The Report Committee for Puneet Bansal

Certifies that this is the approved version of the report:

Analysis and Classification of Drift Susceptible Chemosensory Responses

APPROVED BY

SUPERVISING COMMITTEE:

Supervisor: ________________________________

Joydeep Ghosh

____

Christine Julien

Analysis and Classification of Drift Susceptible Chemosensory Responses

by

Puneet Bansal, B.E.

Report

Presented to the Faculty of the Graduate School

of the University of Texas at Austin

in Partial Fulfillment

of the Requirements

for the Degree of

Master of Science in Engineering

The University of Texas at Austin

December 2014

Acknowledgements

I would like to thank my supervisor, Joydeep Ghosh, for introducing me to the field of machine learning and supervising this project. I would also like to thank my reader, Christine Julien, for reviewing this report.

Abstract

Analysis and Classification of Drift Susceptible Chemosensory Responses

by

Puneet Bansal, M.S.E.

The University of Texas at Austin, 2014

SUPERVISOR: Joydeep Ghosh

This report presents machine learning models that can accurately classify gases by analyzing data from an array of 16 sensors. More specifically, the report presents basic decision tree models and advanced ensemble versions. The contribution of this report is to show that basic decision trees perform reasonably well on the gas sensor data, however their accuracy can be drastically improved by employing ensemble decision tree classifiers. The report presents bagged trees, Adaboost trees and Random Forest models in addition to basic entropy and Gini based trees. It is shown that ensemble classifiers achieve a very high degree of accuracy of 99% in classifying gases even when the sensor data is drift ridden. Finally, the report compares the accuracy of all the models developed.

Table of Contents

List of Tables..viii

List of Figures...ix

Chapter 1: Introduction...1

- 1.1 Data Source and Collection..2
- 1.2 Data Features..2
- 1.3 Literature Survey...5

Chapter 2: Data Preprocessing and Exploration...14

- 2.1 Data Format...14
- 2.2 Data Preprocessing..15
- 2.3 Data Exploration..17

Chapter 3: Analysis and Classification of Sensor Response.............................20

- 3.1 Decision Trees..20
- 3.2 Bagging..33
- 3.3 Boosting...37
- 3.4 Random Forest...41

Chapter 4: Results..44

- 4.1 Misclassification / Test Error Comparison...44
- 4.2 Receiver Operator Characteristic (ROC) Curves...44

Chapter 5: Conclusion..46

References..47

List of Tables

Table 1.1 – Data Features..........3

Table 2.1 – Gases and Concentration..........14

Table 4.1 – Test Error for Different Classifiers..........44

Table 4.2 – Area Under ROC Curves..........45

List of Figures

Figure 2.1 – Raw Data Sample..15

Figure 2.2 – Preprocessed Data Sample...16

Figure 2.3 – Relation of Sensor 1 reading and Ethanol Concentration................17

Figure 2.4 – Relation of Sensor 1 reading and Acetone Concentration...............18

Figure 2.5 – Relation of Sensor 1 reading and Toluene Concentration................18

Figure 3.1 – Classification Tree using Entropy Split...23

Figure 3.2 – Complexity Parameter for Entropy Split...24

Figure 3.3 – Complexity Parameter vs Relative Error for Entropy Split.............24

Figure 3.4 – Variable Importance for Entropy Split..26

Figure 3.5 – Confusion Matrix for Entropy Split...26

Figure 3.6 – Classification tree using Gini Split..28

Figure 3.7 – Complexity Parameter for Gini Split...29

Figure 3.8 – Cost Complexity Parameter vs Relative Error for Gini Split...........30

Figure 3.9 – Variable importance for Gini Split..31

Figure 3.10 – Confusion Matrix for Gini Split..32

Figure 3.11 – Confusion Matrix for Bagged Tree...33

Figure 3.12 – Error As a Function of Number of Trees.......................................35

Figure 3.13 – Variable Importance for Bagged Tree...36

Figure 3.14 – Confusion Matrix for Adaboost Using Brieman Coefficient..................38

Figure 3.15 – Error Evolution of Brieman Coefficient Based Adaboost.......................39

Figure 3.16 – Confusion matrix for Zhu Coefficient Boosted Model.................40

Figure 3.17 – Error Evolution for Boosted Model Using Zhu Coefficient..........41

Figure 3.18 – Random Forest Confusion Matrix..42

Figure 3.19 – Error vs Number of Trees for Random Forest..............................43

Figure 4.1 – ROC Curves...45

Chapter 1: Introduction

Gas sensors play an important role in countless industrial applications. They are used to monitor gas concentration levels, detect toxic fumes, identify gases present in the environment and keep industrial processes running smoothly. A typical gas sensor works by measuring change in electrical resistance across its surface when exposed to a gas. Over time, the sensor surface undergoes irreversible chemical changes due to which its accuracy decreases. This is called 'sensor drift'. After just one year of operation, this drift can become so large that the sensor starts to misidentify the gases. This is a vexing problem in chemosensory applications and there is a need to address this drift so that sensors can correctly identify the gases and do not have to replaced frequently. This report presents several machine learning models that can be used to analyze gas sensor data and correctly classify gases based on sensor readings that are prone to drift. The models demonstrate how weak classifiers can be combined to build strong ensemble classifiers capable of accurately classifying gases even in presence of sensor drift.

1.1 Data Source and Collection

The models presented in this report use the gas sensor array data set available at the machine learning repository of University of California, Irvine [1]. The data was collected by conducting a tightly controlled experiment at BioCircuits Institute, University of California, San Diego. This data set is freely available on-line. The motivation for making this data set freely available is to provide an extensive dataset to the sensor and artificial intelligence research communities to develop and test strategies to solve the gas classification problem in the presence of sensor drift. The dataset is meant to be used exclusively for research purposes. Commercial purposes are fully excluded. It is hoped that this data set will facilitate building robust machine learning models that can identify gases correctly in the same gas concentration range for other data sets as well.

1.2 Data Features

The data set comprises of 129 features and 13,910 observations giving a total of 1,794,390 data points. The first feature is a combination of gas identity and its concentration. The other 128 features correspond to the readings of 16 sensors in the sensor array. Each sensor has eight features associated with it. These features are described in Table 1.1. The first feature is the direct resistance of the sensor which is defined as the maximal change in resistance with respect to baseline. The second feature is the normalized version of the direct resistance and is obtained by dividing the direct resistance by the acquired value. The other six features are exponential moving averages.

Feature	Description
Steady State Feature (DR)	Maximal change in resistance with respect to baseline
Normalized DR	DR divided by the acquired value
EMA Increasing ($\alpha = 0.1$)	Exponential moving average for rising portion of the sensor response with smoothing parameter of $\alpha = 0.1$
EMA Increasing ($\alpha = 0.01$)	Exponential moving average for rising portion of the sensor response with smoothing parameter $\alpha = 0.01$
EMA Increasing ($\alpha = 0.001$)	Exponential moving average for rising portion of the sensor response with smoothing parameter $\alpha = 0.001$
EMA Decreasing ($\alpha = 0.1$)	Exponential moving average for decaying portion of the sensor response with smoothing parameter $\alpha = 0.1$
EMA Decreasing ($\alpha = 0.01$)	Exponential moving average for decaying portion of the sensor response with smoothing parameter $\alpha = 0.01$
EMA Decreasing ($\alpha = 0.001$)	Exponential moving average for decaying portion of the sensor response with smoothing parameter $\alpha = 0.001$

Table 1.1 Data Features

Exponential Moving Average (EMA) is a transform that converts increasing / decaying and saturating time series r[.] collected from the sensor into a real scalar $f_\alpha\{r[.]\}$ by estimating the maximum (or minimum for decaying portion of sensor response) of its exponential moving average transform calculated by the following formula

$$y[k] = (1-\alpha)y[k - 1] + \alpha(r[k] - r[k - 1])$$

where k=1,2,....T; y[0]=0 (the initial condition set to 0) and the scalar α being a smoothing parameter of the operator that defines both the quality of the feature fα{r[.]} and the time of its occurrence along the time series.

For each sensor measurement, the data set contains three EMAs with α=0.1, α=0.01, α=0.001 corresponding to the rising portion of the sensor response and three EMAs with α=0.1, α=0.01, α=0.001 corresponding to the decaying portion of the sensor response giving a total of six EMAs. This results in a 128 dimensional feature vector - (DR, Normalized DR, 6 EMAs) times 16 sensors.

The data is divided into ten batches with some batches containing data for one month while some contain data for more than one month.

The cited approaches described in Literature Survey (section 1.3) use this data set to train different types of classifiers. This is done by dividing the data into two parts – training data set and test data set. The purpose of training data set is to learn how gas class is dependent on the sensor response and gas concentration. A model is built using the training data set. The model is then used to predict the gas class on the test data set. Since the actual gas class is already known for the test data set, the model accuracy can be computed by comparing the predicted class and the actual class. Typically the training data set is larger than the test data set. This is primarily because more data generally results in better models with high prediction accuracy. If the model performs well on the test data, it is a good indicator that model will perform well on new data that model has not seen before.

1.3 Literature Survey

Since the gas sensor array data set was published in 2013, there has been interest from the machine learning community to accurately classify the gases in presence of sensor drift. Support Vector Machine (SVM) classifier has been the choice of most researchers since it is an effective classifier. Many researchers have chosen to employ an ensemble of classifiers to the sensor array dataset. What varies is the methodology used for ensemble construction. We now discuss the techniques employed and results obtained giving an overview of the existing state of research on gas sensor array drift dataset.

The earliest attempts to predict sensor drift were primarily limited to univariate and multivariate analyses where drift compensation is performed either on each sensor individually or on the entire sensor array [2], [3]. The most popular approach for multivariate analysis was based on unsupervised component correction techniques [4], [5], [6]. These techniques employ a linear transform that normalizes the sensor response across time so that the classifier can be directly applied to the resulting stationary data. These techniques found limited success because they assumed the drift direction to be linear in feature space and hence tried to apply only linear techniques (such as PCA).

Vergara et al. [7] took a novel approach (at the time) and applied supervised learning, more specifically, an ensemble of classifiers to cope with the sensor drift. Their technique complemented prior attempts using component correction techniques since the latter are primarily data preprocessing steps and the ensemble developed in [7] could easily be applied after component correction has been applied. The authors trained a new SVM classifier $f(t_i)$ on the batch of sensor data obtained at time t_i ($i=0,1,2,...$). The final classifier for $f(t+1)$ for time $t+1$ was the weighted combination of the classifiers $f(t_0)$ through $f(t)$. The weights were estimated using majority voting scheme. The results showed that this ensemble of SVMs performed significantly better than unsupervised component correction techniques. The drift compensation proposed by this ensemble of SVMs remains valid for long periods of time. Also, this technique did not make any assumptions about the nature of drift.

