ILLINOIS

Electronic Nose:
A Classification Analysis

Luis Steven Lin

STAT 428 – Statistical Computing
Professor Wenxuan Zhong
05-08-13

ABSTRACT

This study compares various supervised machine learning methods using electronic nose data. More specifically, six popular classification methods are implemented: linear discriminant analysis (LDA), logistic regression (LOG), support vector machines (SVM), classification and regression tree (CART), random forest (RF), and boosting (BOOST). The advantages and disadvantages of each method are discussed, and then the various methods are compared in terms of misclassification errors on a testing dataset after using a training set to train the machine-learning algorithms. Finally, the computational difficulty of high-dimensionality is addressed by using principal component analysis (PCA).

Contents

List of Tables

List of Figures

1. INTRODUCTION

1.1 Background

The electronic nose (colorimetric sensor array) data contains 147 rows and 108 columns. The rows refer to 21 groups of toxic chemical compounds (TIC), with 7 replicates for each group. The columns refer to the color changes in the array. The objective is to apply a classification analysis so that given the color changes, one can classify the TIC.

Figure 1 shows a picture of a colorimetric sensor array. The difference color maps for a few TICs are shown and are a result of color changes captured by nanoporous pigments.

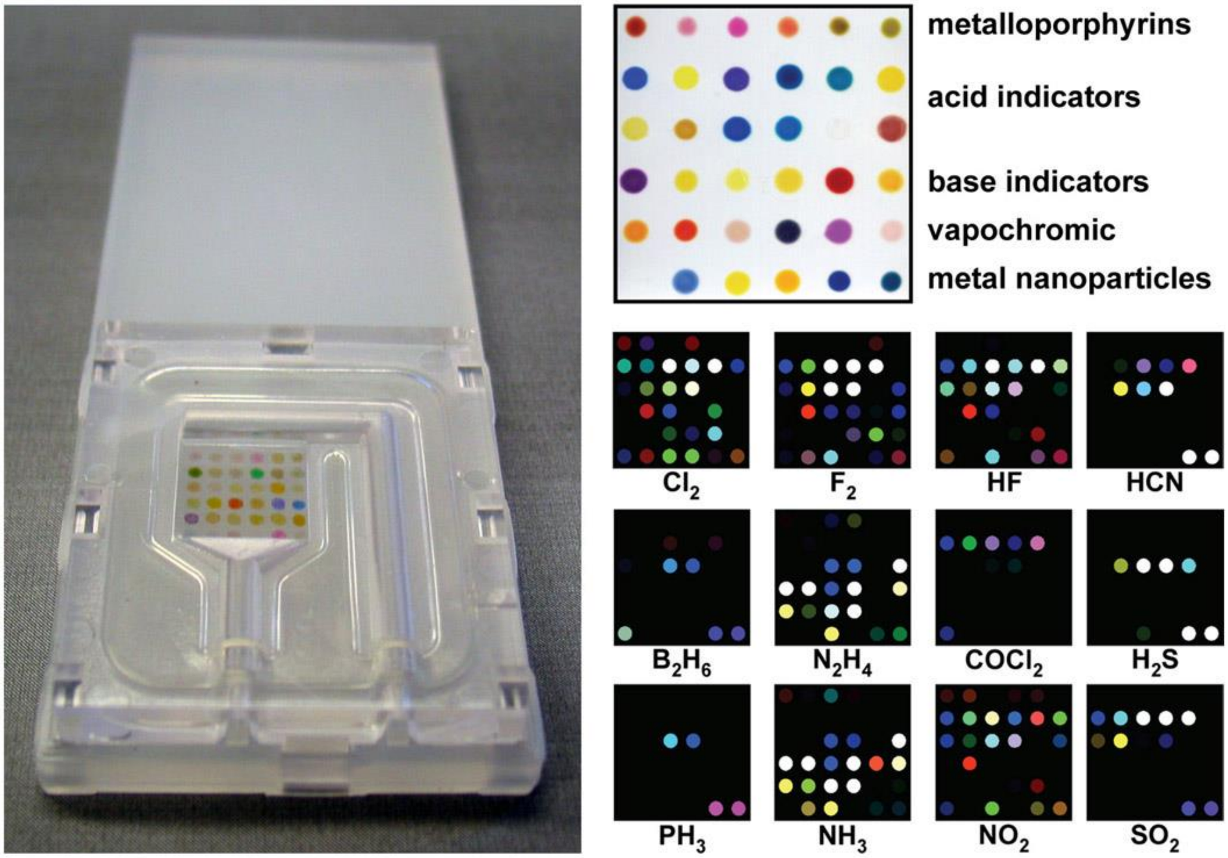


Figure 1. Colorimetric sensor array (Source: http://spie.org/x39363.xml)

2. METHODOLOGY

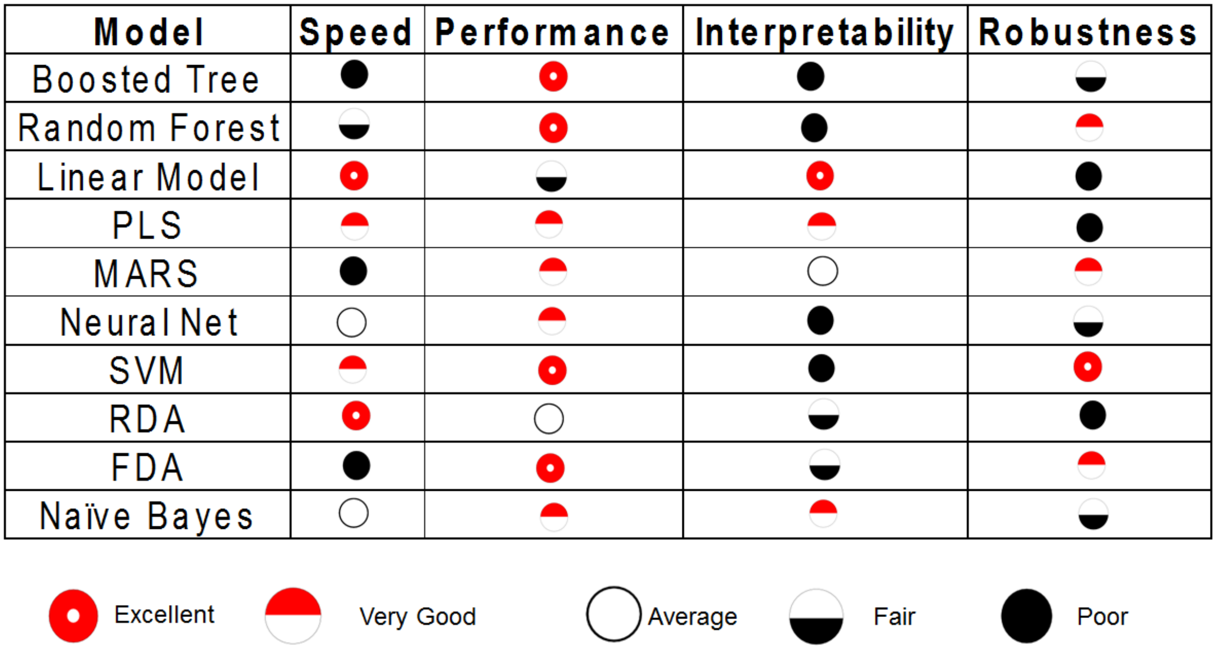
2.1 Classification Methods

The classification methods used in the study were linear discriminant analysis (LDA), logistic regression (LOG), support vector machines (SVM), classification and regression tree (CART), random forest (RF), and boosting (BOOST).

