train,]
> ytest=data.2[-train,1]
> naivebayes.fit <- naiveBayes(y~x251+x389, data=data.2, type = "response")
> nbpred <- predict(naivebayes.fit, datatest)
> table(nbpred,ytest)
      ytest
nbpred  0   1
      0 733  51
      1   0   0
> mean(nbpred!=ytest)
[1] 0.06505102
```

**Inference: The classification error rate is 6.50%**

#### 4.9.19. CROSS VALIDATION for Naïve Bayes Classifier

```
> library('ElemStatLearn')
> library("klaR")
> library("caret")
> model = train(datatrain,ytrain,trControl=trainControl(method='cv',number=10))
> ytest=data.2[-train,1]
> 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
```



## CONCLUSIONS FROM NAÏVE BAYES CLASSIFIER

MODEL	PREDICTORS	ERROR RATE	CV ERROR RATE
MODEL 1	x523,x251,x389,x494,x222,x360	7.01%	6.6%
MODEL 2	x251,x389, x360	6.66%	6.6%
MODEL 3	x251,x389	6.5%	6.6%

Table 8: Error rates of different models for Naïve Bayes Classifier

**Inference : Model 3 is the Best**

### 4.9.20. ROC Curve for Naïve Bayes Classifier

```
library(e1071)
library(ROCR)
pred.naiveBayes <- predict(naivebayes.fit,datatest)
pred <- prediction(pred.naiveBayes,ytest)
perf=performance(pred,"tpr","fpr")
plot(perf,add=TRUE,col="blue",lwd=2)
```

**Note:** We tried executing the code but we encountered some errors. Below is the screen shot.

```
> library(ROCR)
> pred.naiveBayes <- predict(naivebayes.fit,datatest)
> pred <- prediction(pred.naiveBayes,ytest)
Error in prediction(pred.naiveBayes, ytest) :
  Format of predictions is invalid.
> perf=performance(pred,"tpr","fpr")
Error: unexpected input in "perf=performance(pred,""
> plot(perf,add=TRUE,col="blue",lwd=2)
Error: unexpected input in "plot(perf,add=TRUE,col=""
```

## 4.9.21. SUPPORT VECTOR CLASSIFIER

### LINEAR KERNEL

#### MODEL 1:

```
> data.3=data.frame(data.2,Pass)
> train=sample(1:nrow(data.3), (nrow(data.3))/2)
> datatrain=data.3[train,]
> datatest=data.3[-train,]
> Pass.test= datatest$Pass
> library(e1071)
> svmfit=svm(Pass~x523+x251+x389+x494+x222+x360,data=datatrain ,kernel="linear",cost=0.01)
> summary(svmfit)
```

Call:

```
svm(formula = Pass ~ x523 + x251 + x389 + x494 + x222 + x360, data = datatrain,
     kernel = "linear", cost = 0.01)
```

Parameters:

SVM-Type: C-classification

SVM-Kernel: linear

cost: 0.01

gamma: 0.1666667

Number of Support Vectors: 106

( 60 46 )

Number of Classes: 2

Levels:

No Yes

```
> predtest=predict(svmfit,datatest)
```

```
> table(Pass.test,predtest)
```

	predtest	
Pass.test	No	Yes
No	0	58
Yes	0	726

```
> mean(Pass.test!=predtest)
```

```
[1] 0.07397959
```

## MODEL 2:

```
> library(e1071)
> library("e1071")
> svmfit=svm(Pass~x251+x389+x360,data=datatrain ,kernel="linear",cost=0.01)
> summary(svmfit)
```

Call:

```
svm(formula = Pass ~ x251 + x389 + x360, data = datatrain, kernel = "linear",
     cost = 0.01)
```

Parameters:

```
  SVM-Type:  C-classification
SVM-Kernel:  linear
      cost:  0.01
      gamma: 0.3333333
```

Number of Support Vectors: 117

```
( 63 54 )
```

Number of Classes: 2

Levels:

```
No Yes
```

```
> predtest=predict(svmfit,datatest)
```

```
> table(Pass.test,predtest)
```

```
      predtest
Pass.test No Yes
      No    0  50
      Yes   0 734
```

```
> mean(Pass.test!=predtest)
```

```
[1] 0.06377551
```

### MODEL 3:

```
> library(e1071)
> library("e1071")
> svmfit=svm(Pass~x251+x389,data=datatrain ,kernel="linear",cost=0.01)
> summary(svmfit)
```

Call:

```
svm(formula = Pass ~ x251 + x389, data = datatrain, kernel = "linear",
     cost = 0.01)
```

Parameters:

```
  SVM-Type:  C-classification
SVM-Kernel:  linear
      cost:  0.01
      gamma: 0.5
```

Number of Support Vectors: 112

```
( 57 55 )
```

Number of Classes: 2

Levels:

```
No Yes
```

```
> predtest=predict(svmfit,datatest)
```

```
> table(Pass.test,predtest)
```

```
      predtest
Pass.test No Yes
      No    0 49
      Yes   0 735
```

```
> mean(Pass.test!=predtest)
```

```
[1] 0.0625
```

```
> |
```

## **RADIAL KERNEL:**

### **MODEL 1:**

```
> library(e1071)
> library("e1071")
> svmfit=svm(Pass~x523+x251+x389+x494+x222+x360,data=datatrain ,kernel="radial",cost=0.01)
> summary(svmfit)
```

Call:

```
svm(formula = Pass ~ x523 + x251 + x389 + x494 + x222 + x360, data = datatrain,
     kernel = "radial", cost = 0.01)
```

Parameters:

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

Number of Support Vectors: 119

```
( 53 66 )
```

Number of Classes: 2

Levels:

```
No Yes
```

```
> predtest=predict(svmfit,datatest)
```

```
> table(Pass.test,predtest)
```

```
      predtest
Pass.test No Yes
      No    0  51
      Yes   0 733
```

```
> mean(Pass.test!=predtest)
```

```
[1] 0.06505102
```

## MODEL 2:

```
> library("e1071")
> svmfit=svm(Pass~x251+x389+x360,data=datatrain ,kernel="radial",cost=0.01)
> summary(svmfit)

Call:
svm(formula = Pass ~ x251 + x389 + x360, data = datatrain, kernel = "radial",
     cost = 0.01)

Parameters:
  SVM-Type:  C-classification
 SVM-Kernel: radial
       cost: 0.01
      gamma: 0.3333333

Number of Support Vectors: 132

( 73 59 )

Number of Classes: 2

Levels:
No Yes

> predtest=predict(svmfit,datatest)
> table(Pass.test,predtest)
      predtest
Pass.test No Yes
      No   0  45
      Yes  0 739
> mean(Pass.test!=predtest)
[1] 0.05739796
```

### MODEL 3:

```
> library(e1071)
> library("e1071")
> svmfit=svm(Pass~x251+x389,data=datatrain ,kernel="radial",cost=0.01)
> summary(svmfit)
```

Call:

```
svm(formula = Pass ~ x251 + x389, data = datatrain, kernel = "radial",
     cost = 0.01)
```

Parameters:

```
  SVM-Type:  C-classification
SVM-Kernel:  radial
      cost:  0.01
      gamma:  0.5
```

Number of Support Vectors: 105

```
( 54 51 )
```

Number of Classes: 2

Levels:

```
No Yes
```

```
> pretest=predict(svmfit,datatest)
```

```
> table(Pass.test,pretest)
```

```
      pretest
Pass.test No Yes
      No   0  53
      Yes  0 731
```

```
> mean(Pass.test!=pretest)
```

```
[1] 0.06760204
```

## 4.9.22. CROSS VALIDATION for Support Vector Machine

### MODEL 1:

```
> library(ipred)
> errorest(Pass~x523+x251+x389+x494+x222+x360,data=data.3, model=svm, estimator="cv")$error
[1] 0.06636886
```

### MODEL 2:

```
> library(ipred)
> errorest(Pass~x251+x389+x360,data=data.3, model=svm, estimator="cv")$error
[1] 0.06636886
```

### MODEL 3:

```
> library(ipred)
> errorest(Pass~x251+x389,data=data.3, model=svm, estimator="cv")$error
[1] 0.06636886
```

### CONCLUSIONS FROM SUPPORT VECTOR CLASSIFIER:

MODEL	PREDICTORS	LINEAR KERNEL	RADIAL KERNEL	CV ERROR RATE
MODEL 1	x523,x251,x389,x494,x222,x360	7.39%	6.5%	6.63%
MODEL 2	x251,x389, x360	6.37%	5.7%	6.63%
MODEL 3	x251,x389	6.25%	6.7%	6.63%

Table 9: Error rates of different models for support vector classifier

**Inference : Model 2 with Radial Kernel is the Best**



## 5. Results & Conclusions:

Method	Error Rate
Logistic Regression	6.17%
Linear Discriminant Analysis	6.37%
Quadratic Discriminant Analysis	6.5%
K-nearest neighbors	7.4%
CART	7.65%
Bagging & Random Forest	6.3%
Boosting	14.95 (MSE)
Naïve Bayes Classifier	6.5%
<b>Support Vector Classifier</b>	<b>5.7%</b>

### **Conclusions:**

According to the analysis completed till now. We find that Support Vector Classifier is the best method from all the above models due to low error rate. The error rates tend to change by a factor of (+) or (-) 1 % as we keep running the methods multiple times. The research papers have played a very important role in our understanding of the data and completion of this project. Almost all of these classifier methods were covered in some research paper or the other and has helped us in expecting the possible outcomes even before we executed the models.

## References

- [1] 13. M. Espinoza, J. A. K. Suykens, and B. De Moor, "Fixed-size least squares support vector machines: A large scale application in electrical load forecasting," Computational Management Science-Special Issue on Support Vector Machines, 2006, vol. 3, 2: pp. 113-129
- [2] Z. Ge and Z. Song, "Semiconductor manufacturing process monitoring based on adaptive substatistical PCA," IEEE Trans. Semiconductor Manufacturing, vol.23, no.1, pp.99-108, Feb. 2010.
- [3] B.E. Goodlin, D.S. Boning, H.H. Sawin, and B.M. Wise, "Simultaneous fault detection and classification for semiconductor manufacturing tools," Journal of The Electrochemical Society, vol.150, no.12, pp.G778-G784, 2003.
- [4] I. Guyon and A. Elisseeff, "An introduction to variable and feature selection," Journal of Machine Learning Research, vol.3, pp.1157-1182, 2003.
- [5] Q.P. He and J. Wang, "Fault detection using the k-nearest neighbor rule for semiconductor manufacturing processes," IEEE Trans. Semiconductor Manufacturing, vol.20, no.4, pp.345-354, Nov. 2007.
- [6] A.M. Ison, W. Li, and C.J. Spanos, "Fault diagnosis of plasma etch equipment," in Proc. IEEE Int. Symp. Semiconductor Manufacturing, San Francisco, 68 October 1997, pp.-49-B-52.
- [7] A.M. Ison and C. Spanos, "Robust fault detection and fault classification of semiconductor manufacturing equipment," in Proc. Int. Symp. Semiconductor Manufacturing, Tokyo, 2-4 October 1996.
- [8] G.S. May and C.J. Spanos, Fundamentals of Semiconductor Manufacturing and Process Control. John Wiley & Sons, 2006.
- [9] M. McCann, Y. Li, L. Maguire, and A. Johnston, "Causality challenge: benchmarking relevant signal components for effective monitoring and process control," in Proc. JMLR Workshop, Canada, 12 December 2008, pp.277-288.
- [10] Semi-Conductor Manufacturing. (2010, November 25). Available: <http://www.causality.inf.ethz.ch/repository.php>
- [11] G. Spitzlsperger, C. Schmidt, G. Ernst, H. Strasser, and M. Speil, "Fault detection for a via etch process using adaptive multivariate methods," IEEE Trans. Semiconductor Manufacturing, vol.18, no.4, pp.528-533, Nov. 2005
- [12] S.J. Qin, G. Cherry, R. Good, J. Wang, and C.A. Harrison, "Semiconductor manufacturing process control and monitoring: a fab-wide framework," Journal of Process Control, vol.16, pp.179-191, 2006.
- [13] E. Tafazzoli and M. Saif, "Application of combined support vector machines in process fault diagnosis," in Proc. American Control Conf., St. Louis, USA, 10-12 June 2009, pp.3429-3433.
- [14] G. Verdier and A. Ferreira, "Fault detection with an adaptive distance for the k-nearest neighbor rule," in Proc. Int. Conf. Computer & Industrial Engineering

, Troyes, France, 6-9 July 2009, pp.1273-1278.

[15] G. Verdier and A. Ferreira. (2010). Adaptive Mahalanobis distance and k-nearest neighbor rule for fault detection in semiconductor manufacturing. IEEE Trans. Semiconductor Manufacturing. Available: doi:10.1109/TSM.2010.2065531

[16] WEKA. (2010, October 23). Available: <http://www.cs.waikato.ac.nz/ml/weka/>