One disadvantage of the approach employed in [7] is that it is a supervised approach. This approach requires the classifier weights in the ensemble to be constantly updated if the ensemble is to be used to make accurate predictions in the future. This assumes that labels (gas classification) is available for the training data. In other words, in a purely supervised approach the labels have to be available on an ongoing basis to keep the weights accurate. A semi-supervised approach was adopted by Liu et al. in [8]. In [8], domain adaption was applied to tackle the sensor drift problem. Domain adaption establishes a feature space between the source and the target domains [9]. The classifier is then trained on this feature space. This technique gets around the issue of assuming the target domain has the same distribution as the source domain which is not true for sensor drift. A weighted geodesic kernel flow was constructed [10]. Since the gas sensor data is time series a combination weighted geodesic kernel flow was employed. A technique for selecting unlabeled data from the target domain is also discussed [11]. Based on the combination kernel and selected unlabeled data, a semi supervised method called manifold regularization was presented and used to train the classifier. Multi class SVM was trained using RBF kernel (rbf-svm), geodesic flow kernel (gfk-svm) and combination kernel (comgfk-svm). Manifold regularization for RBF kernel (rbf-ml) and combination kernel (comgfk-ml) was used for semi-supervised methods. rbf-svm and rbf-ml were used as baselines for evaluation. Results showed that geodesic kernel and combination kernel can effectively deal with sensor drift. Accuracy of comgfk-svm was generally higher than that of gfk-svm indicating that combination kernel was efficient. It was shown that domain adaption and semi-supervised methods can efficiently solve sensor drift problem. However, either technique alone cannot solve the problem. The performance of comgfk-ml and ensemble are comparable but the former has the advantage that it requires labeled data only in the source domain and not the target domain whereas the latter requires labeled data in the target domain as well.

One advantage of semi-supervised approach is that it reduces the cost of re-calibrating the sensors periodically by requiring less labeled data. Lujan et al. [12] have tried to reduce the cost of sensor calibration using a different approach. Lujan et al. [12] attempted to answer the question - “When selecting a new training sample, what gas class and concentration must be selected that maximizes learning?”. To answer this question, an Inhibitory Support Vector Machine (ISVM) with RBF kernel was employed. This ISVM was used to classify ethanol, ethylene and acetaldehyde as a function of the multivariate response of the sensor array. The choice of ISVM was motivated by its consistency for three class classification problems and its robustness on small datasets. The algorithm presented takes the entire data set, rate of sampling distribution at current stage, the sequence of optimal values for the previous stages and the batch size as the input. The algorithm then uses ISVM to output 100 sampling draws for the next stage though other classifiers can be used as well. The authors showed that the overall performance of the model significantly improved when more samples were made available. It was also shown that the sampling strategy greatly influences the performance of the classifier in the first stages but not much when enough samples have already been presented to the sensor array. Results showed that in this active sampling methodology it is especially important to include low concentrations of gases in the calibration process and that the calibration must be done over the entire range of concentrations that the sensor may be exposed to. Ignoring either of these results in more frequent calibration and hence higher cost.

Liu and Tang [13] proposed yet another ensemble called Dynamic Weights based on Fitting (DWF). Their goal was to solve the gas discrimination problem regardless of the gas concentration with high accuracy over extended periods of time. The DWF method uses a dynamic weighted combination of SVMs trained by datasets that are collected at different time periods. An attempt was made to find optimal weights using traversal search for each of the training batches. The row vectors in the resulting optimal weight matrix was the subset of the weight of the classifier at different time steps and the column vector represented the optimal weights assigned to all the classifiers at the corresponding time step. The weight for a training batch and the corresponding mean measurement time of the batch form a two dimensional array. These arrays were used to fit a weight curve that is a function of time. This curve was then used to predict the weight of a classifier at a future time. The performance of DWF was compared with ensembles based on MLP, ANN and theoretical optimal weights of SVMs. It was shown that while all the ensembles degraded in performance over time, DWF degraded more slowly as compared to others. In DWF, the weights were predicted by fitting functions and the time span of the training dataset affected the fitting result. In general, it was found that performance of DWF method can be improved by increasing the time span of the training dataset. In other words, DWF can mitigate the effect of sensor drift for longer periods of time when the the time span of training data is increased.

Extreme learning machine (ELM) approach was applied by Daniel et al. in [15]. The problem of data correction in the presence of simultaneous sources of drift, other than sensor drift, was investigated, since it is often the case in practical situations. ELM with different activation functions was implemented for gas sensor array drift dataset. Results showed that ELM with bipolar function classifies the drift dataset with 96% accuracy. Since this paper is not freely available, further details could not be gathered.

So far our discussion has been restricted to the analyses done on data set in [1]. We now discuss approaches employed by researchers on other gas classification data. While discussing supervised approach in [7], we discussed the shortcomings of assuming the linearity of sensor drift and why techniques such as PCA and ICA (independent component analysis) have limited effectiveness in capturing sensor drift behavior. Dang et al. proposed a novel classifier ensemble in [12]. The dataset used was e-nose data comprising of 260, 164, 66, 58, 48 and 30 samples of formaldehyde, benzene, toluene, carbon monoxide, ammonia and nitrogen dioxide respectively. They used a kernelized version of PCA (KPCA) by integrating the kernel trick into PCA. KPCA was used for feature extraction. The key idea involved mapping the input feature space to higher dimensional feature space and then performing PCA on it. SVM was used to train classifiers for classification of the gases. An improved support vector machine ensemble (ISVMEN) was proposed. ISVMEN used weighted voted fusion method to combine the base classifiers. Results showed that using ISVMEN improved average classification accuracy from 86% to 93%. As compared to majority voting scheme, ISVMEN gained a 2% increase in classification accuracy.

Kim et al. adopted a different approach in [14] that was not based on SVMs. They applied a neuro-genetic classification algorithm (NGCA) to classify the following gases - ozone, LPG, NOx, alcohol, smoke, VOC, CO and ammonia. The pattern recognition approach proposed involved data signal acquisition from the sensor array and signal preprocessing with smoothed moving average (SMMA). SMMA was used to identify noise and obtain the mean value while eliminating the oldest data and adding new data over time. The NGCA algorithm was then applied. NGCA applied was a combination of artificial neural network (ANN) and genetic algorithm (GA). First the GA function was applied to extract data with high fitness after noise filtering. Then the ANN function was implemented using the back propagation algorithm. It was found that NGCA classification accuracy was 95% as compared to 82% with ANN alone and 91% with GA alone.

Most approaches discussed in this section have employed some variation of SVM. This is because SVMs are known to be effective classifiers. However, SVMs have their limitations. They tend to perform somewhat poorly on multi-class classification problems where there are more than two classes to predict.

SVMs also tend to be sensitive to noise in the data. The predictions made by SVMs can change dramatically by the introduction of just a few noisy observations. On the other hand, tree based models are known to be robust to noise in the data. They are also known to perform well in multi-class scenarios. In the remainder of the report, we tackle the gas classification problem using decision tree approach.

Chapter 2: Data Preprocessing and Exploration

2.1 Data Format

The data is made available as text files separated into ten batches with each batch representing one or more months. The ten batches span a 36 month period. The first feature comprises of gas identity and concentration separated by a semi-colon. The other 128 features are prefixed with the feature number separated by the data using a colon. The gas identity is designated using a class label instead of the name of the gas. There are six gases in the data set with the labels and concentration ranges as shown in Table 2.1.

Gas	Label	Concentration Range (ppmv)
Ethanol	1	10-600
Ethylene	2	10-300
Ammonia	3	50-1000
Acetaldehyde	4	5-500
Acetone	5	12-1000
Toluene	6	10-100

Table 2.1 – Gases and Concentration

Figure 2.1 shows a single record from the dataset.

```
1;10.000000 1:15596.162100 2:1.868245 3:2.371604 4:2.803678 5:7.512213 6:-2.739388 7:-
3.344671 8:-4.847512 9:15326.691400 10:1.768526 11:2.269085 12:2.713374 13:6.915721
14:-2.488324 15:-3.082212 16:-5.056975 17:2789.383100 18:2.754759 19:0.430440
20:0.649457 21:1.795029 22:-0.426662 23:-0.584313 24:-1.438976 25:2581.568600
26:2.680623 27:0.399746 28:0.605065 29:1.786704 30:-0.400115 31:-0.550743 32:-1.728611
33:685.399400 34:1.682904 35:0.122736 36:0.223703 37:0.584691 38:-0.138196 39:-
0.236907 40:-0.781959 41:797.773800 42:1.742488 43:0.152483 44:0.218904 45:0.841862
46:-0.164646 47:-0.315720 48:-0.791447 49:3128.848900 50:3.605537 51:0.532422
52:0.763062 53:2.118983 54:-0.557197 55:-0.809953 56:-2.344130 57:3136.877800
58:3.555169 59:0.535883 60:0.761388 61:1.499244 62:-0.571480 63:-0.944425 64:-2.658358
65:13540.673800 66:1.765738 67:2.006883 68:2.519022 69:6.261430 70:-2.172101 71:-
2.694967 72:-3.791499 73:13831.753900 74:1.746493 75:2.057165 76:2.391239 77:5.695234
78:-2.350776 79:-2.888766 80:-8.129869 81:3020.919100 82:2.819354 83:0.474520
84:0.723993 85:2.160130 86:-0.467900 87:-0.638167 88:-1.643650 89:2185.974100
90:2.949381 91:0.342575 92:0.515090 93:1.340477 94:-0.361030 95:-0.493482 96:-1.200617
97:862.747900 98:1.779291 99:0.165138 100:0.246473 101:1.358106 102:-0.187465 103:-
0.416382 104:-1.058061 105:1059.756200 106:1.896047 107:0.198946 108:0.334017
109:0.815048 110:-0.204467 111:-0.345119 112:-0.969336 113:3357.112400 114:3.860647
115:0.580818 116:0.806830 117:1.729739 118:-0.619214 119:-1.071137 120:-3.037772
121:3037.039000 122:3.972203 123:0.527291 124:0.728443 125:1.445783 126:-0.545079
127:-0.902241 128:-2.654529
```

Figure 2.1 – Raw Data Sample

2.2 Data Preprocessing

The data, in its given format, has the first feature as a combination of two features - gas class and concentration. The other 128 features have the feature number prefixed to the the actual data. Data preprocessing is required to convert the data into a format on which analysis can be done. A Java program was written to iterate through all the ten batches and split the first feature into two - GAS (class label) and CONC (concentration). For the other 128 features, the prefix was discarded. A new feature BATCH was created to allow combining all the ten batches into a single data set without losing the identity of batches. Though GAS and BATCH have numeric values (1-6 and 1-10) respectively, they are actually categorical variables. These features were therefore converted to 'factors' in R before doing any analysis. The other features were left as numeric. Features are separated by a single space. Figure 2.2 shows an example of the data after processing.

```
1 10.000000 15596.162100 1.868245 2.371604 2.803678 7.512213 -2.739388 -3.344671
-4.847512 15326.691400 1.768526 2.269085 2.713374 6.915721 -2.488324 -3.082212
-5.056975 2789.383100 2.754759 0.430440 0.649457 1.795029 -0.426662 -0.584313
-1.438976 2581.568600 2.680623 0.399746 0.605065 1.786704 -0.400115 -0.550743
-1.728611 685.399400 1.682904 0.122736 0.223703 0.584691 -0.138196 -0.236907 -0.781959
797.773800 1.742488 0.152483 0.218904 0.841862 -0.164646 -0.315720 -0.791447
3128.848900 3.605537 0.532422 0.763062 2.118983 -0.557197 -0.809953 -2.344130
3136.877800 3.555169 0.535883 0.761388 1.499244 -0.571480 -0.944425 -2.658358
13540.673800 1.765738 2.006883 2.519022 6.261430 -2.172101 -2.694967 -3.791499
13831.753900 1.746493 2.057165 2.391239 5.695234 -2.350776 -2.888766 -8.129869
3020.919100 2.819354 0.474520 0.723993 2.160130 -0.467900 -0.638167 -1.643650
2185.974100 2.949381 0.342575 0.515090 1.340477 -0.361030 -0.493482 -1.200617
862.747900 1.779291 0.165138 0.246473 1.358106 -0.187465 -0.416382 -1.058061
1059.756200 1.896047 0.198946 0.334017 0.815048 -0.204467 -0.345119 -0.969336
3357.112400 3.860647 0.580818 0.806830 1.729739 -0.619214 -1.071137 -3.037772
3037.039000 3.972203 0.527291 0.728443 1.445783 -0.545079 -0.902241 -2.654529 1
```

Figure 2.2 – Preprocessed Data Sample

2.3 Data Exploration

Sensor response is likely to depend on the identity of the gas and its concentration. It is natural to explore the relation between sensor readings and these two features. There are 16 sensors in the array. Figures 2.3, 2.4 and 2.5 show the plots for response from sensor 1 (its direct resistance) and concentration for ethanol, acetone and toluene respectively.