LDA is a parametric method which maximizes the full log-likelihood often by least squares estimates and has normal assumption disadvantage, while logistic regression is distribution-free and maximizes the conditional likelihood by a Newton-Raphson algorithm. For multi-classification, a logistic regression is trained for each group to predict the probability of belonging to that group. Then, the predicted group is the one that has the highest probability of belonging to that group. One advantage of LDA is that it tends to show consistently higher performance because the data support only a simple boundary. A major disadvantage is that LDA suffers from small sample size problems. One advantage of logistic regression is that it does not require many assumptions, and one drawback is the lack of flexibility and binary nature. Note that the normal assumption is not required by LDA, but if satisfied, LDA is proven to give good estimates and outperform logistic regression. Research has shown that the efficiency (variance of the estimate) in the misclassification error rate reduces 30% compared to LDA if the data is normally distributed.

The other classification methods are also widely use and each has its advantages and disadvantages. SVM is distribution-free and its main advantage is that it minimizes the risk of over fitting by choosing the maximal margin hyperplane. However, the speed of learning tends to average and is a binary classifier. In addition, there are currently no asymptotic results available under this method. In contrasts, CART has the advantage of having excellent speed of classification and supports multi-classification. However, accuracy does not tend to be as good as other methods and over-fitting might be an issue. On the other hand, random forest is non-parametric and has the advantages of having no over-fitting problems and not being sensitive to outliers in general. However, one drawback is that it cannot predict beyond the range in the training data. Finally, boosting is a popular community voting (ensemble method) method that uses the principles of resampling. One advantage is that it has wide applicability and high accuracy, but tends to have a very slow learning speed.

A top-level comparison of other classification methods is summarized below in Table 1.



Model	Speed	Performance	Interpretability	Robustness
Boosted Tree	●	◉	●	◒
Random Forest	◒	◉	●	◓
Linear Model	◉	◒	◉	●
PLS	◓	◓	◓	●
MARS	●	◓	○	◓
Neural Net	○	◓	●	◒
SVM	◓	◉	●	◉
RDA	◉	○	◒	●
FDA	●	◉	◒	◓
Naïve Bayes	○	◓	◓	◒

◉ Excellent ◓ Very Good ○ Average ◒ Fair ● Poor

Table 1. Summary classification methods (Source: Max Kuhn)

2.2 High Dimensionality

When there are more variables than the sample size, high-dimensionality becomes a problem. The reason is that in order to apply LDA, the within-class covariance matrix needs to be non-singular (i.e. full rank) in order to be inverted to find the LDA directions. However, when there is high-dimensionality, there are more variables than the sample size, so the matrix is singular (e.g. not full rank) and inversion is not possible. The approach covered in this study is to apply PCA in the pre-processing stage before LDA is conducted. By finding and selecting linear combination of the predictors, the dimensions of the data can be reduced and LDA can be applied

2.3 Procedure

The classification and testing procedure is shown in Figure 2. The data was divided into a training set and testing set by randomly removing a repeat from each of the 21 groups to use for the testing set. Thus, the training set and testing set contained 126 and 21 observations respectively. Note that for the training set, the sample size (126) is more than the number of variables (108), so no high-dimensionality problems in the methods are expected. The training set was used to train the machine-learning algorithms, and then the classification labels were predicted for the testing dataset. The procedure was repeated 100 times for each classification method.

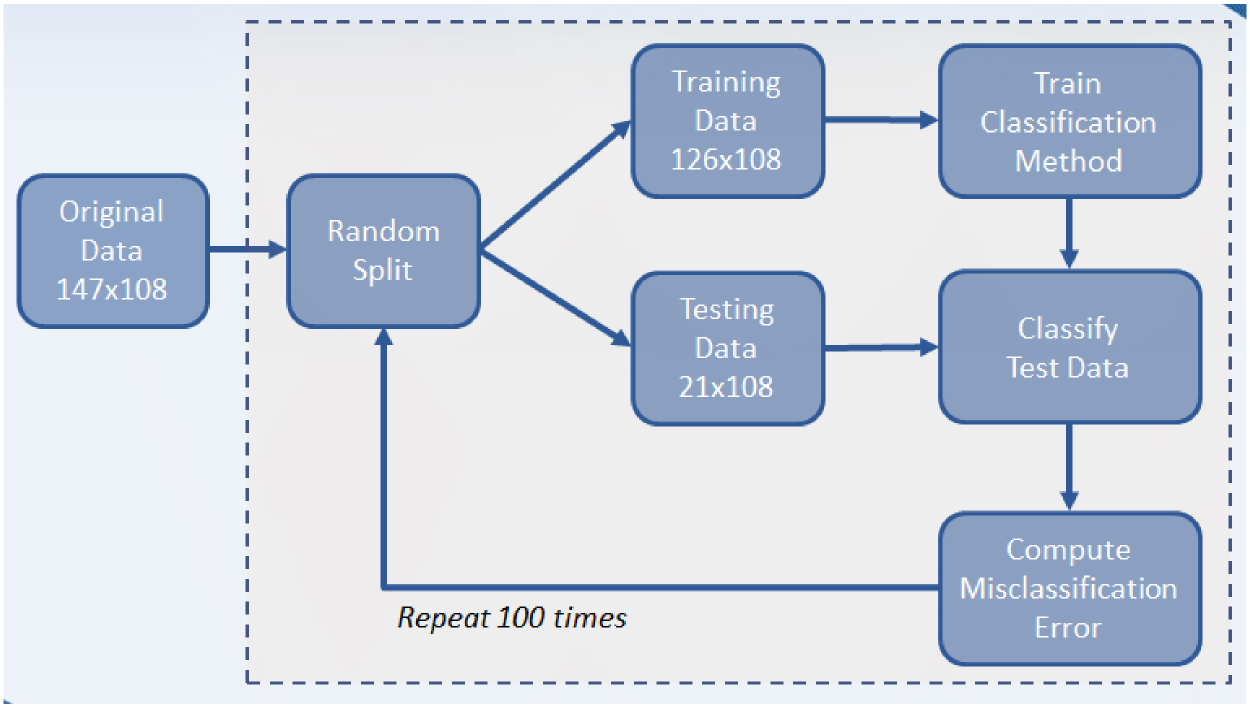


Figure 2. Procedure

3. DISCUSSION

3.1 Classification Methods Results

The results of the classification analysis summarized in Figure 3. As the results show, the average misclassification error was lowest for random forest (0%) and was highest for CART (14.24%). Regarding computation time, boosting has the longest time (average of 20.140 seconds per iteration) while LDA, SVM and CART had the lowest with times under 0.1 second. Comparing LDA and logistic, it can be seen that LDA outperforms logistic in terms of average misclassification error (4.81% vs. 12.05%) and average computation time (e.g. 0.09 s vs. 1.51 s).