[17] N. Syed, H. Liu, and K. Sung. "Incremental learning with support vector machines," In Proceedings of the Workshop on Support Vector Machines at the International Joint Conference on Artificial Intelligence (IJCAI-99), Stockholm, Sweden, 1999

[18] S. Ruping, "Incremental learning with support vector machines," Proceedings of the IEEE International Conference on Data Mining, 2001, pp.641-642

[19] H. Alhammady and K. Ramamohanarao, The application of emerging patterns for improving the quality of rare-class classification, in Proc. PKDD, 2004, pp. 207-211.

[20] H. Alhammady and K. Ramamohanarao, Using emerging patterns and decision trees in rare-class classification, in Proc. ICDM, 2004, pp. 315-318.

[21] J. Burez and D. Van den Poel, Handling class imbalance in customer churn prediction, Expert Systems with Applications, vol. 36, 2009, pp. 4626-4636.

[22] G. Costa, M. Guarascio, G. Mamco, R. Ortale, and E. Ritacco, Rule learning with probabilistic smoothing, in Proc. DaWaK, 2009, pp. 428-440.

[23] Z. Ge and Z. Song, Semiconductor manufacturing process monitoring based on adaptive substatistical PC, IEEE Trans. Semiconductor Manufacturing, vol. 23, no. 1, 2010, pp. 99-108

[24] B. E. Goodlin, D. S. Boning, H. H. Sawin, and B. M. Wise, Simultaneous fault detection and classification for semiconductor manufacturing tools, J. Electrochemical Society, vol. 150, no. 12, 2003, pp. G778-G784.

[25] S. Han, B. Yuan, and W. Liu, Rare class mining: progress and prospect, in Proc. Chinese Conference on Pattern Recognition, 2009, pp. 1-5.

[26] Q. P. He and J. Wang, Fault detection using the k-nearest neighbor rule for semiconductor manufacturing processes, IEEE Trans. Semiconductor Manufacturing, vol. 20, no. 4, 2007, pp. 345-354.

[27] A. M. Ison, W. Li, and C. J. Spanos, Fault diagnosis of plasma etch equipment, in Proc. IEEE Int. Symp. Semiconductor Manufacturing, 1997, pp. B49-B52.

[28] G. S. May and C. J. Spanos, Fundamentals of Semiconductor Manufacturing and Process Control, John Wiley & Sons, 2006.

[29] C. Seiffert, T. M. Khoshgoftaar, and J. Van Hulse, Improving software-quality predictions with data sampling and boosting, IEEE Trans. Systems, Man, Cybernetics –Part A: Systems and Humans, vol. 39, no. 6, 2009, pp. 1283-1294.

[30] G. Spitzlsperger, C. Schmidt, G. Ernst, H. Strasser, and M. Speil, Fault detection for a via etch process using adaptive multivariate methods, IEEE Trans. Semiconductor Manufacturing, vol. 18, no. 4, 2005, pp. 528-533.

- [31]J. Stefanowski and S. Wilk, Selective pre-processing of imbalanced data for improving classification performance, in Proc. DaWaK, 2008, pp. 283-292.
- [32]E. Tafazzoli and M. Saif, Application of combined support vector machines in process fault diagnosis, in Proc. American Control Conf., 2009, pp. 3429-3433.
- [33]J. Van Hulse and T. Khoshgoftaar, Knowledge discovery from imbalanced and noisy data, Data & Knowledge Engineering, vol. 68, 2009, pp. 1513-1542.
- [34]J. Van Hulse, T. M. Khoshgoftaar, and A. Napolitano, An empirical comparison of repetitive undersampling techniques, in Proc. IEEE Int. Conf. Information Reuse & Integration, 2009, pp. 29-34.
- [35]G. Verdier and A. Ferreira, Adaptive Mahalanobis distance and k-nearest neighbour rule for fault detection in semiconductor manufacturing, IEEE Trans. Semiconductor Manufacturing, vol. 24, no. 1, 2011, pp. 59-68.
- [36]G. M. Weiss, Mining with rarity: a unifying framework, SIGKDD Explorations Newsletter, vol. 6, issue 1, 2004, pp. 7-19.
- [37]G. M. Weiss, Mining with rare cases, in O. Maimon, L. Rokach (eds.), Data Mining and Knowledge Discovery Handbook, 2nded., Springer Science + Business Media, 2010, pp. 747-757.
- [38] Huang, G.-B. , Zhou, H. , Ding, X. , Zhang, R. , "Extreme Learning Machine for Regression and Multiclass Classification ," IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics, 2011, Issue 99: 1-17
- [39]G.-B. Huang, P. Saratchandran, and N. Sundararajan, "A generalized growing and pruning RBF (GGAP-RBF) neural network for function approximation," IEEE Trans. Neural Netw., 2005, vol. 16, 1: pp. 57-67
- [40] Nan-Ying Liang, Guang-Bin Huang, P. Saratchandran, and N.Sundararajan: A Fast and Accurate On-line Sequential Learning Algorithm for Feedforward Networks. IEEE Transactions on Neural Networks, vol. 17, 6: pp. 1411-1423. (2006)