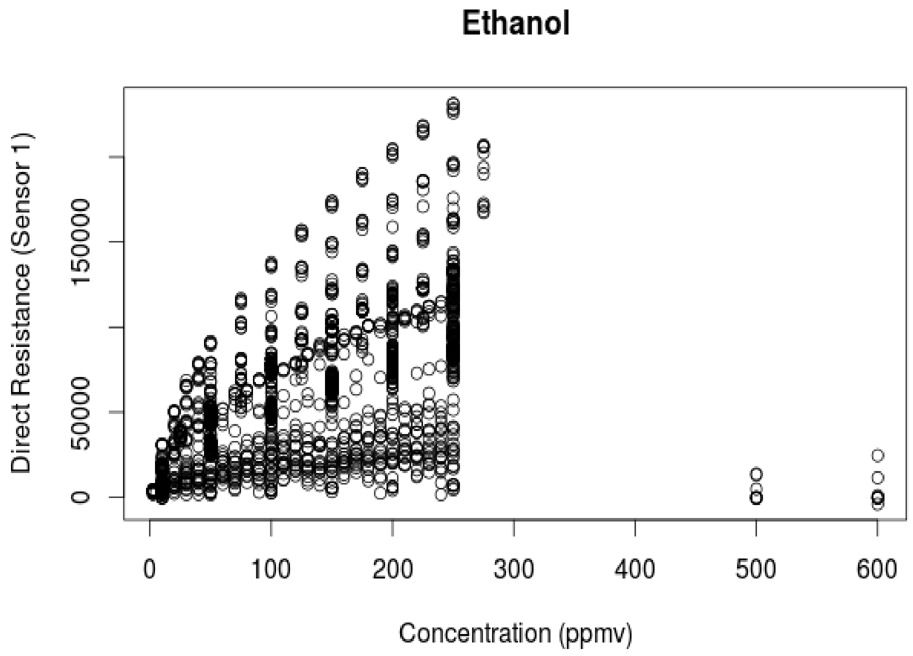


Figure 2.3 – Relation of Sensor 1 reading and Ethanol Concentration

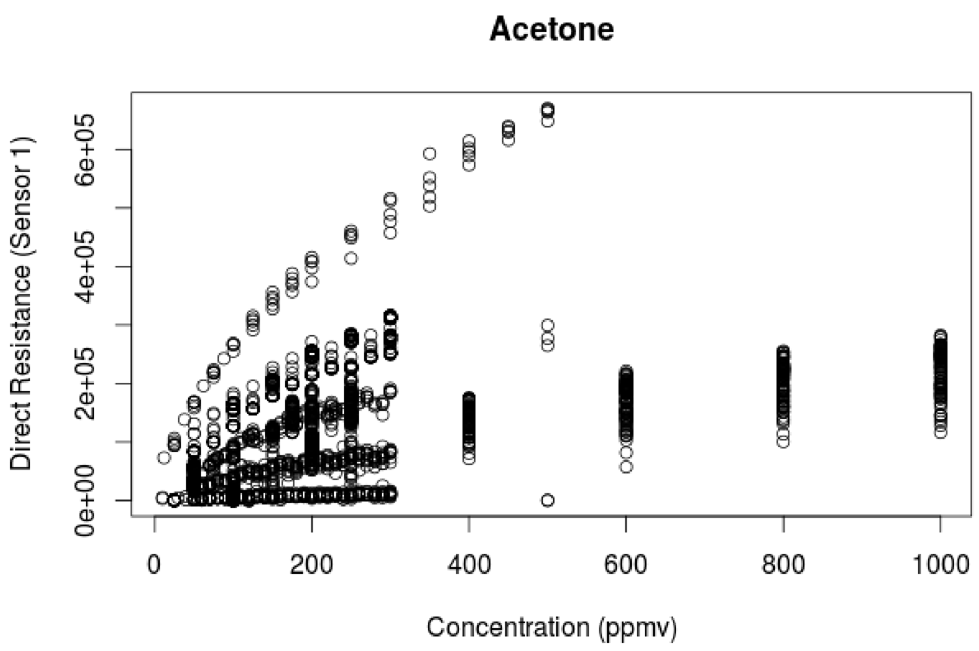


Figure 2.4 – Relation of Sensor 1 reading and Acetone Concentration

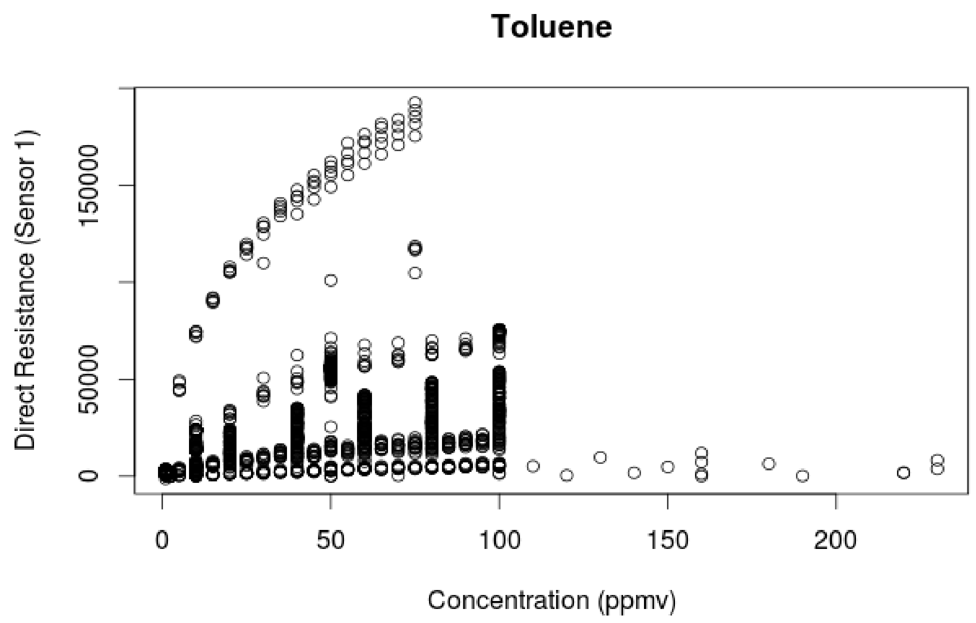


Figure 2.5 – Relation of Sensor 1 reading and Toluene Concentration

The plots show there is some relation between concentration and sensor response, but the relation is not linear. It can also be seen that the direct resistance of sensor 1 varies dramatically for the same gas concentration. This gives us a hint that there are factors other than sensor 1 and gas concentration influencing these plots. We know that sensor 1 does not operate in isolation – it is part of an array of 16 sensors. Therefore, we must take into account the effect of other sensors as well. Trying to determine a relationship between sensor response and gas and its concentration is futile without taking into consideration the entire sensor array.

A principal component analysis was performed as part of data exploration. Its results, however, were not interesting and therefore will not be discussed here further.

Chapter 3: Analysis and Classification of Sensor Response

3.1 Decision Trees

Given that the response variable (GAS) is qualitative and there is a non-linear relationship between the response and features, it is worthwhile to explore decision tree modeling to classify the gases. Decision trees have been known to be effective in such scenarios. Moreover, the current research has been mostly based on classification techniques such as Support Vector Machines and variants and decision trees seem to have been largely ignored. We therefore analyze this data using decision tree modeling. Two types of basic decision trees are presented below based on the splitting strategy. First tree has been grown using entropy index. The second tree has been grown using Gini index. The basic strategy adopted to get to the final tree is same for both the cases but the splitting criterion is different. The basic algorithm to build decision trees is presented below.

1. Use sampling to split data into test (20%) and training (80%) sets.
2. Use each feature to split the data into the response classes.
3. Select the feature that best splits the training data based on the chosen splitting criterion (Gini or entropy) and discard other splits from Step 2.
4. Repeat steps 2 and 3 recursively until all nodes have fewer than a threshold number of observations.

5. Use cross-validation and cost complexity pruning to select the optimal tree size.

Gini split criterion is given by

$$Gini(t)=1-\sum_{k=1}^{c-1}(p(i_t))^2$$

Entropy split criterion is given by

$$Entropy(t)=-\sum_{i=0}^{c-1}p(i_t)\log_2 p(i_t)$$

where

$p(i_t)$ = the fraction of records belonging to class i at node t

c = number of classes

Figure 3.1 shows the optimal tree built based on the entropy split criterion using 'rpart' package in R. The tree shows the splits used to classify the gases. The feature names on the tree have been abbreviated. For example -

S2DR - Direct Resistance of sensor 2.

S11NDR - Normalized Direct Resistance of sensor 11.

S9I_1 - Exponential Moving Average with a smoothing parameter of 0.1 when current is increasing for Sensor 9.

S9D_01 - Exponential Moving Average with a smoothing parameter of 0.01

when current is decreasing for Sensor 9.

CONC - Concentration (ppmv).

The first split is based on the value of the increasing component of the current in sensor 9 when the smoothing parameter is 0.1. Other splits can be read from the tree in a similar manner. The cost complexity parameter and its relation to number of splits and error are shown in Figure 3.2. Also Figure 3.3 shows the relation between the cost complexity parameter, cross-validation error and the number of splits.