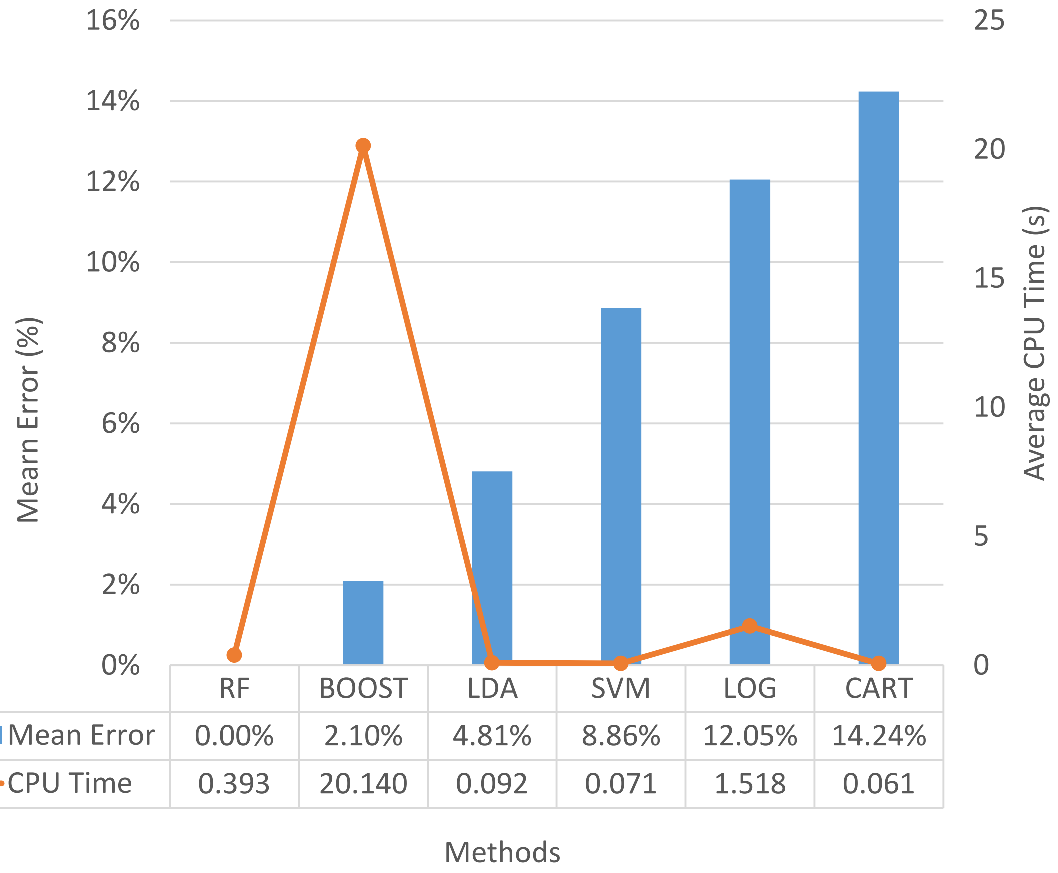


Figure 3. Mean misclassification error and CPU time by method

Table 2 summarizes the total number of misclassifications of the testing dataset by TIC and method. These results are illustrated graphically in a heat map in Figure 4. The heat map shows that certain TICs tend to be misclassified more often for certain methods. For example, SVM misclassified Arsine and Diborane in the testing datasets more often than any other method, but did not misclassify the other TICs (except Phosphine) at all. This analysis can be helpful in determining if any relationship among TICs that tend to be misclassified more often exist. From the heat map, it can also be observed that certain TICs (e.g. HCN, Cl2, MA) tend not to be misclassify very often.

Table 2. Misclassification errors by TIC and method

	LDA	LOG	SVM	CART	RF	BOOST
HCHO	4	3	0	34	0	0
Arsine	17	6	76	33	0	2
Control	9	2	0	38	0	0
Diborane	5	2	81	14	0	5
Phosphine	25	16	29	0	0	17
Phosgene	1	2	0	10	0	2
H2S	0	1	0	6	0	0
DM	0	15	0	7	0	2
MA	1	11	0	0	0	0
TA	2	19	0	7	0	0
Ammonia	5	15	0	0	0	0
HCl	3	72	0	11	0	0
Cl2	1	7	0	1	0	0
F2	1	5	0	16	0	0
NO2	0	7	0	17	0	3
MH	2	0	0	30	0	0
SO2	0	4	0	14	0	0
HCN	0	8	0	1	0	0
HF	9	28	0	12	0	8
HNO3	12	16	0	44	0	5
Hydrazine	4	14	0	4	0	0

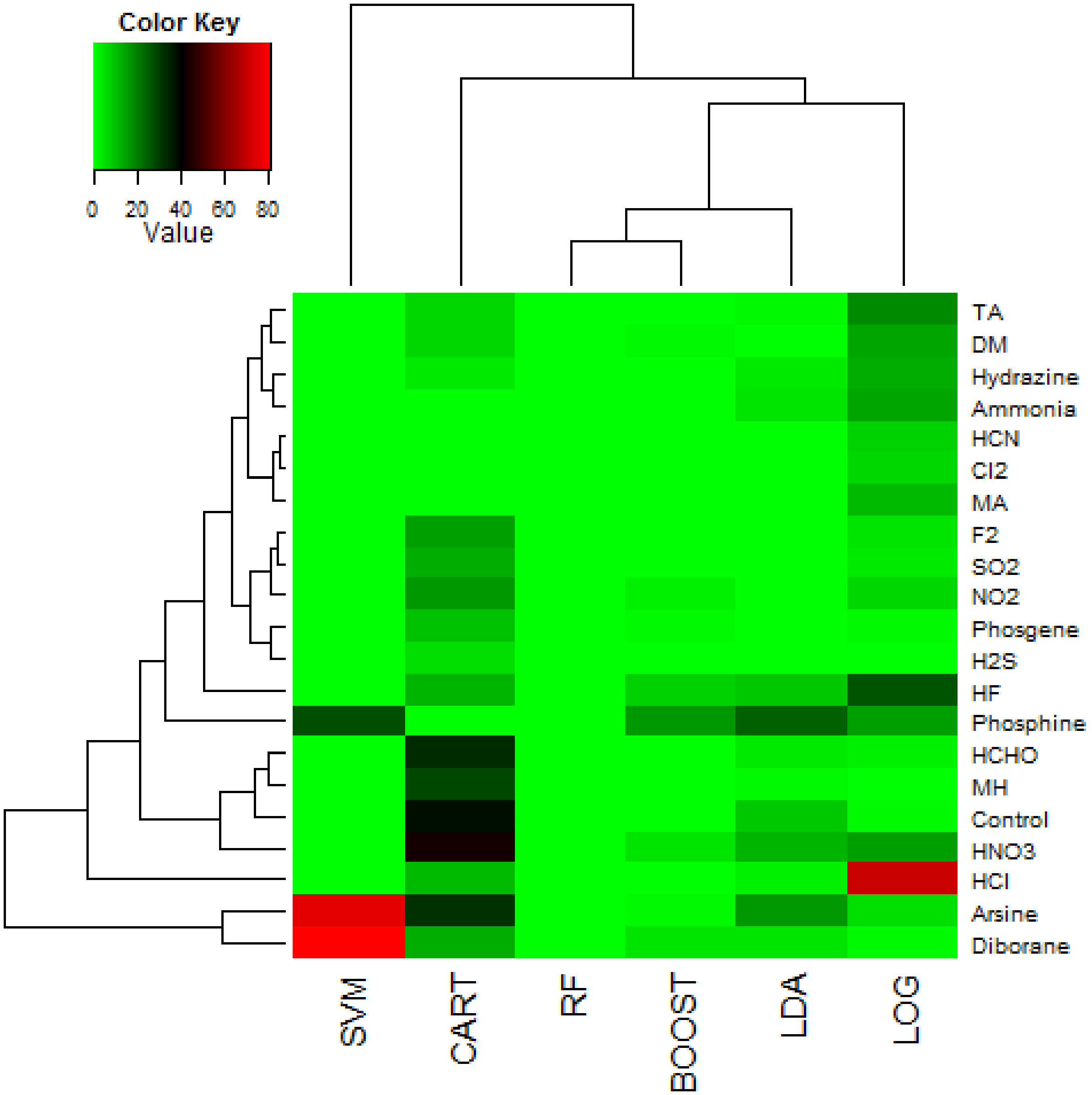


Figure 4. Heat map of misclassification errors by TIC and method

3.2 High Dimensionality

In order to test the high dimensionality issue, two repeats from each of the 21 groups were removed from the data, leaving a total of 105 observations. The data was further divided into a training set and testing set by randomly removing an additional repeat from each of the 21 groups to use for the testing set. Thus, the training set and testing set contained 84 and 21 observations respectively. Note that for the training set, the sample size (84) is more than the number of variables (108), so a high-dimensionality problem is expected. This was confirmed by running LDA.

In order to address this issue, PCA was used to reduce the dimension of the data (with the two repeats removed) by finding a linear combination of the predictors (i.e. principal components) that account for most of the variation. The PCA results are shown in the scree plot below (Figure 5) and indicate that 14 PCs explain 99% of the variation in the data.