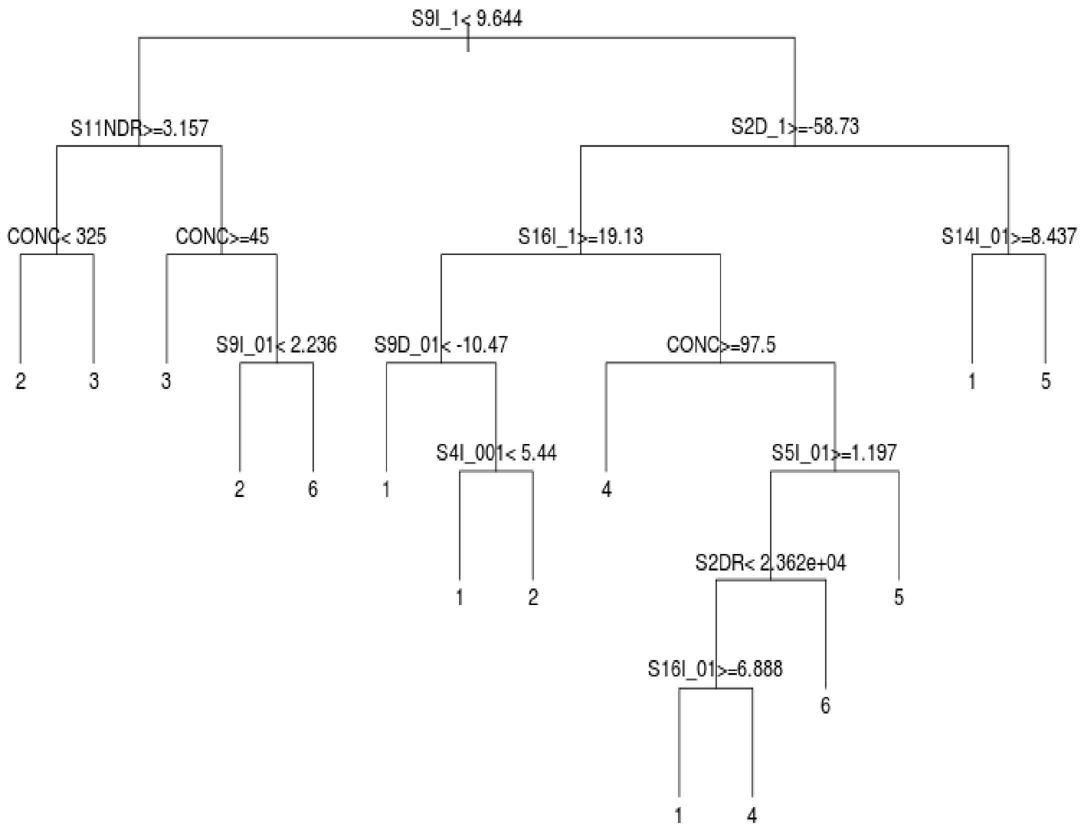
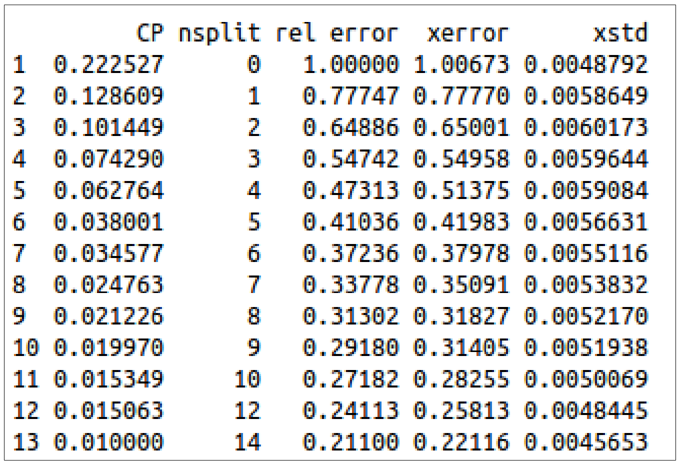


Figure 3.1 – The classification tree using entropy split



```
          CP nsplit rel error  xerror      xstd
1  0.222527      0   1.00000 1.00673 0.0048792
2  0.128609      1   0.77747 0.77770 0.0058649
3  0.101449      2   0.64886 0.65001 0.0060173
4  0.074290      3   0.54742 0.54958 0.0059644
5  0.062764      4   0.47313 0.51375 0.0059084
6  0.038001      5   0.41036 0.41983 0.0056631
7  0.034577      6   0.37236 0.37978 0.0055116
8  0.024763      7   0.33778 0.35091 0.0053832
9  0.021226      8   0.31302 0.31827 0.0052170
10 0.019970      9   0.29180 0.31405 0.0051938
11 0.015349     10   0.27182 0.28255 0.0050069
12 0.015063     12   0.24113 0.25813 0.0048445
13 0.010000     14   0.21100 0.22116 0.0045653
```

Figure 3.2 – Complexity parameter for entropy split

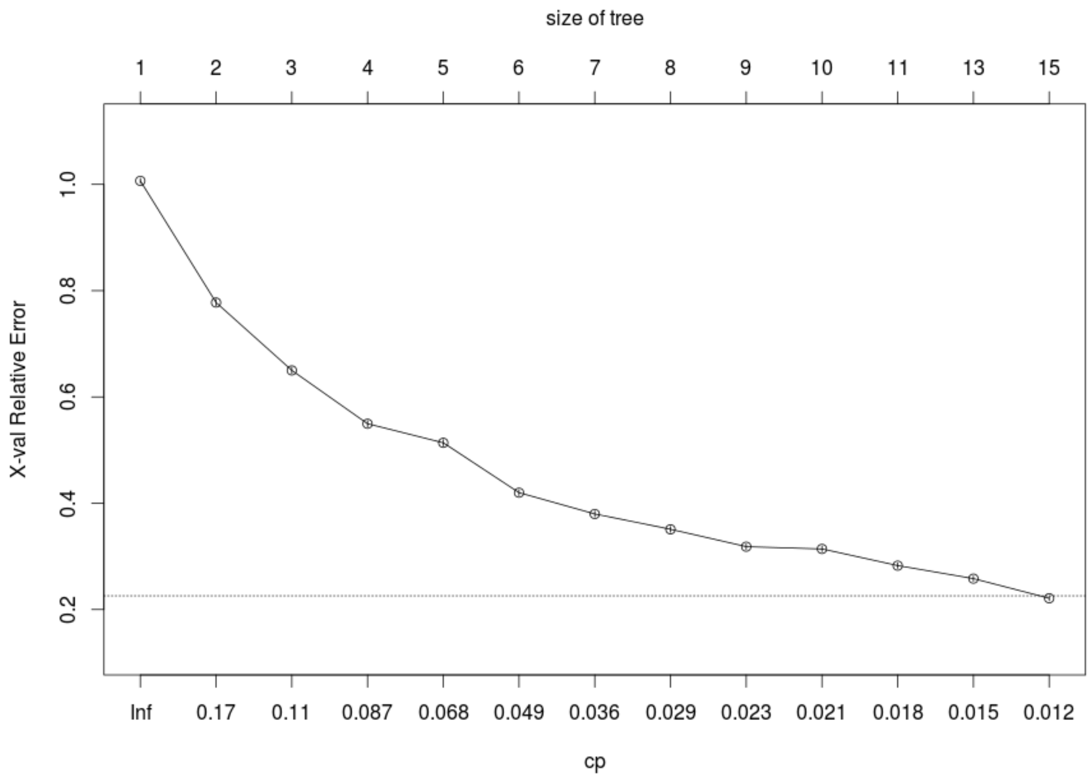


Figure 3.3 – Complexity parameter vs relative error for entropy split

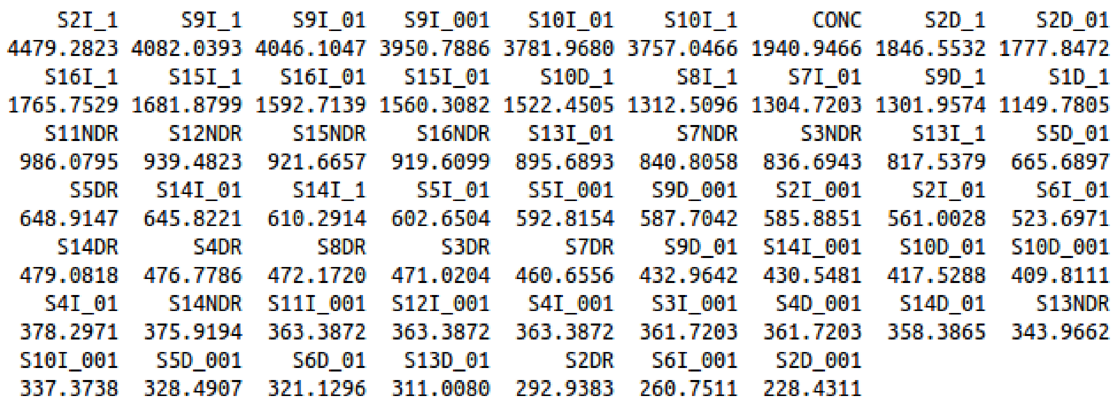
It is seen that as the cost complexity parameter decreases, so does the cross-validation error. Each value of cost complexity parameter corresponds to certain number of splits in the tree. To pick the tree with optimal number of splits, we use 1-Standard Error rule. We pick the tree with lowest cross-validation error and add standard error to it. These are 'xerror' and 'xstd' values on line 13 in Figure 3.2. Then we pick the tree with least number of splits whose cross-validation is less than this value.

0.22116 + 0.0045653 = 0.2257253

The tree which has the least number of splits and cross-validation error ('xerror') less than 0.2257253 is the tree with 14 splits on line 13. This is the optimal tree and is shown in Figure 3.1.

It is natural to wonder – of all the 130 features, which features influence the gas class the most? To find this out we enumerate the importance of the features used in building the tree. Figure 3.4 shows the variable importance for this tree in decreasing order of importance. Below each feature is its importance weight. It can be seen that the six most important variables have 'I' in the feature name. As per our abbreviation convention, these features correspond to the increasing portion of the current in the sensor. Thus, increasing portion of the current seems to have higher importance as also the concentration of the gas (the seventh most important variable). The decreasing portion of the current appears to be less important as these features seem to carry lower weights.

Variable Importance

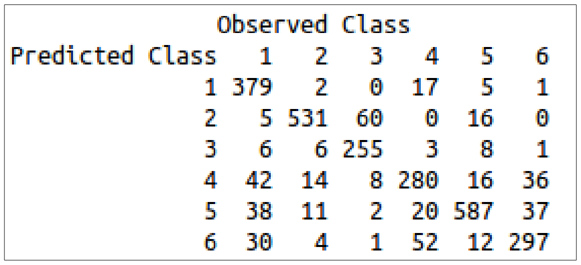


```
    S2I_1     S9I_1    S9I_01   S9I_001   S10I_01    S10I_1      CONC     S2D_1    S2D_01 
4479.2823 4082.0393 4046.1047 3950.7886 3781.9680 3757.0466 1940.9466 1846.5532 1777.8472 
   S16I_1    S15I_1   S16I_01   S15I_01    S10D_1     S8I_1    S7I_01     S9D_1     S1D_1 
1765.7529 1681.8799 1592.7139 1560.3082 1522.4505 1312.5096 1304.7203 1301.9574 1149.7805 
   S11NDR    S12NDR    S15NDR    S16NDR   S13I_01     S7NDR     S3NDR    S13I_1    S5D_01 
 986.0795  939.4823  921.6657  919.6099  895.6893  840.8058  836.6943  817.5379  665.6897 
     S5DR   S14I_01    S14I_1    S5I_01   S5I_001   S9D_001   S2I_001    S2I_01    S6I_01 
 648.9147  645.8221  610.2914  602.6504  592.8154  587.7042  585.8851  561.0028  523.6971 
    S14DR      S4DR      S8DR      S3DR      S7DR    S9D_01  S14I_001   S10D_01  S10D_001 
 479.0818  476.7786  472.1720  471.0204  460.6556  432.9642  430.5481  417.5288  409.8111 
   S4I_01    S14NDR  S11I_001  S12I_001   S4I_001   S3I_001   S4D_001   S14D_01    S13NDR 
 378.2971  375.9194  363.3872  363.3872  363.3872  361.7203  361.7203  358.3865  343.9662 
 S10I_001   S5D_001    S6D_01   S13D_01      S2DR   S6I_001   S2D_001 
 337.3738  328.4907  321.1296  311.0080  292.9383  260.7511  228.4311 
```

Figure 3.4 – Entropy Tree Variable Importance

Test Error

This entropy tree model is used to predict the gas class on the test data. The confusion matrix for this model is shown in Figure 3.5.



```
               Observed Class
Predicted Class   1   2   3   4   5   6
              1 379   2   0  17   5   1
              2   5 531  60   0  16   0
              3   6   6 255   3   8   1
              4  42  14   8 280  16  36
              5  38  11   2  20 587  37
              6  30   4   1  52  12 297
```

Figure 3.5 – Entropy Tree Confusion Matrix

Based on this confusion matrix, it is found that this model misclassifies 16.28% of the test records. The test error for this model is therefore 0.1628.