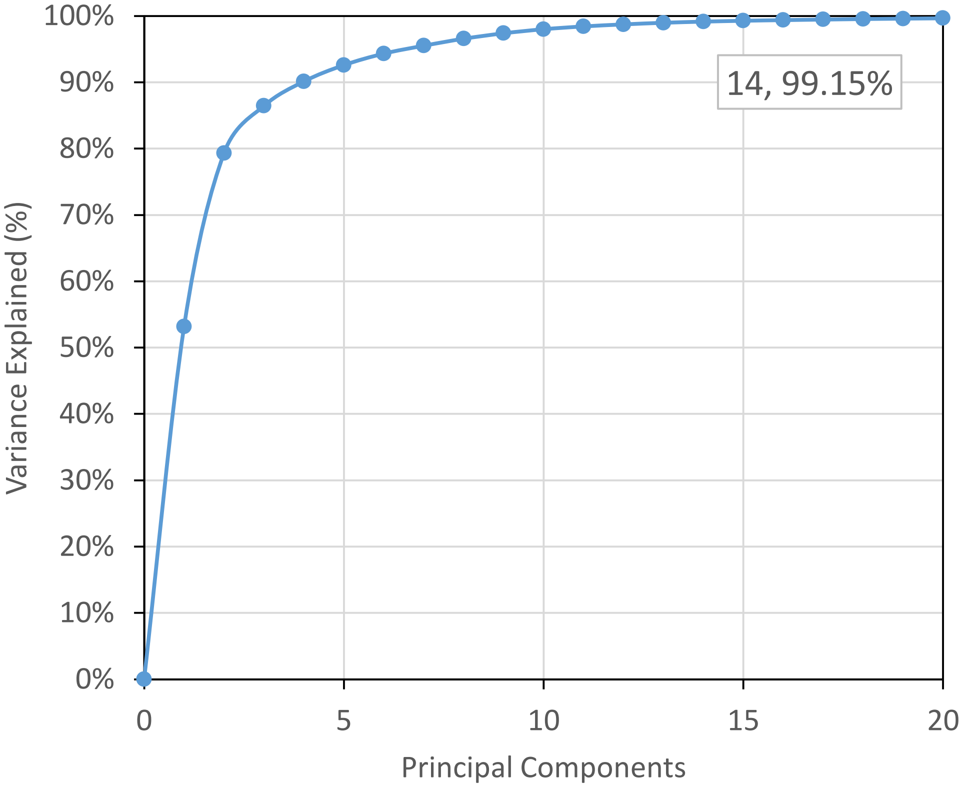


Figure 5. Scree plot for the first 20 PCs

Using the PCs as predictors instead of the original variables, the training set was used to train the machine-learning algorithms, and then the classification labels were predicted for the testing dataset. The procedure was repeated 100 times for LDA and for different number of PCs. The results are summarized in Table 3 and Figure 6 and indicate that, as expected, increasing the number of PCs used decreases the average misclassification error of LDA. The fact that the average misclassification error rate with 3 PCs (i.e. 4.38%) is comparable to the average misclassification error rate with all the variables in the original data (i.e. 4.81%) suggests that PCA is an effective way of building a reduced-dimensional space for classification, which can be noted by the fact that no collinearity warnings are given by R.

Table 3. Mean error of LDA by PC

PC	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Mean Error (%)	31.48	13.19	4.38	3.81	2.33	1.67	2.10	1.10	0.43	0.29	0.67	0.05	0.05	0.05

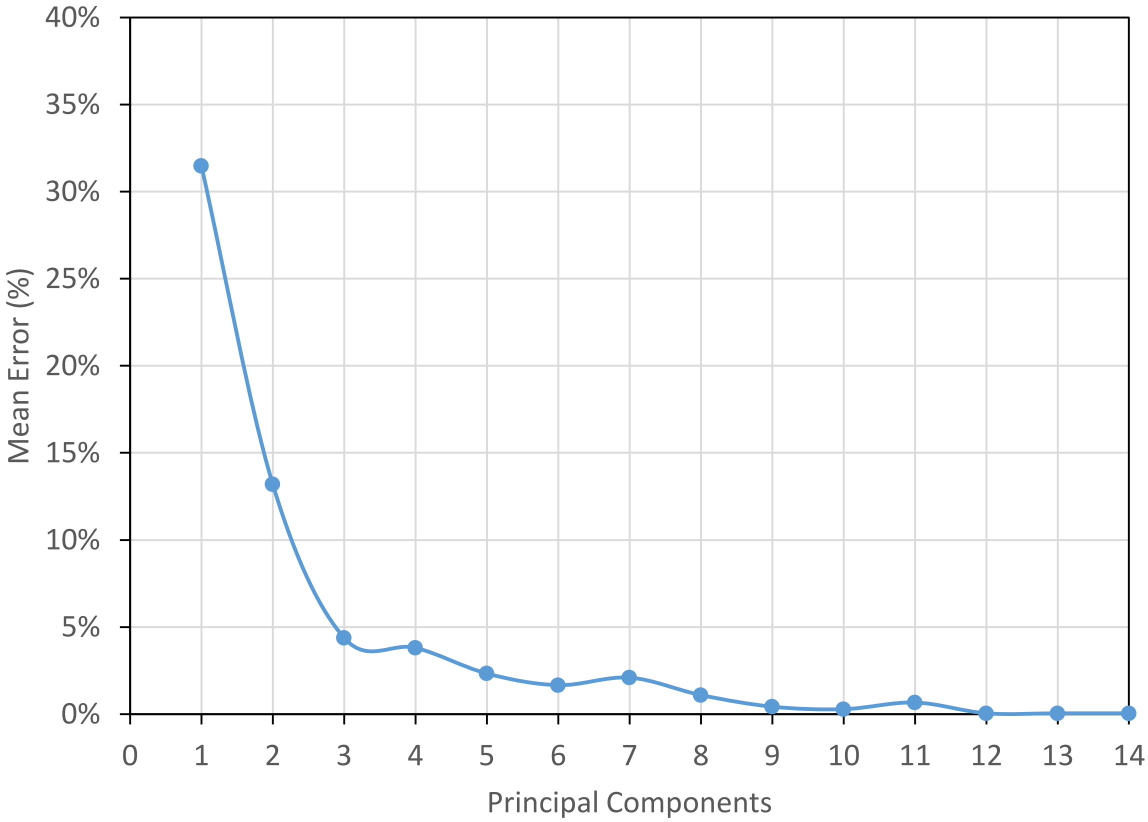


Figure 6 Mean error of LDA by PC

The misclassification errors of LDA by TIC and PC are shown in Table 4 and graphically as a heat map in Figure 7. Overall, the misclassification decrease as more PCs are sued. However, the results also suggest that there are a cluster of TICs that tend to be misclassified more often than others. For example, the upper and lower TICs in the heat map have large misclassification numbers when few PCs are used. On the other hand, the TICs in the center have low misclassification numbers or none at all regardless of the number of PCs used.

Table 4. Misclassification errors of LDA by TIC and PC

	PC1	PC2	PC3	PC4	PC5	PC6	PC7	PC8	PC9	PC10	PC11	PC12	PC13	PC14
HCHO	37	48	13	8	6	12	22	23	9	6	14	1	1	1
Arsine	18	0	0	0	0	0	0	0	0	0	0	0	0	0
Control	37	27	0	5	9	1	1	0	0	0	0	0	0	0
Diborane	65	33	0	0	0	1	1	0	0	0	0	0	0	0
Phosphine	44	33	11	11	15	14	17	0	0	0	0	0	0	0
Phosgene	7	0	0	0	0	0	0	0	0	0	0	0	0	0
H2S	77	10	0	0	0	0	0	0	0	0	0	0	0	0
DM	0	0	0	0	0	0	0	0	0	0	0	0	0	0
MA	28	25	25	21	2	0	0	0	0	0	0	0	0	0
TA	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Ammonia	32	21	22	13	5	7	3	0	0	0	0	0	0	0
HCl	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Cl2	0	0	0	0	0	0	0	0	0	0	0	0	0	0
F2	43	0	0	0	0	0	0	0	0	0	0	0	0	0
NO2	65	17	0	0	0	0	0	0	0	0	0	0	0	0
MH	0	0	0	0	0	0	0	0	0	0	0	0	0	0
SO2	34	31	0	0	0	0	0	0	0	0	0	0	0	0
HCN	0	0	3	0	0	0	0	0	0	0	0	0	0	0
HF	82	1	4	0	0	0	0	0	0	0	0	0	0	0
HNO3	92	23	6	22	12	0	0	0	0	0	0	0	0	0
Hydrazine	0	8	8	0	0	0	0	0	0	0	0	0	0	0