We now build a similar model but this time we use the Gini index. Again we use 'rpart' package in R to build this model. The optimal tree obtained is shown in Figure 3.6. The features on the tree have been abbreviated as explained earlier. Again, we find that the first split is based on the value of the increasing component of the current in sensor 9 when the smoothing parameter is 0.01. Subsequent splits can be read in a similar manner.

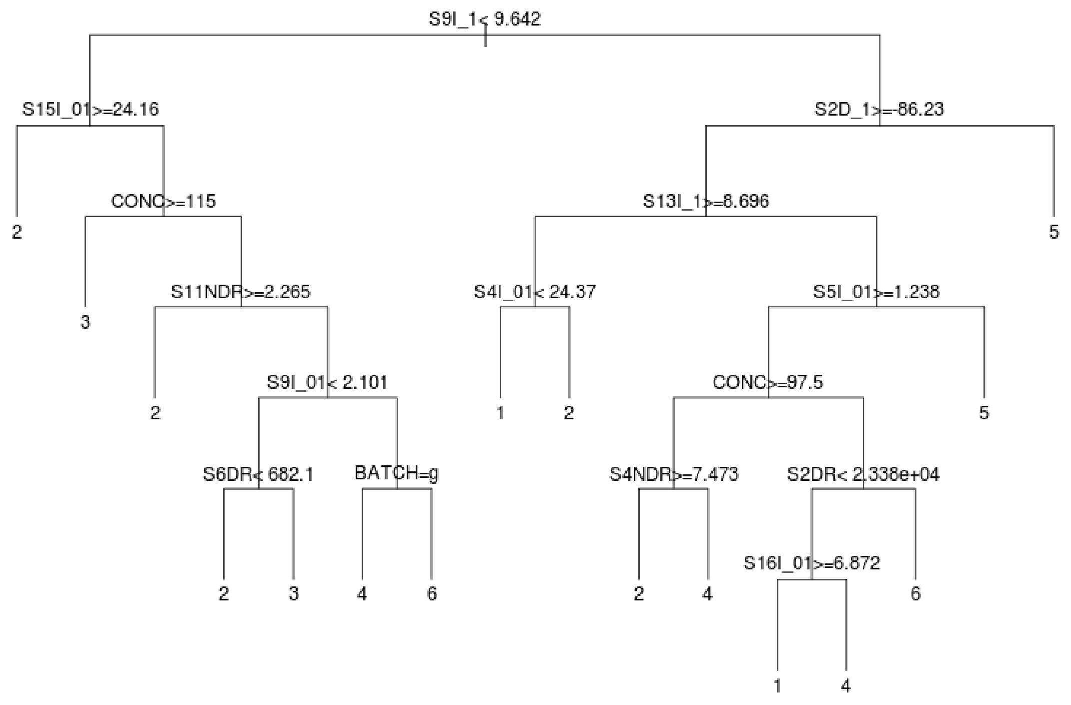
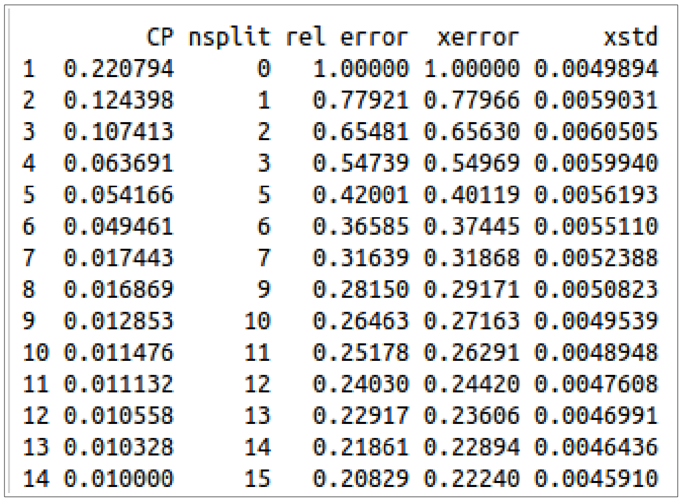


Figure 3.6 – The classification tree using Gini split



```
          CP nsplit rel error  xerror      xstd
1  0.220794      0   1.00000 1.00000 0.0049894
2  0.124398      1   0.77921 0.77966 0.0059031
3  0.107413      2   0.65481 0.65630 0.0060505
4  0.063691      3   0.54739 0.54969 0.0059940
5  0.054166      5   0.42001 0.40119 0.0056193
6  0.049461      6   0.36585 0.37445 0.0055110
7  0.017443      7   0.31639 0.31868 0.0052388
8  0.016869      9   0.28150 0.29171 0.0050823
9  0.012853     10   0.26463 0.27163 0.0049539
10 0.011476     11   0.25178 0.26291 0.0048948
11 0.011132     12   0.24030 0.24420 0.0047608
12 0.010558     13   0.22917 0.23606 0.0046991
13 0.010328     14   0.21861 0.22894 0.0046436
14 0.010000     15   0.20829 0.22240 0.0045910
```

Figure 3.7 – Complexity Parameter for Gini split

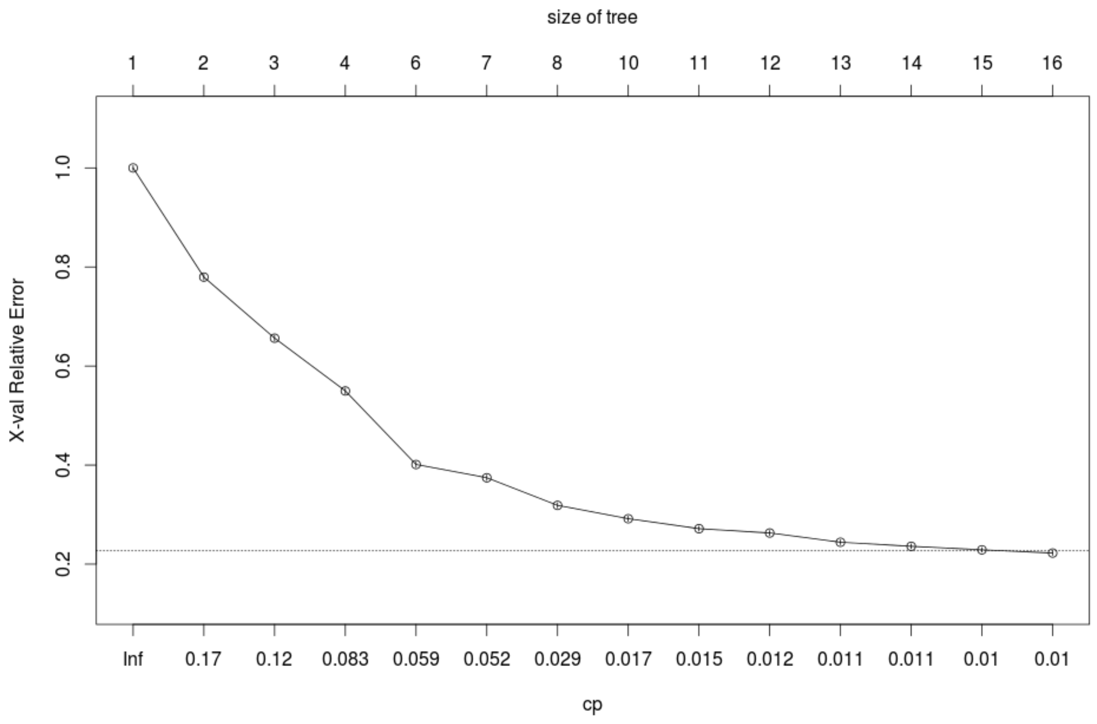


Figure 3.8 – Cost Complexity Parameter vs Relative Error for Gini split

Figure 3.7 shows the relation between cost complexity parameter and the number of splits along with the respective errors. The graph in Figure 3.8 also shows the relation between cross-validation error, cost complexity parameter and the number of splits. We find that the relation between cost complexity parameter, number of splits and cross-validation error is similar to the one in previous model (entropy tree).

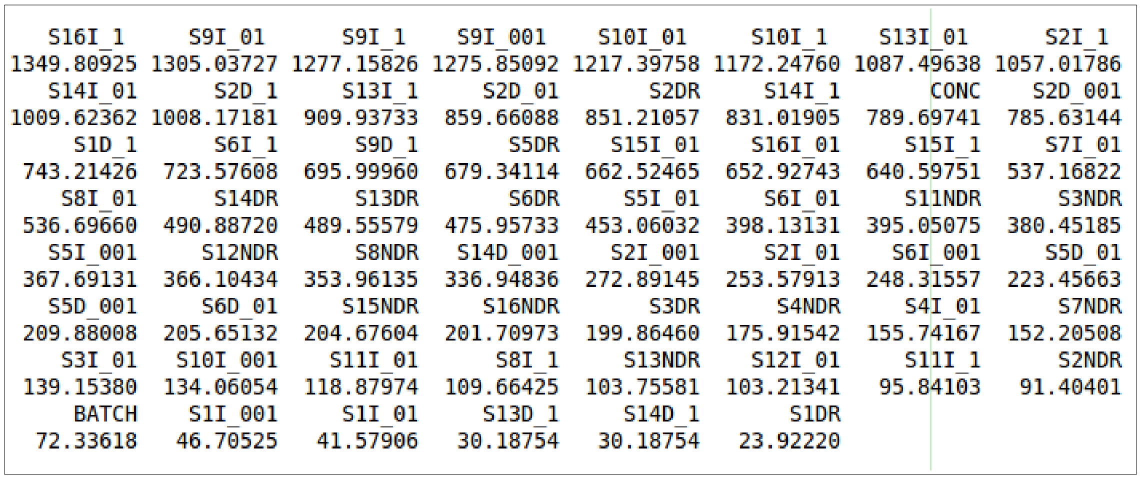
As before, to select the optimal tree we use 1-Standard Error rule. We pick the tree with lowest cross-validation error and add standard error to it. Then we pick the tree with least number of splits whose cross-validation is less than this value. Put simply, we pick the simplest model whose error is within one standard error of the minimal error.

0.22240 + 0.0045910 = 0.226991

Thus the tree with 15 splits is the most stable tree. This tree is shown in Figure 3.6. It is worth observing that Gini tree has one more split than the entropy tree.

Variable Importance

Figure 3.9 shows the importance of various features in decreasing order of importance. Below each variable is its weight. Again we find that increasing portion of the current seems to be important as also the concentration.



```
   S16I_1     S9I_01      S9I_1     S9I_001    S10I_01     S10I_1    S13I_01      S2I_1
1349.80925 1305.03727 1277.15826 1275.85092 1217.39758 1172.24760 1087.49638 1057.01786
   S14I_01      S2D_1     S13I_1     S2D_01       S2DR     S14I_1       CONC    S2D_001
1009.62362 1008.17181  909.93733  859.66088  851.21057  831.01905  789.69741  785.63144
     S1D_1      S6I_1      S9D_1       S5DR    S15I_01    S16I_01     S15I_1     S7I_01
 743.21426  723.57608  695.99960  679.34114  662.52465  652.92743  640.59751  537.16822
    S8I_01      S14DR      S13DR       S6DR     S5I_01     S6I_01     S11NDR      S3NDR
 536.69660  490.88720  489.55579  475.95733  453.06032  398.13131  395.05075  380.45185
   S5I_001     S12NDR      S8NDR   S14D_001    S2I_001     S2I_01    S6I_001     S5D_01
 367.69131  366.10434  353.96135  336.94836  272.89145  253.57913  248.31557  223.45663
   S5D_001     S6D_01     S15NDR     S16NDR       S3DR      S4NDR     S4I_01      S7NDR
 209.88008  205.65132  204.67604  201.70973  199.86460  175.91542  155.74167  152.20508
    S3I_01   S10I_001    S11I_01      S8I_1     S13NDR    S12I_01     S11I_1      S2NDR
 139.15380  134.06054  118.87974  109.66425  103.75581  103.21341   95.84103   91.40401
     BATCH    S1I_001     S1I_01     S13D_1     S14D_1       S1DR
  72.33618   46.70525   41.57906   30.18754   30.18754   23.92220
```

Figure 3.9 – Variable importance for Gini Split

Test Error

The model built using Gini split decision is used to predict gas class on the test data. The confusion matrix is shown in Figure 3.10.

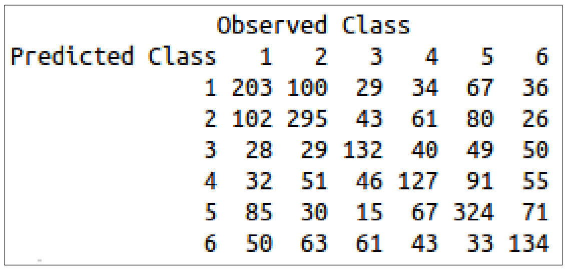

```
                 Observed Class
Predicted Class   1   2   3   4   5   6
              1 203 100  29  34  67  36
              2 102 295  43  61  80  26
              3  28  29 132  40  49  50
              4  32  51  46 127  91  55
              5  85  30  15  67 324  71
              6  50  63  61  43  33 134
```

Figure 3.10 – Gini Tree Confusion Matrix

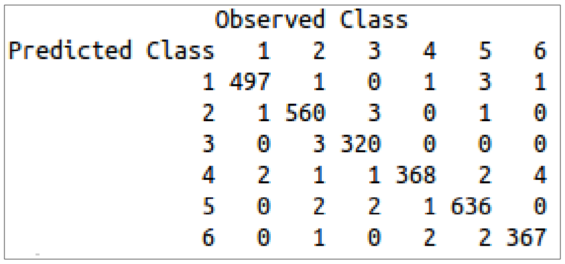
This model misclassifies the training data 15.49% of the time. The test error is therefore 0.1549. Thus it performs slightly better than the entropy tree.

Both Gini and entropy split seem to perform reasonably well with their test errors being very close to each other. The trees constructed so far suffer from high variance. This means that we cannot rely on this model to accurately predict new observations. In order to reduce the variance, we need to employ a technique that will average out the variance of several trees and hence reduce it. Bootstrap aggregating or bagging is one such technique. We now build a 'bagged' tree and see if we can reduce the test error of our model. In bagging, we develop multiple tree models and make predictions using all the models and then average out the prediction. Since this is a classification case, the final prediction is made by taking a majority voting of all the models.

3.2 Bagging

The decision trees developed so far suffer from the disadvantage of a high variance model. In other words, if decision trees were built on a different data set from the same sensors, the results could be significantly different. It is known that averaging a set of observations reduces variance. Bootstrap aggregation or bagging attempts to do just that. Bagging involves repeatedly sampling the data and building different models on the data. The final prediction is made by averaging the prediction of all the models. The final predicted class for an observation is the one predicted by majority of the individual models.

As before, the data was split into 80% training and 20% test data sets. 500 trees were built using the bagging technique and the final prediction was made by taking the majority vote. Figure 3.11 shows the confusion matrix of this model on the test data.



```
               Observed Class
Predicted Class   1   2   3   4   5   6
              1 497   1   0   1   3   1
              2   1 560   3   0   1   0
              3   0   3 320   0   0   0
              4   2   1   1 368   2   4
              5   0   2   2   1 636   0
              6   0   1   0   2   2 367
```

Figure 3.11 – Confusion matrix for bagged tree

Misclassification rate of this model is found to be only 1.1%. It is obvious that bagging has reduced the error drastically from 15-16% for the entropy and Gini trees.

Since averaging tends to reduce the test error, it is clearly of interest to see how the test error depends on number of trees built. Figure 3.12 shows a plot of test error vs the number of trees. Note that the test error (y axis) uses the log scale. The graph clearly shows that for all classes there is a marked drop in test error for the first 100 trees. For more than hundred trees, the rate of error drop is significantly reduced and by 500 trees, the test error has become mostly stable. In addition to the six classes, an additional error OOB is shown on the plot. This error is 'Out of Bag' error which is an acceptable representation of test error considering that bagging uses only 66% of the observations in building the model. The OOB error is then the prediction error on the other 34% of the observations. The OOB error, 0.012, is found to be very close to the computed test error (0.011) as expected.

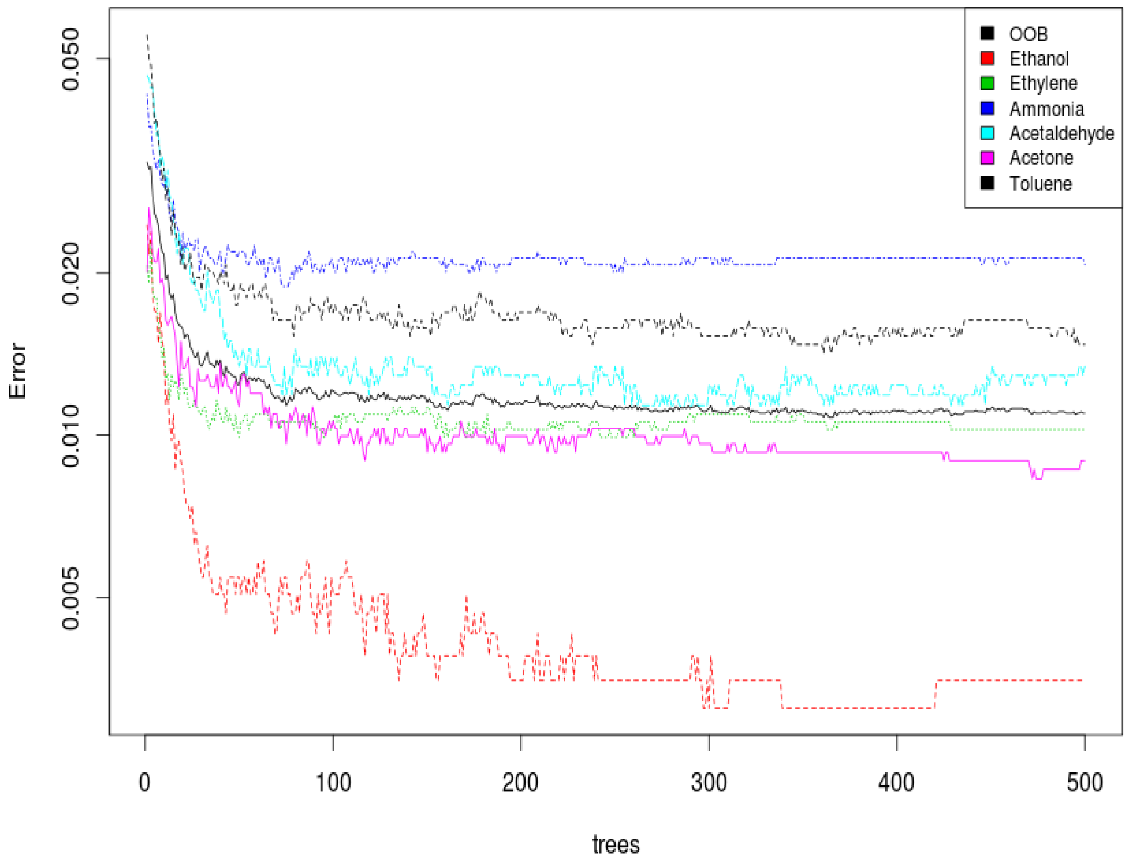


Figure 3.12 – Error as a function of number of trees.

(OOB = out of bag error, others are classification errors)

Unlike entropy and Gini trees, it is not easy to visualize bagged tree because there is no single tree that can be plotted. Each of the 500 trees can be quite different from each other. In fact this is the point of bagging – to build models that can be quite different from each other so as to capture the variance. To visualize which are the important features, a variable importance plot is provided in Figure 3.13. The left plot shows the variables in order of decreasing importance based on the entropy split. The right plot shows the same based on Gini split. There are some differences between the two plots, however it is clear that S9I_1, S2D_1, S13I_1 and CONC are important features since they show up in the top five most important features in both the plots.