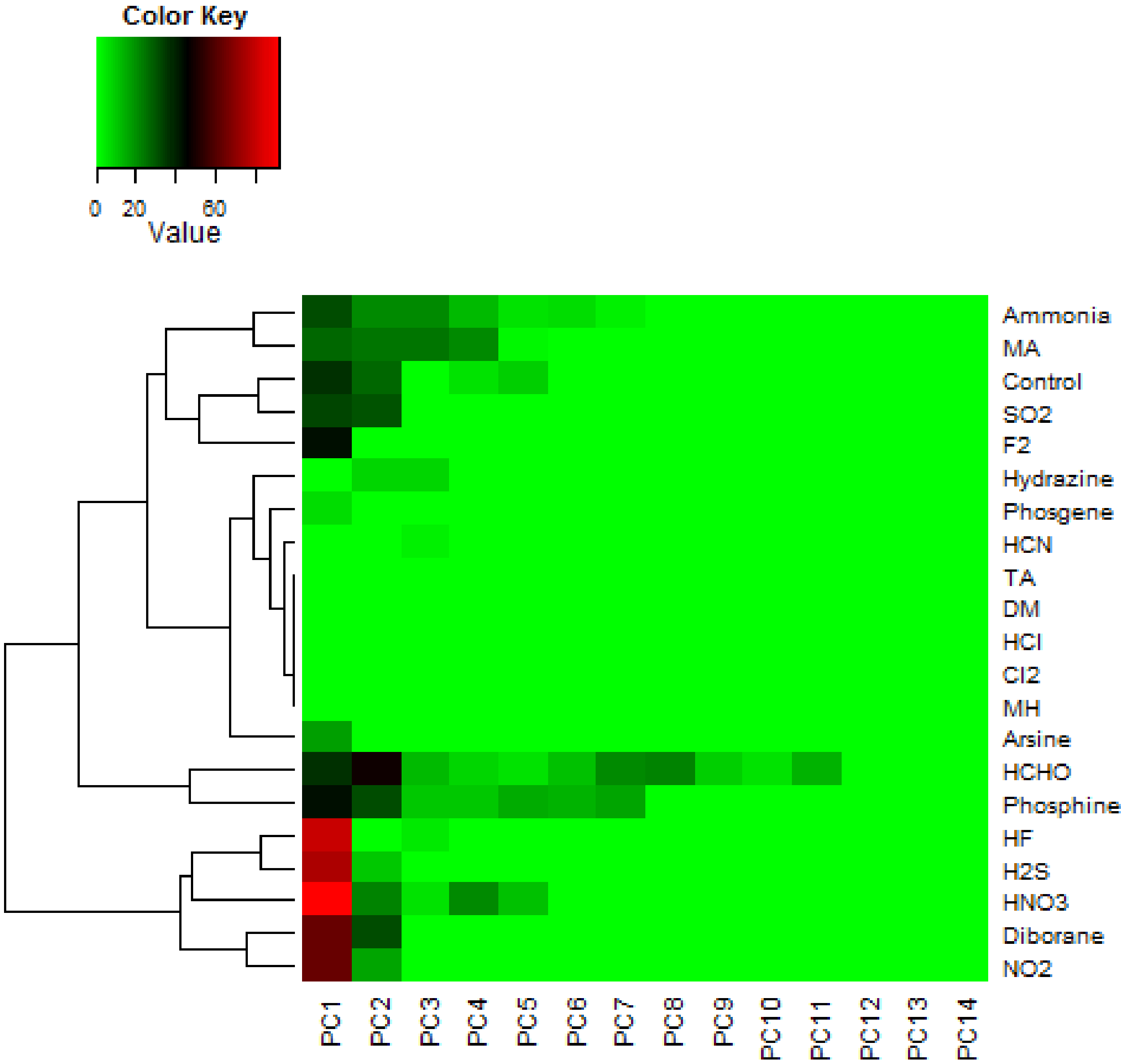


Figure 7. Heat map of misclassification errors of LDA by TIC and PC

4. CONCLUSIONS

4.1 Summary

This study compared various supervised machine learning methods using electronic nose data. The advantages and disadvantages of each method are discussed, and then the various methods are compared in terms of misclassification errors on a testing dataset after using a training set to train the machine-learning algorithms. The results indicate that random forest (0%) and boosting (2.10%) performed well in terms of misclassification error, while logistic regression and CART had the highest error rates (over 10%). However, boosting took significantly more time than the rest of the methods.

The computational difficulty of high-dimensionality was also addressed by using principal component analysis (PCA) and the results indicate that PCA is an effective pre-processing technique to find a low-dimensional space for LDA. An analysis of the misclassification by TIC and method, and by TIC and PC for LDA were also done and revealed interesting insights for future investigation.

REFERENCES

SVM

http://cran.r-project.org/web/packages/e1071/e1071.pdf

CART

http://cran.r-project.org/web/packages/tree/tree.pdf

Random Forest

http://cran.r-project.org/web/packages/randomForest/randomForest.pdf

Boosting

http://cran.r-project.org/web/packages/adabag/adabag.pdf

General

http://geossdev.med.virginia.edu/research/teaching/2007/L7%20SupervisedLearning.ppt