Variable Importance

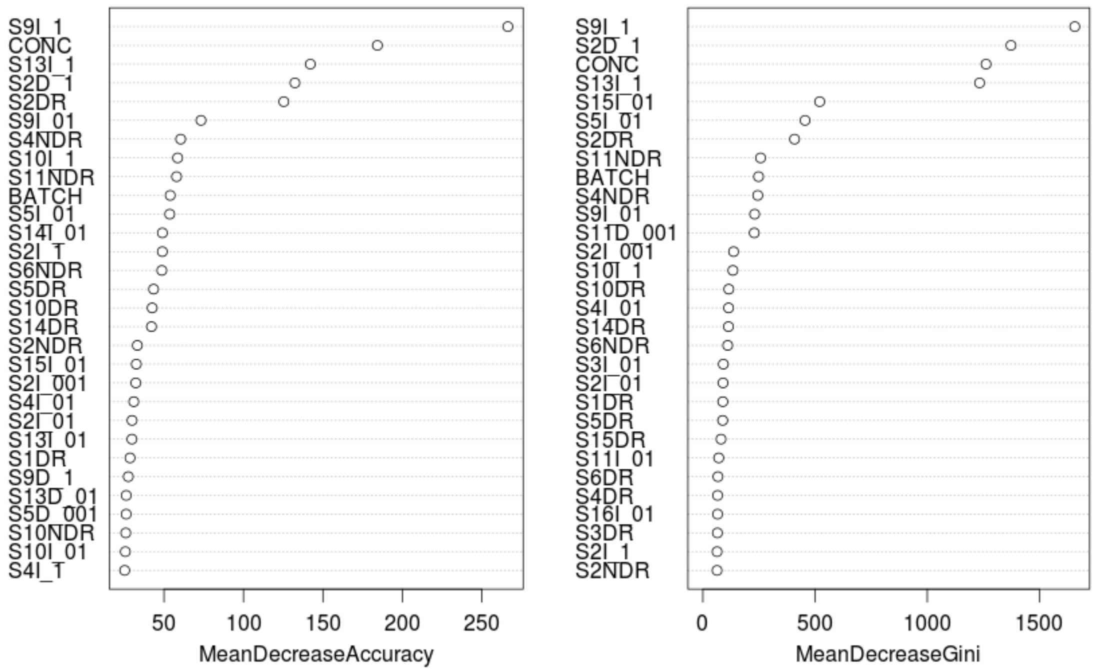


Figure 3.13 – Variable importance for bagged tree

3.3 Boosting

Though a bagged tree seems to perform remarkably well, we would like to investigate if the model can be improved further. A natural choice to consider is boosting. Unlike bagging in which individual independent decision trees are fitted to bootstrapped data, in boosting, the trees are built sequentially based on the test error of the last iteration. Boosting results in slow learning. In each iteration the tree is built using the same data set except that the misclassified observations from the last iteration are given higher weights, so that the model being built in the current iteration concentrates on learning about hard to classify observations. The boosted tree was built using 'adabag' R library. First Adaboost model is presented which uses the Breiman coefficient to update the weights. Breiman coefficient is given by

$$\alpha = \frac{1}{2} \ln \frac{1-\epsilon_t}{\epsilon_t}$$

where ϵ_t = test error. Breiman coefficient is based on the observation that population minimizer of exponential loss is one half the log odds.

Figure 3.14 shows the confusion matrix of the boosted tree built using Breiman coefficient.

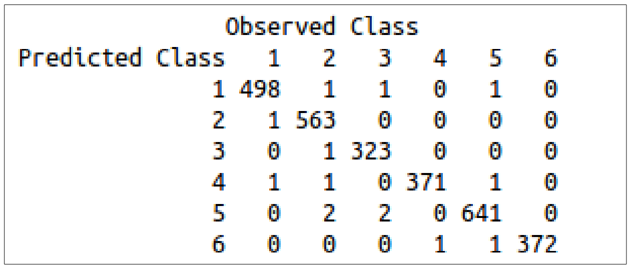

```
               Observed Class
Predicted Class   1   2   3   4   5   6
              1 498   1   1   0   1   0
              2   1 563   0   0   0   0
              3   0   1 323   0   0   0
              4   1   1   0 371   1   0
              5   0   2   2   0 641   0
              6   0   0   0   1   1 372
```

Figure 3.14 – Confusion matrix for Adaboost using Brieman coefficient

The confusion matrix clearly shows that model predicts all six gas classes with high degree of accuracy. Test error can be computed from this confusion matrix as it clearly shows how many observations are misclassified. Test error thus computed using Breiman coefficient is 0.0050 or 0.5%.

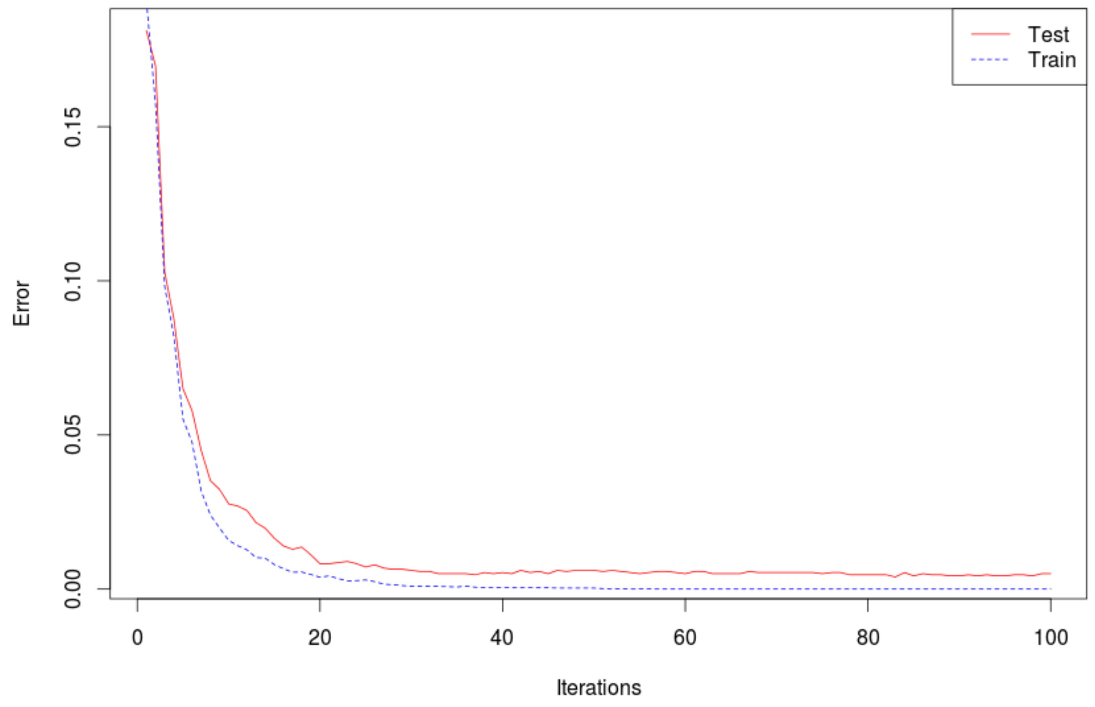


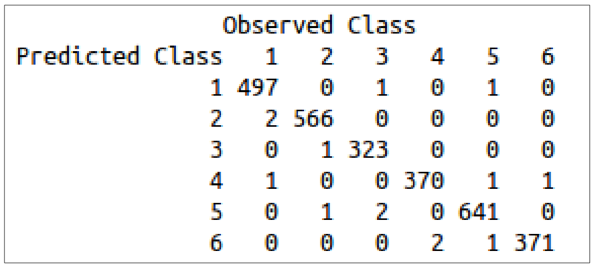
Figure 3.15 – Error evolution of Brieman coefficient based Adaboost

Figure 3.15 shows the evolution of error as a function of number of boosting iterations. Training error always stays less than the test error for the obvious reason that the model was built using training data. The interesting bit of the graph is that up to 40 iterations the error drops off rapidly as the boosting algorithm works on hard to classify observations. After 40 iterations, the gain is minimal as most of the hard to classify observations have been classified to the best of algorithm's ability.

Zhu et. al showed [16] that there can be cases where Adaboost can fail to perform well especially in multi-class scenarios. They proposed another weight updating mechanism. This mechanism uses Zhu coefficient given by -

$$\alpha = \ln \frac{1-\epsilon_t}{\epsilon_t} + \ln(K-1)$$

Zhu coefficient has the additional term ln (K – 1) where K is the number of classes. This term captures the fact that in a multi-class scenario, a classifier can be considered to perform better than random guessing if it correctly classifies the observations with an error not one-half but 1 / (K – 1). We build the boosted model using Zhu coefficient. The confusion matrix obtained from this model is shown in Figure 3.16.



```
               Observed Class
Predicted Class   1   2   3   4   5   6
              1 497   0   1   0   1   0
              2   2 566   0   0   0   0
              3   0   1 323   0   0   0
              4   1   0   0 370   1   1
              5   0   1   2   0 641   0
              6   0   0   0   2   1 371
```

Figure 3.16 – Confusion matrix for Zhu coefficient boosted model

The above confusion matrix corresponds to a test error of 0.0050 or 0.5%. We observe that this error is same as boosted model with Breiman coefficient and in the case of gas sensor array data, which coefficient is used does not seem to make a difference. This is primarily because the number of classes is small. Figure 3.17 shows test error as a function of number of iterations using Zhu coefficient. This plot looks similar to the one obtained by using Breiman coefficient except that the plot is a little wiggly in the 10-15 iteration interval.

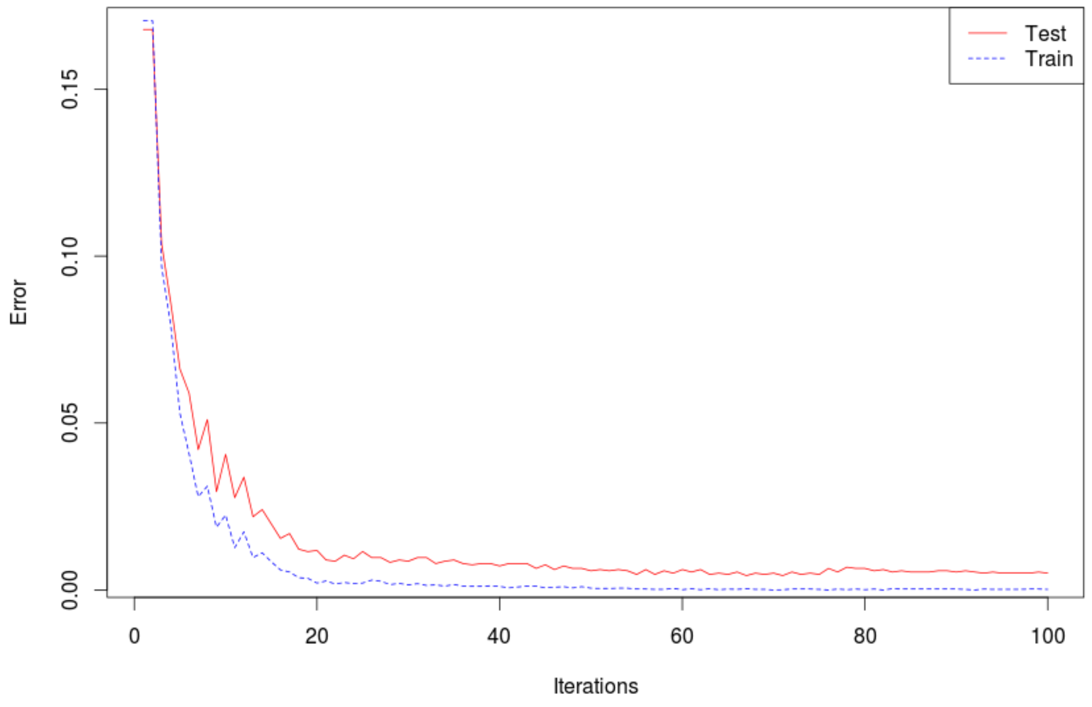
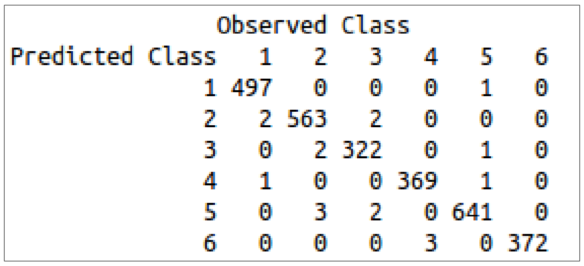


Figure 3.17 – Error evolution for boosted model using Zhu coefficient

3.4 Random Forest

Our bagged and boosted tree models have demonstrated the power of reducing variance by building several trees and then averaging the prediction. Bagging, though very effective, tends to build correlated trees. This is because bagging considers all predictors at every step in deciding the split. If there is one very strong predictor, then it is likely to be a top level split in all the trees built by bagging. This results in trees that are correlated and the reduction in variance is not as high as we would like it to be. To overcome this limitation we now build a Random Forest model. The idea of Random Forest is to consider only a small random subset of predictors at each step. This results in exclusion of very strong predictors in several models and thus allows the model to take into consideration other predictors and possibly capture more variance. As is typical of a Random Forest model we consider √p predictors (where p is the total number of predictors). For gas sensor array data, this number would be √130 ≈ 12. 'randomForest' R package was used to build this random forest model. The confusion matrix for the this model is shown in Figure 3.18.



```
                Observed Class
Predicted Class   1   2   3   4   5   6
              1 497   0   0   0   1   0
              2   2 563   2   0   0   0
              3   0   2 322   0   1   0
              4   1   0   0 369   1   0
              5   0   3   2   0 641   0
              6   0   0   0   3   0 372
```

Figure 3.18 – Random Forest confusion matrix

The above confusion matrix corresponds to a test error of 0.0064 or 0.64%. As observed with other ensemble methods, random forest model performs remarkably well.

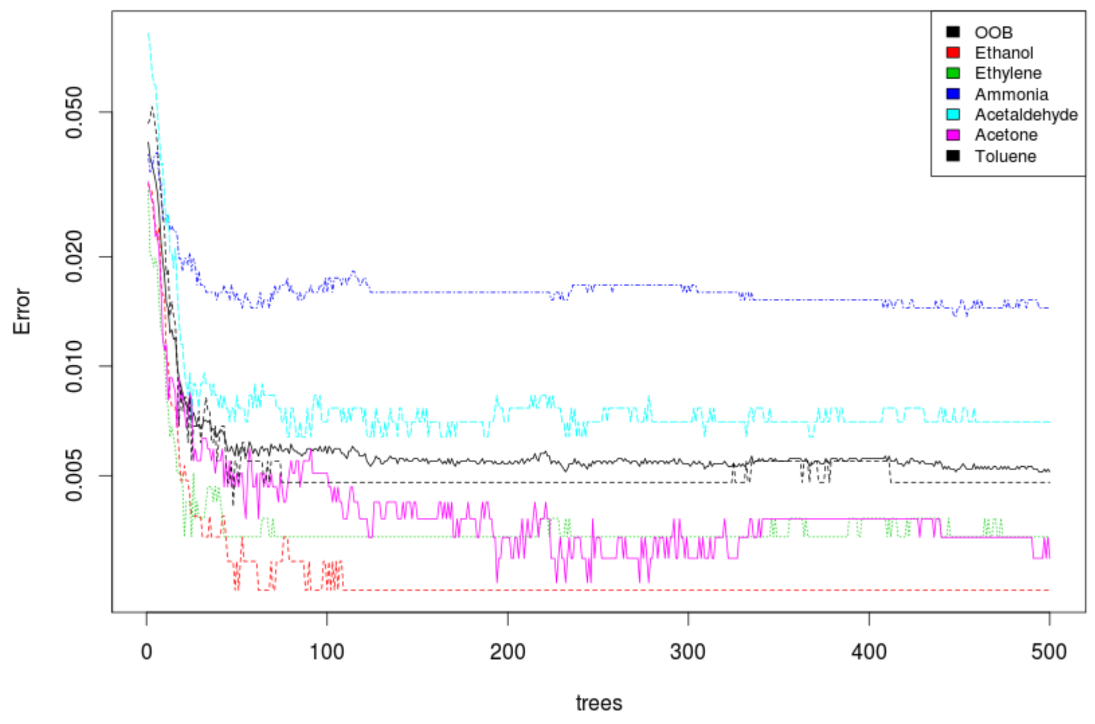


Figure 3.19 – Error vs Number of Trees for Random Forest

Figure 3.19 shows the test error as a function of number of trees considered in building the random forest. There is a marked reduction in test error up to 100 trees. Beyond that test error more or less stabilizes. It must be noted that the y- axis uses log scale.

Chapter 4: Results

This report has presented five different decision tree models - entropy based tree, Gini based tree, bagged tree, Adaboost tree and Random Forest. We now compare these models to get an estimate of how the models perform in comparison to each other.

4.1 Misclassification / Test Error Comparison

Table 4.1 shows the test error for each of these models.

Classifier	Ensemble	Misclassification Rate (%)
Entropy Tree	No	16.28
Gini Tree	No	15.49
Bagged Tree	Yes	1.1
Adaboost Tree	Yes	0.50
Random Forest	Yes	0.64

Table 4.1 – Test Error for different classifiers

It is clear that ensemble classifiers perform dramatically better than single base classifiers like entropy and Gini trees.

4.2 Receiver Operator Characteristic (ROC) Curves

Receiver Operator Characteristic (ROC) curves are plotted for each model in Figure 4.1. Table 4.2 shows the area under the corresponding curves. It is clear that the models fall into two categories. All ensemble models achieve very high true positive rate and a very low false positive rate. In fact, ROC curves for all the ensemble classifiers are very close to ideal and they all overlap enough that it is hard to distinguish them in the plot even though they have been plotted using curves of different colors. Non ensemble methods too perform fairly but clearly their ROC curves are not ideal like those of ensemble classifiers.

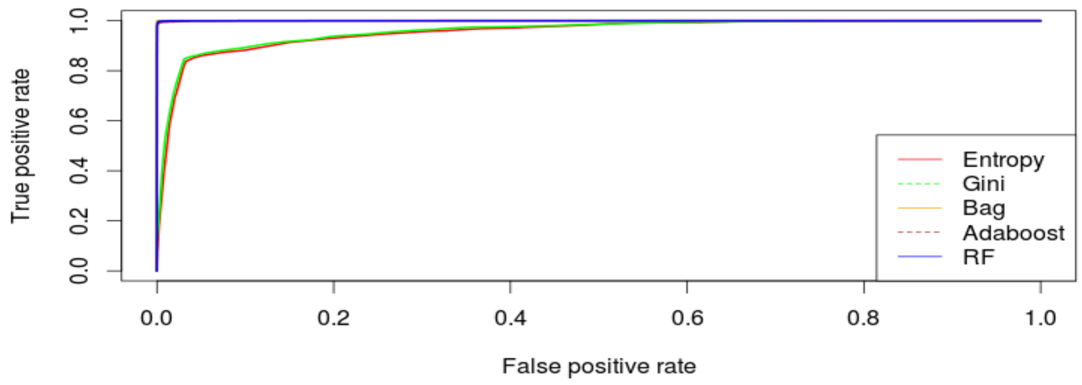


Figure 4.1 – ROC Curves

Classifier	Ensemble	Area Under ROC Curve
Entropy Tree	No	0.9548684
Gini Tree	No	0.9592732
Bagged Tree	Yes	0.9995542
Adaboost Tree	Yes	0.9995898
Random Forest	Yes	0.9999243

Table 4.2 – Area Under ROC Curves

Chapter 5: Conclusion

The work presented in this report has shown that it is possible to accurately classify gases even when the sensor data is drift prone. Decision tree modeling makes very accurate predictions on gas sensor array data. This is a reminder that decision trees are a powerful modeling tool and they must be present in the arsenal of any machine learning researcher. Their simplicity does not necessarily mean that the models built using them are sub par. By constructing ensembles, it is possible to eliminate the disadvantages of decision trees especially their sensitivity to high variance data.

This scope of this work has been limited to just six gases and few discrete levels of concentration under tightly controlled conditions. Future work may extend these models to more gases and concentrations and under more realistic conditions.

References

[1] UCI Machine Learning Repository,
https://archive.ics.uci.edu/ml/datasets/Gas+Sensor+Array+Drift+Dataset+at+Different+Concentrations

[2] A. Hierlemann, R. Gutierrez-Osuna, Higher-order chemical sensing, *ACS Chemical Reviews* 108 (2008) 563–613.

[3] J.-E. Haugen, O. Tomic, K. Kvaal, A calibration method for handling the temporal drift of solid state gas-sensors, *Analytica Chimica Acta* 407 (1–2) (2000) 23–39.

[4] T. Artusson, T. Eklöv, I. Lundström, P. Mårtensson, M. Sjöström, M. Holmberg, Drift correction for gas sensors using multivariate methods, *Journal of Chemometrics* 14 (5–6) (2000) 711–723.

[5] R. Gutierrez-Osuna, Drift reduction for metal-oxide sensor arrays using canonical correlation regression and partial least squares, *Proceedings of the 7th International Symposium On Olfaction & Electronic Nose*, 2000.

[6] A. Ziyatdinov, S. Marco, A. Chaudry, K. Persaud, P. Caminal, A. Perera, Drift compensation of gas sensor array data by common principal component analysis, *Sensors and Actuators B: Chemical* 146 (2) (2010) 460–465.

[7] Alexander Vergara , Shankar Vembu, Tuba Ayhan, Margaret A. Ryan, Margie L. Homer, Ramon Huerta - Chemical gas sensor drift compensation using classifier ensembles, *Sensors and Actuators B: Chemical* 166 (2012): 320-329

[8] Qihe Liu, Xue Li, Member, IEEE, Mao Ye, Member, IEEE, Shuzhi Sam Ge, Fellow, IEEE, and Xiaosong Du - Drift Compensation for Electronic Nose by Semi-Supervised Domain Adaption, *IEEE Sensors Journal* Vol 14, No. 3, March 2014.

[9] R. Gopalan, R. Li, and R. Chellappa, Domain adaptation for object recognition: An unsupervised approach, *Proc. ICCV*, 2011, pp. 999–1006.

[10] B. Gong, Y. Shi, F. Sha, and K. Grauman, Geodesic flow kernel for unsupervised domain adaptation, *Proc. CVPR*, 2012, pp. 2066–2073.

[11] A. Margolis, A literature review of domain adaptation with unlabeled data, Dept. Electr. Eng., Univ. Washington, Seattle, WA, USA, *Tech. Rep*., Mar. 2011.

[12] Irene Rodriguez-Lujan, Jordi Fonollosa, Alexander Vergara, Margie Homer, Ramon Huerta, On the calibration of sensor arrays for pattern recognition using the minimal number of experiments, *Chemometrics and Intelligent Laboratory Systems* 130 (2014) 123–134.

[13] Hang Liu and Zhenan Tang, Metal Oxide Gas Sensor Drift Compensation Using a Dynamic Classifier Ensemble Based on Fitting, *Sensors* 2013, 13, 9160-9173; doi:10.3390/s130709160.

[14] Eungyeong Kim, Seok Lee, Jae Hun Kim, Chulki Kim, Young Tae Byun, Hyung Seok Kim and Taikjin Lee, Pattern Recognition for Selective Odor Detection with Gas Sensor Arrays, *Sensors* 2012, 12, 16262-16273; doi:10.3390/s121216262.

[15] Daniel, D. Arul Pon, et al., ELM-Based Ensemble Classifier for Gas Sensor Array Drift Dataset, *Computational Intelligence*, *Cyber Security and Computational Models*.

Springer India, 2014. 89-96.

[16] Ji Zhu, Hui Zou, Saharon Rosset and Trevor Hastie, Multi-class AdaBoost, *Statistics and Its Interface* Volume 2 (2009) 349–360.