http://www.whrc.org/education/indonesia/pdf/DecisionTrees_RandomForest_v2.pdf

Comparison

http://www.ncbi.nlm.nih.gov/pmc/articles/PMC2253491/

http://zoo.cs.yale.edu/classes/cs445/slides/Pfizer_Yale_Version.ppt

APPENDIX

A. R-Code

```
####################
# Class: STAT 428  #
# Date: 04/15/13   #
# Project          #
# Luis Steven Lin  #
####################

########################################################################
# Terminology

# loadings or rotations = principal component
# 1st PC = direction of maximum spread
# scores or 'x' (data%*%rotation)= projecttion of data points
# along the PCs (rotated data) = predict(my.pca)
# LDA
# loadings/coeff of the discriminant functions = 'scaling'
# discriminant functions = predict(my.lda)$x

########################################################################
# Data Info

# rows = checmial analytes
# 147 chemical analytes can be separated into 21 groups
# with each group has 7 analytes
# colunms = 108 colors


########################################################################
# Load data

getwd()
data=read.csv('DB_220_IDLH_2mins.txt', header=F, row.names=1, sep='\t')

# Examine data
dim(data)
rownames(data)
colnames(data)
data[1:14,1:5]

########################################################################
# Data Preparation

# get group labels
y = rownames(data)

# replace "-" with "_" to have the same pattern
y = gsub("-","_",y,fixed=TRUE)

# Keep the group name only (string before "_")
y = sapply(strsplit(y,"_",fixed = TRUE),"[[",1)

# String as levels
y = as.factor(y)

# Bind y to data
data = cbind(y,data)
```

```
# Function to randomly pick 1 analyte from each group

randomIndices = function(seed=F){
        if(seed==T){set.seed(1)}
        indicesTest = rep(0,21)
        for(i in 1:21){
                indicesTest[i] = sample(7,1) + (i-1)*7
        }
        return(indicesTest)
}

# Randomly pick 1 analyte from each group
# indicesTest=randomIndices()

# Testing and training data set
        dataTest = data[indicesTest,]
        dataTrain = data[-indicesTest,]
        yTest = dataTest[,'y']
        yTrain = dataTrain[,'y']

########################################################################
# Linear Discriminant Analysis (LDA)

library(MASS)

# Start the clock!
ptm <- proc.time()

errorRate.LDA = rep(0,100)
errorID.LDA = matrix(0,21)

        # get group labels
        yTestNames = rownames(dataTest)

        # replace "-" with "_" to have the same pattern
        yTestNames  = gsub("-","_",yTestNames,fixed=TRUE)

        # Keep the group name only (string before "_")
        yTestNames  = sapply(strsplit(yTestNames,"_",fixed = TRUE),"[[",1)


rownames(errorID.LDA)=yTestNames
colnames(errorID.LDA)="Errors"

# Run LDA 100 times and compute the error

for(i in 1:100){

        set.seed(i)
        # Randomly pick 1 analyte from each group
        indicesTest=randomIndices()

        # Testing and training data set
        dataTest = data[indicesTest,]
        dataTrain = data[-indicesTest,]
        yTest = dataTest[,'y']
        yTrain = dataTrain[,'y']

        # Construct LDA model with training set
        my.lda=lda(y~.,data=dataTrain)

        # Use LDA to predict group of testing data
        my.pred=predict(my.lda,dataTest)

        # Compute misclassification error rate and save
        miss = my.pred$class!=yTest
        errorRate.LDA[i] = sum(miss)/length(yTest)

        # Find which groups were misclassified in the test data
        errorID.LDA = errorID.LDA + as.numeric(miss)
}
```

```
# Average misclassfication rate (0.04809524)
meanError.LDA  = mean(errorRate.LDA)
meanError.LDA

# Double check
sum(errorID.LDA)/(21*100)

# Misclassified groups in test data
errorID.LDA

# Stop the clock (9.16)
proc.time() - ptm

########################################################################
# Logistic Regression

# Start the clock!
ptm <- proc.time()

errorRate.LOG = rep(0,100)
errorID.LOG = matrix(0,21)

        # get group labels
        yTestNames = rownames(dataTest)

        # replace "-" with "_" to have the same pattern
        yTestNames  = gsub("-","_",yTestNames,fixed=TRUE)

        # Keep the group name only (string before "_")
        yTestNames  = sapply(strsplit(yTestNames,"_",fixed = TRUE),"[[",1)


rownames(errorID.LOG)=yTestNames
colnames(errorID.LOG)="Errors"
for(i in 1:100){

        set.seed(i)

        # Randomly pick 1 analyte from each group
        indicesTest=randomIndices()

        # Testing and training data set
        dataTest = data[indicesTest,]
        dataTrain = data[-indicesTest,]
        yTest = dataTest[,'y']
        yTrain = dataTrain[,'y']

# Run 21 different logistic regressions for the 21 groups

prob=matrix(0,21)
for(j in 1:21){
        y = rep(0,6*21)
        y[((j-1)*6+1):((j-1)*6+6)]=1

        # Substitute group labels with 1 for current group, 0 for the rest
        dataTrain[,1]=y

        # Run logistic regression for the current group with training data
        my.glm = glm(y~.,family=binomial,data=dataTrain)

        # Predict probability that the test data of the current group
        # falls in the current group
        my.pred = predict(my.glm,newdata = dataTest, type="response")
        prob=cbind(prob,my.pred)
        }

# Find classification of test data: for each group in test data (row),
# choose the group with highest probability (column)
# This probability represents the predicted probability of belonging
# to that group vs. other
prob = prob[,-1]
```

```
rownames(prob) = yTestNames
colnames(prob) = yTestNames
yClass=yTest
for(j in 1:21){
        index=which(max(prob[j,])==prob[j,])
        yClass[j] = yTest[index]
        }
miss = yClass!=yTest

# Compute misclassification error rate and save
errorRate.LOG[i] = sum(miss)/length(yTest)


# Find which groups were misclassified in the test data
errorID.LOG = errorID.LOG + as.numeric(miss)
}

# Average misclassfication rate (0.1219048)
meanError.LOG = mean(errorRate.LOG)
meanError.LOG

# Double Check
sum(errorID.LOG)/(21*100)

# Misclassified groups in test data
errorID.LOG

# Stop the clock (151.79)
proc.time() - ptm

########################################################################
# Support Vector Machine (SVM)

# Install package e1071
install.packages("e1071")
library(e1071)

# Start the clock!
ptm <- proc.time()

errorRate.SVM = rep(0,100)
errorID.SVM = matrix(0,21)

        # get group labels
        yTestNames = rownames(dataTest)

        # replace "-" with "_" to have the same pattern
        yTestNames  = gsub("-","_",yTestNames,fixed=TRUE)

        # Keep the group name only (string before "_")
        yTestNames  = sapply(strsplit(yTestNames,"_",fixed = TRUE),"[[",1)


rownames(errorID.SVM)=yTestNames
colnames(errorID.SVM)="Errors"

# Run SVM 100 times and compute the error

for(i in 1:100){

        set.seed(i)

        # Randomly pick 1 analyte from each group
        indicesTest=randomIndices()

        # Testing and training data set
        dataTest = data[indicesTest,]
        dataTrain = data[-indicesTest,]
        yTest = dataTest[,'y']
        yTrain = dataTrain[,'y']
        # Construct SVM model with training set
        my.svm = svm(y~.,data=dataTrain)

        # Use SVM to predict group of testing data
        my.pred = predict(my.svm,dataTest)

        # Compute misclassification error rate and save
        miss = my.pred!=yTest
        errorRate.SVM[i] = sum(miss)/length(yTest)

        # Find which groups were misclassified in the test data
        errorID.SVM = errorID.SVM + as.numeric(miss)
}

# Average misclassfication rate (0.08857143)
meanError.SVM = mean(errorRate.SVM)
meanError.SVM

# Double Check
sum(errorID.SVM)/(21*100)

# Misclassified groups in test data
errorID.SVM

# Stop the clock (7.11)
proc.time() - ptm

########################################################################
# Classification and Regression Tree (CART)

# Install package tree
install.packages("tree")
library(tree)

# Start the clock!
ptm <- proc.time()

errorRate.CART = rep(0,100)
errorID.CART = matrix(0,21)

        # get group labels
        yTestNames = rownames(dataTest)

        # replace "-" with "_" to have the same pattern
        yTestNames  = gsub("-","_",yTestNames,fixed=TRUE)

        # Keep the group name only (string before "_")
        yTestNames  = sapply(strsplit(yTestNames,"_",fixed = TRUE),"[[",1)


rownames(errorID.CART)=yTestNames
colnames(errorID.CART)="Errors"

# Run CART 100 times and compute the error

for(i in 1:100){

        set.seed(i)

        # Randomly pick 1 analyte from each group
        indicesTest=randomIndices()

        # Testing and training data set
        dataTest = data[indicesTest,]
        dataTrain = data[-indicesTest,]
        yTest = dataTest[,'y']
        yTrain = dataTrain[,'y']

        # Construct CART model with training set
        my.tree=tree(y~.,data=dataTrain)
        # Use CART to predict group of testing data
        my.pred=predict(my.tree,dataTest,type='class')

        # Compute misclassification error rate and save
        miss = my.pred!=yTest
        errorRate.CART[i] = sum(miss)/length(yTest)

        # Find which groups were misclassified in the test data
        errorID.CART = errorID.CART + as.numeric(miss)
}

# Average misclassfication rate (0.142381)
meanError.CART = mean(errorRate.CART)
meanError.CART

# Double Check
sum(errorID.CART)/(21*100)

# Misclassified groups in test data
errorID.CART

# Stop the clock (6.06)
proc.time() - ptm

########################################################################
# Random Foreset

# Install package random forest
install.packages("randomForest")
library(randomForest)

# Start the clock!
ptm <- proc.time()

errorRate.RF = rep(0,100)
errorID.RF = matrix(0,21)

        # get group labels
        yTestNames = rownames(dataTest)

        # replace "-" with "_" to have the same pattern
        yTestNames  = gsub("-","_",yTestNames,fixed=TRUE)

        # Keep the group name only (string before "_")
        yTestNames  = sapply(strsplit(yTestNames,"_",fixed = TRUE),"[[",1)


rownames(errorID.RF)=yTestNames
colnames(errorID.RF)="Errors"

# Run RF 100 times and compute the error

for(i in 1:100){

        set.seed(i)

        # Randomly pick 1 analyte from each group
        indicesTest=randomIndices()

        # Testing and training data set
        dataTest = data[indicesTest,]
        dataTrain = data[-indicesTest,]
        yTest = dataTest[,'y']
        yTrain = dataTrain[,'y']

        # Construct RF model with training set
        my.RF = randomForest(y~.,data=dataTrain)

        # Use RF to predict group of testing data
        my.pred=predict(my.RF,dataTest,type='class')
        # Compute misclassification error rate and save
        miss = my.pred!=yTest
        errorRate.RF[i] = sum(miss)/length(yTest)

        # Find which groups were misclassified in the test data
        errorID.RF = errorID.RF + as.numeric(miss)
}

# Average misclassfication rate (0)
meanError.RF = mean(errorRate.RF)
meanError.RF

# Double Check
sum(errorID.RF)/(21*100)

# Misclassified groups in test data
errorID.RF

# Stop the clock (39.28)
proc.time() - ptm

########################################################################
# Boosting

# Install adabag package
install.packages("adabag")
library(adabag)


# Start the clock!
ptm <- proc.time()

errorRate.BOOST  = rep(0,100)
errorRate.BOOST2 = rep(0,100)
errorID.BOOST = matrix(0,21)

        # get group labels
        yTestNames = rownames(dataTest)

        # replace "-" with "_" to have the same pattern
        yTestNames  = gsub("-","_",yTestNames,fixed=TRUE)

        # Keep the group name only (string before "_")
        yTestNames  = sapply(strsplit(yTestNames,"_",fixed = TRUE),"[[",1)


rownames(errorID.BOOST)=yTestNames
colnames(errorID.BOOST)="Errors"

# Run BOOST 100 times and compute the error

for(i in 1:100){

        set.seed(i)

        # Randomly pick 1 analyte from each group
        indicesTest=randomIndices()

        # Testing and training data set
        dataTest = data[indicesTest,]
        dataTrain = data[-indicesTest,]
        yTest = dataTest[,'y']
        yTrain = dataTrain[,'y']

        # Construct BOOST model with training set
        my.BOOST = boosting(y~.,data=dataTrain)

        # Use BOOST to predict group of testing data
        my.pred = predict(my.BOOST,newdata=dataTest)

        # Compute misclassification error rate and save
        miss = my.pred$class!=yTest
        errorRate.BOOST[i] = sum(miss)/length(yTest)
        errorRate.BOOST2[i] = my.pred$error

        # Find which groups were misclassified in the test data
        errorID.BOOST = errorID.BOOST + as.numeric(miss)
}

# Average misclassfication rate (0.02095238)
meanError.BOOST = mean(errorRate.BOOST)
meanError.BOOST
mean(errorRate.BOOST2)

# Double Check
sum(errorID.BOOST)/(21*100)

# Misclassified groups in test data
errorID.BOOST

# Stop the clock (elapsed = 2013.95 sec)
proc.time() - ptm


########################################################################
# Summary

errorIDs=cbind(errorID.LDA, errorID.LOG, errorID.SVM,
                errorID.CART, errorID.RF, errorID.BOOST)
colnames(errorIDs) = c("LDA","LOG","SVM","CART","RF","BOOST")

meanErrors=cbind(meanError.LDA, meanError.LOG, meanError.SVM,
                meanError.CART, meanError.RF, meanError.BOOST)
colnames(meanErrors) = c("LDA","LOG","SVM","CART","RF","BOOST")

times = cbind(9.16,151.79,7.11,6.06,39.28,2013.95)
colnames(times) = c("LDA","LOG","SVM","CART","RF","BOOST")

colnames(meanErrors) = c("LDA","LOG","SVM","CART","RF","BOOST")

summary=rbind(meanErrors,times)
rownames(summary)=c("meanError","CPUtime")

summary
errorIDs

# Export
write.csv(summary, "k:/summary.csv")
write.csv(errorIDs, "k:/errorIDs.csv")

# Display heat map
install.packages("gplots")
library("gplots")
heatmap.2(errorIDs,key=TRUE,trace="none",density.info="none",
               col=greenred(75) )

# Save heat map
bmp(filename="k:/heatMapCLASS.bmp")
heatmap.2(errorIDs,key=TRUE,trace="none",density.info="none",
               col=greenred(75) )
dev.off()
```

```
########################################################################
# Take out two repeats from each chemical class (one for testting)

# Function to randomly pick 1 analyte to remove from each group
# and 1 analyte for test data from each group
indices = function(seed=F){
        if(seed==T){set.seed(1)}
        indicesRemove1 = rep(0,21)
        indicesRemove2 = rep(0,21)
        indicesTest = rep(0,21)
        for(i in 1:21){
                indicesRemove1[i] = sample(7,1) + (i-1)*7
                indicesRemove2[i] = sample(6,1) + (i-1)*6
                indicesTest[i] = sample(5,1) + (i-1)*5
        }
        return(list("remove1"=indicesRemove1,"remove2"=indicesRemove2,
                      "test"=indicesTest))
}

########################################################################
# Reapeat Linear Discriminant Analysis (LDA) with removed data

library(MASS)

# Start the clock!
ptm <- proc.time()

errorRate.LDA2 = rep(0,100)
errorID.LDA2 = matrix(0,21)

        # get group labels
        yTestNames = rownames(dataTest)

        # replace "-" with "_" to have the same pattern
        yTestNames  = gsub("-","_",yTestNames,fixed=TRUE)

        # Keep the group name only (string before "_")
        yTestNames  = sapply(strsplit(yTestNames,"_",fixed = TRUE),"[[",1)


rownames(errorID.LDA2)=yTestNames
colnames(errorID.LDA2)="Errors"

# Run LDA 100 times and compute the error

for(i in 1:100){

        set.seed(i)

        # Randomly pick 1 analyte from each group to remove
        # and 1 for test data
        index=indices()

        # Remove analyte from data
        data2 = data[-index$remove1,]
        data2 = data2[-index$remove2,]

        # Testing and training data set
        dataTest = data2[index$test,]
        dataTrain = data2[-index$test,]
        yTest = dataTest[,'y']
        yTrain = dataTrain[,'y']

        # Construct LDA model with training set
        my.lda=lda(y~.,data=dataTrain)

        # Use LDA to predict group of testing data
        my.pred=predict(my.lda,dataTest)

        # Compute misclassification error rate and save
        miss = my.pred$class!=yTest
        errorRate.LDA2[i] = sum(miss)/length(yTest)

        # Find which groups were misclassified in the test data
        errorID.LDA2 = errorID.LDA2 + as.numeric(miss)
}

# Average misclassfication rate (0.04809524)
meanError.LDA2  = mean(errorRate.LDA2)
meanError.LDA2

# Double check
sum(errorID.LDA2)/(21*100)

# Misclassified groups in test data
errorID.LDA2

# Stop the clock (9.16)
proc.time() - ptm


########################################################################
# Principal Component Analysis to reduce dimensions

        # Randomly pick 1 analyte from each group to remove
        # and 1 for test data
        index=indices()

        # Remove analyte from data
        data2 = data[-index$remove1,]
        data2 = data2[-index$remove2,]

        # Use built-in R function for PCA
        my.pca = prcomp(data2[,-1])

        # Get the cumulative propotions of PC
        cumProp = summary(my.pca)$imp["Cumulative Proportion",]
        cumProp

        # Output the cumulative scree plot of first, 25 PC's
        plot(1:25,cumProp[1:25],type="o", pch=16,
        xlab="Principal Components",
        ylab="Proportion of variance explained",
        main="Cumulative Screeplot of the first 25 PCs")

        # Save plot
        bmp(filename="k:/screePlot.bmp")
        plot(1:25,cumProp[1:25],type="o", pch=16,
        xlab="Principal Components",
        ylab="Proportion of variance explained",
        main="Cumulative Screeplot of the first 25 PCs")
        dev.off()

        # Get rotated data (scores) of the first 14 PC s
        scores = my.pca$x[,1:14]


        # Note that my.pca$x, predict(my.pca, data) and predict(data)
        # are all equivalent.

        # Create reduced data
        dataR = data2[,1:2]
        #need to keep V2 because if not y is converted
        #to numeric and changes col name when bind with scores
        dataR = cbind(dataR,scores)
        dataR = dataR[,-2]
```

```
########################################################################
# Reapeat Linear Discriminant Analysis (LDA) with reduced data

library(MASS)

# Start the clock!
ptm <- proc.time()

errorRate.LDAR = rep(0,100)
errorID.LDAR = matrix(0,21)

        # get group labels
        yTestNames = rownames(dataTest)

        # replace "-" with "_" to have the same pattern
        yTestNames  = gsub("-","_",yTestNames,fixed=TRUE)

        # Keep the group name only (string before "_")
        yTestNames  = sapply(strsplit(yTestNames,"_",fixed = TRUE),"[[",1)


rownames(errorID.LDAR)=yTestNames
colnames(errorID.LDAR)="Errors"

# Run LDA 100 times and compute the error

for(i in 1:100){

        set.seed(i)

        # Randomly pick 1 analyte from each group to remove
        # and 1 for test data
        index=indices()

        # Remove analyte from data
        data2 = data[-index$remove1,]
        data2 = data2[-index$remove2,]

        # Use built-in R function for PCA
        my.pca = prcomp(data2[,-1])

        # Get rotated data (scores) of the first 14 PC s
        scores = my.pca$x[,1:14]

        # Create reduced data
        dataR = data2[,1:2]
        #need to keep V2 because if not y is converted
        #to numeric and changes col name when bind with scores
        dataR = cbind(dataR,scores)
        dataR = dataR[,-2]

        # Testing and training data set
        dataTest = dataR[index$test,]
        dataTrain = dataR[-index$test,]
        yTest = dataTest[,'y']
        yTrain = dataTrain[,'y']

        # Construct LDA model with training set
        my.lda=lda(y~.,data=dataTrain)

        # Use LDA to predict group of testing data
        my.pred=predict(my.lda,dataTest)

        # Compute misclassification error rate and save
        miss = my.pred$class!=yTest
        errorRate.LDAR[i] = sum(miss)/length(yTest)

        # Find which groups were misclassified in the test data
        errorID.LDAR = errorID.LDAR + as.numeric(miss)
}
```

```
# Average misclassfication rate (0.04809524)
meanError.LDAR  = mean(errorRate.LDAR)
meanError.LDAR

# Double check
sum(errorID.LDAR)/(21*100)

# Misclassified groups in test data
errorID.LDAR

# Stop the clock (9.16)
proc.time() - ptm


########################################################################
# Reapeat Linear Discriminant Analysis (LDA) with reduced data
# using different number of PCs

library(MASS)

# Start the clock!
ptm <- proc.time()

meanError.LDAR = rep(0,14)
errorIDs.LDAR = matrix(0,21)

# Run LDA using 1 to 14 PCs and compute the error
for(n in 1:14){

errorRate.LDAR = rep(0,100)
errorID.LDAR = matrix(0,21)

        # get group labels
        yTestNames = rownames(dataTest)

        # replace "-" with "_" to have the same pattern
        yTestNames  = gsub("-","_",yTestNames,fixed=TRUE)

        # Keep the group name only (string before "_")
        yTestNames  = sapply(strsplit(yTestNames,"_",fixed = TRUE),"[[",1)


rownames(errorID.LDAR)=yTestNames
colnames(errorID.LDAR)="Errors"

# Run LDA 100 times and compute the error

for(i in 1:100){

        set.seed(i)

        # Randomly pick 1 analyte from each group to remove
        # and 1 for test data
        index=indices()

        # Remove analyte from data
        data2 = data[-index$remove1,]
        data2 = data2[-index$remove2,]

        # Use built-in R function for PCA
        my.pca = prcomp(data2[,-1])

        # Get rotated data (scores) of the first n PC s
        scores = my.pca$x[,1:n]

        # Create reduced data
        dataR = data2[,1:2]
        #need to keep V2 because if not y is converted
        #to numeric and changes col name when bind with scores
        dataR = cbind(dataR,scores)
        dataR = dataR[,-2]
        # Testing and training data set
        dataTest = dataR[index$test,]
        dataTrain = dataR[-index$test,]
        yTest = dataTest[,'y']
        yTrain = dataTrain[,'y']

        # Construct LDA model with training set
        my.lda=lda(y~.,data=dataTrain)

        # Use LDA to predict group of testing data
        my.pred=predict(my.lda,dataTest)

        # Compute misclassification error rate and save
        miss = my.pred$class!=yTest
        errorRate.LDAR[i] = sum(miss)/length(yTest)

        # Find which groups were misclassified in the test data
        errorID.LDAR = errorID.LDAR + as.numeric(miss)
}

# Average misclassfication rate (0.04809524)
meanError.LDAR[n]  = mean(errorRate.LDAR)

# Misclassified groups in test data
errorIDs.LDAR = cbind(errorIDs.LDAR,errorID.LDAR)

}

# Average misclassfication rate
meanError.LDAR=as.matrix(meanError.LDAR)
rownames(meanError.LDAR)= paste(rep("PC",14),1:14,sep="")
colnames(meanError.LDAR)= "meanError"
meanError.LDAR

# Double check for last PC run
sum(errorID.LDAR)/(21*100)

# Misclassified groups in test data
errorIDs.LDAR = errorIDs.LDAR[,-1]
colnames(errorIDs.LDAR)= paste(rep("PC",14),1:14,sep="")
errorIDs.LDAR

# Stop the clock (9.16)
proc.time() - ptm

########################################################################
# Export

cumProp
meanError.LDAR
errorIDs.LDAR

# Export
write.csv(cumProp, "k:/cumProp.csv")
write.csv(meanError.LDAR, "k:/meanErrorLDAR.csv")
write.csv(errorIDs.LDAR, "k:/errorIDsLDAR.csv")

# Display heat map
install.packages("gplots")
library("gplots")
heatmap.2(errorIDs.LDAR,key=TRUE,trace="none",density.info="none",
               col=greenred(75),Colv=F) # Colv=F don't cluster columns

# Save heat map
bmp(filename="k:/heatMapPC.bmp")
heatmap.2(errorIDs.LDAR,key=TRUE,trace="none",density.info="none",
               col=greenred(75),Colv=F) # Colv=F don't cluster columns
dev.off()
```

```
#########################################################################
# Manual Code for LDA (from lab2)

#get same result by diy without using the lda fcn in R
gmean=by(x,y,mean)

#make list as matrix
gmean=matrix(unlist(gmean),ncol=2)
gmean

#compare to R
names(my.lda)
my.lda$means
gmean==t(my.lda$means)

#compute sigma b
gvar=var(t(gmean))

#compute sigma w, by taking avg of the four estimates,
#by law of large number to get a smaller variance
varw=by(x,y,var)
varw1=(varw[[1]]+varw[[2]])/2

#M is asymmtric but it should symmetric
#compute in another way

temp=svd(varw1)
names(temp)
vareta=temp$u%*%diag(1/sqrt(temp$d))%*%t(temp$v)
M=vareta%*%gvar%*%vareta
ldasvd=svd(M)

# eigenvector
ldasvd$u

# coefficient
my.lda$scaling
LDAscale=vareta%*%ldasvd$u

ldaproj=as.matrix(x)%*%LDAscale[,1:2]


#########################################################################
#########################################################################
#multinom
install.packages("VGAM")
library(VGAM)
glm(y~.,family=multinomial,data=dataTrain)
my.MLOG = multinom(y~.,data=dataTrain)
my.pred = predict(my.MLOG,newdata=dataTest,type="prob")
length(my.pred)
```